Escaping The Sandbox By Not Breaking It

Marco Grassi (@marcograss)

Qidan He (@flanker_hqd)

KEEN security lab

About Us

- Marco Grassi
  - Senior Security Researcher @ Tencent KEEN Lab
  - Main Focus: Vulnerability Research, Android, OS X/iOS, Sandboxes
- Qidan He
  - Senior Security Researcher @ Tencent KEEN Lab
  - Main Focus: Bug hunting and exploiting on *nix platform

KEEN security lab

Tencent KEEN Security Lab

- Previously known as KeenTeam
- All the researchers moved to Tencent for business requirements
- New name: Tencent KEEN Security Lab
- In March of this year our union team with Tencent PC Manager (Tencent Security Team Sniper) won the title of “Master Of Pwn” at Pwn2Own 2016

Agenda

- Introduction to Sandboxes
- Safari Sandbox on OS X (WebContent Sandbox)
- Google Chrome Sandbox on Android (Isolated Process)
- Comparison of the Sandbox implementation of the 2 platforms
- Auditing Sandboxes and Case Studies
- Full Sandbox Escape demo from the browser renderer process
- Summary and Conclusions

Introduction to Sandboxes

Sandbox

- In modern operating systems, a “Sandbox” is a mechanism to run code in a constrained environment.
- A Sandbox specifies which resources this code has access to.
- It became a crucial component for security in the last years after it became clear that it’s currently impossible to get rid of a big part of the bugs, especially in very complex software like browser.
- Shift of approach/complementary approach:
  - **Let’s confine software, so even if it’s compromised it has restricted access to the system.**

A couple of Sandboxes implementations methods

First type (Discretionary access control): Android base Sandbox mechanism

- Android from its initial version had a sandbox mechanism implemented mainly on top of standard Linux process isolation with unique UIDs, and GIDs specifying a capability (like access to external storage).
- Almost every application (usually, except shared UID) have a unique UID.
- Very well studied and understood by countless resources and talks, we will not talk about it a lot in this talk.
- Simpler to understand and implement, but not very flexible.

KEEN security lab

Second type (Mandatory Access Control): SELinux

- Mechanism to specify access to resources based with decision policy.
- SELinux is an example of this, you can specify which policy the process is subject to.
- When a resource is accessed, the policy is evaluated and a decision is made.
- SELinux was introduced in Android 4.3 officially and it became enforcing short after.
- Quite flexible but the policies can become very complex and difficult to understand.

WebContent Sandbox on OS X

KEEN security lab

Structure of the Safari Sandbox

- Safari code is split to run in multiple processes, based on the purpose of the code, leveraging WebKit2.
- The 2 main processes that interests us more are the Web Process and the UI Process.
- The UI Process is the parent and in charge of managing the other processes

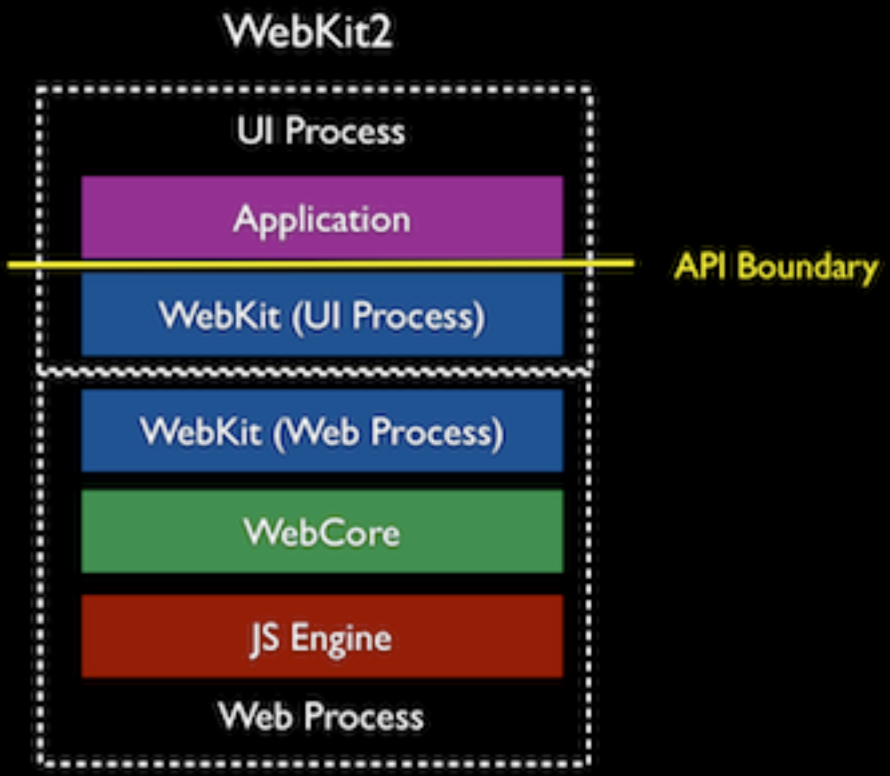


Image courtesy of: https://trac.webkit.org/attachment/wiki/WebKit2/webkit2-stack.png

WebContent process

- The WebContent process is the process responsible for handling javascript, webpages, and all the interesting stuff.
- Usually you get your initial code execution inside here, thanks to a browser bug.
- This process is heavily sandboxed, unlike his UIProcess parent.
- WebProcess can talk to UIProcess thanks to a “broker” interface, so he can request resources (such as when you have to open a file from your computer) under the supervision of the higher privileged UIProcess.

WebContent sandbox

- Regular OS X sandbox implemented on top of Sandbox.kext
- The sandbox profile definition is currently located at: *“/System/Library/Frameworks/WebKit.framework/Versions/A/Resources/com.apple.WebProcess.sb”*
- Sandbox.kext specifies callbacks for a lot of TrustedBSD MAC framework, which places hooks in the kernel where decisions has to be made, to authorize access to a resource or not (for example, on file access, the sandbox profile is used to decide if access should be granted or not)

Example profile snippets

```
(version 1)
(deny default (with partial-symbolication))
(allow system-audit file-read-metadata)

(import "system.sb")
```

Everything is denied by default

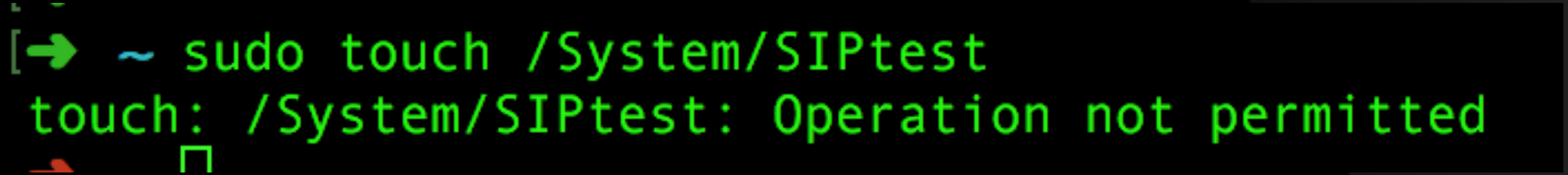
Importing “system.sb” sandbox definition file

```
(allow mach-lookup
       (global-name "com.apple.DiskArbitration.diskarbitrationd")
       (global-name "com.apple.FileCoordination")
       (global-name "com.apple.FontObjectsServer")
       (global-name "com.apple.FontServer")
```

Those particular mach services are whitelisted, their mach port can be asked

System Integrity Protection (SIP)

- In addition to those Sandboxes, on recent OS X versions you are also subject to System Integrity Protection.
- "SIP" is a security policy that applies to every process running on the system, even the root ones.
- Usermode root have not unrestricted access anymore



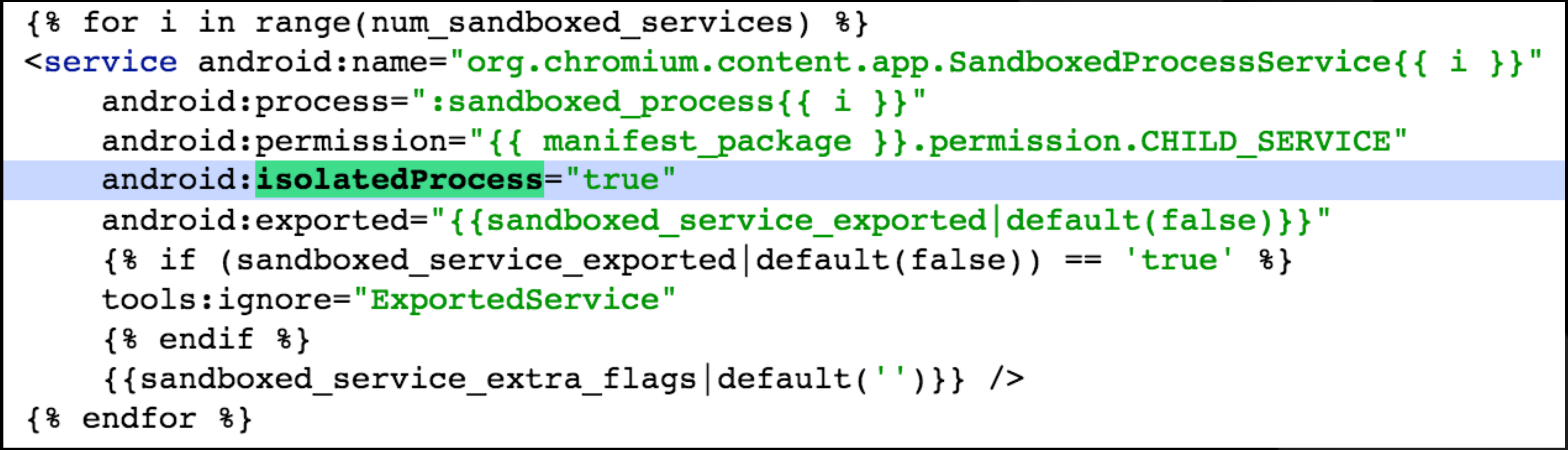
```
➜  ~ sudo touch /System/SIPtest
touch: /System/SIPtest: Operation not permitted
```

- Kernel bugs become more appealing because they allow an attacker to escape the sandbox and also disable SIP.

Google Chrome Sandbox on Android (Isolated Process)

Chromium Android Sandbox (1)

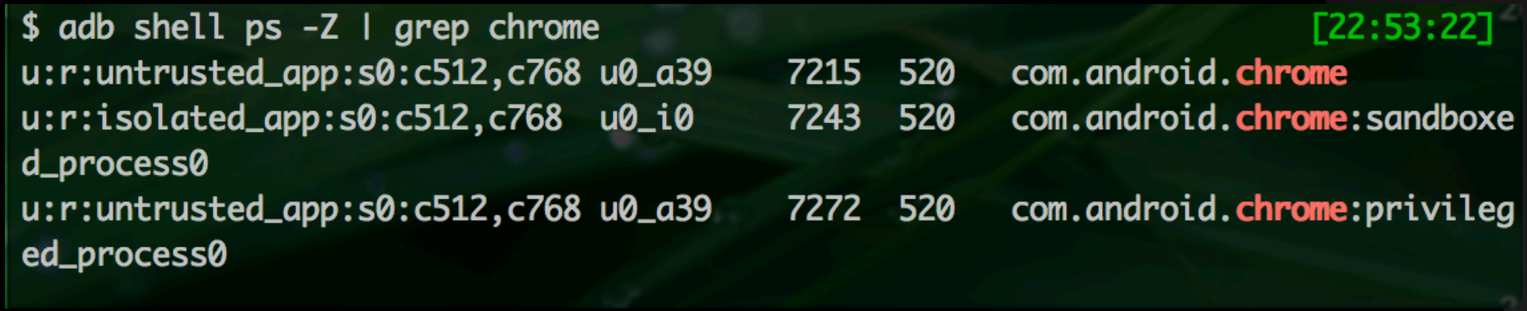
- On Android, Chromium leverages the isolatedProcess feature to implement its sandbox.



```
{% 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 (2)

- 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



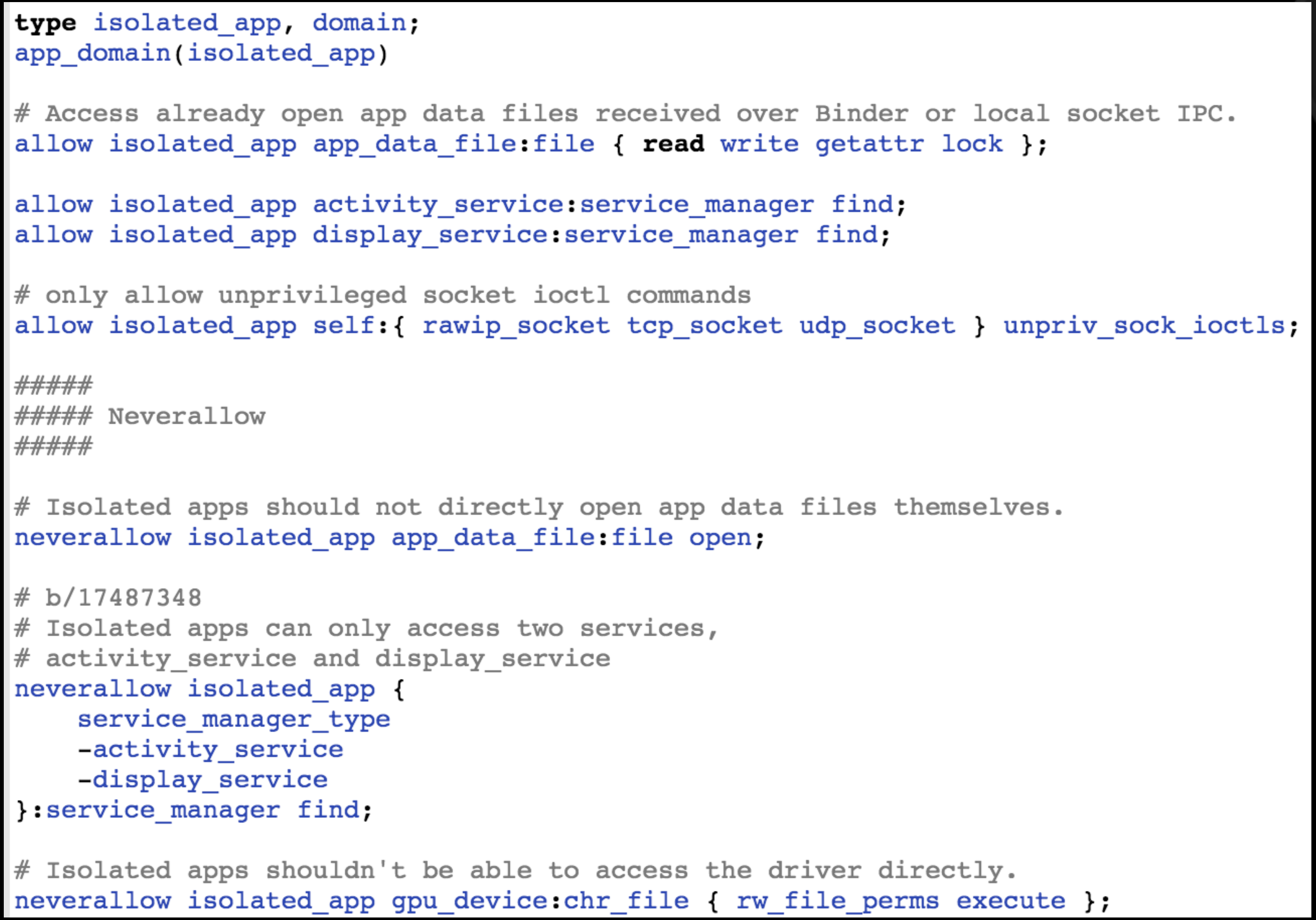
```
$ adb shell ps -Z | grep chrome                                       [22:53:22]
u:r:untrusted_app:s0:c512,c768 u0_a39    7215  520   com.android.chrome
u:r:isolated_app:s0:c512,c768  u0_i0     7243  520   com.android.chrome:sandboxe
d_process0
u:r:untrusted_app:s0:c512,c768 u0_a39    7272  520 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

Chromium Android Sandbox (3)

- So even if code execution in the render process is achieved, we don’t have a lot of capabilities, and actually we have lot of restrictions.
- In order to do something more meaningful, a sandbox escape must be chained after initial code execution.
- Usually it can be a kernel exploit, or a chromium broker exploit, or targeting another available attack surface.
- But what about SELinux? We have to check its SELinux policy, “isolated_app.te”, under external/sepolicy/ in AOSP

KEEN security lab

Chromium Android Sandbox (4)



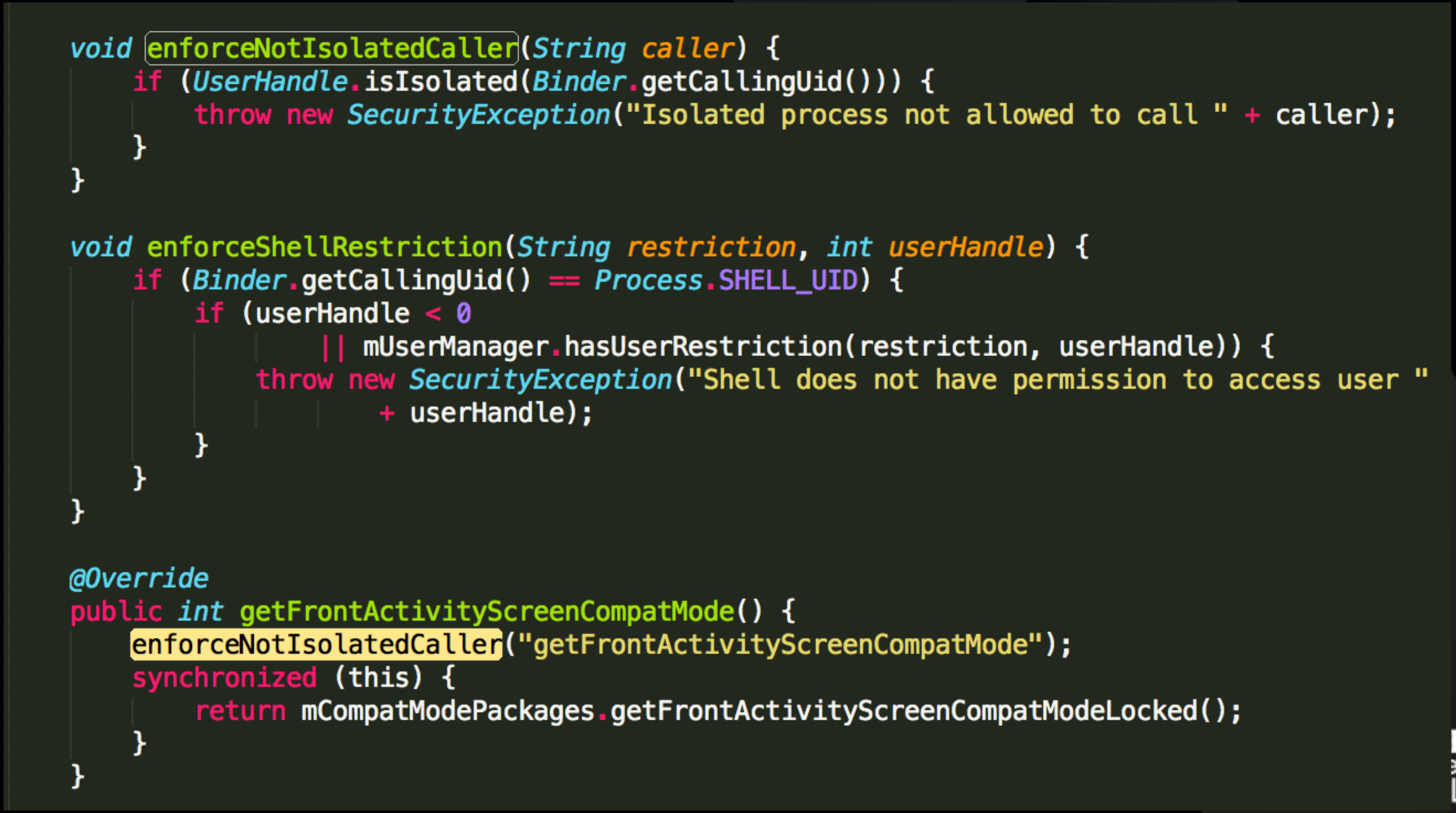
```
type isolated_app, domain;
app_domain(isolated_app)

# 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;

# only allow unprivileged socket ioctl commands
allow isolated_app self:{ rawip_socket tcp_socket udp_socket } unpriv_sock_ioctls;

#####
##### Neverallow
#####

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

# b/17487348
# Isolated apps can only access two services,
# activity_service and display_service
neverallow isolated_app {
    service_manager_type
    -activity_service
    -display_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 };
```

- Very restrictive Sandbox profile
- No data file access at all
- Only 2 IPC services
- Minimum interaction with sockets
- No graphic drivers access ☹
- ServiceManager also restricts implicit service export

KEEN security lab

Per interface constraint

- Isolated_app inherits from app_domain (app.te)
- Only interfaces without enforceNotIsolatedCaller can be invoked

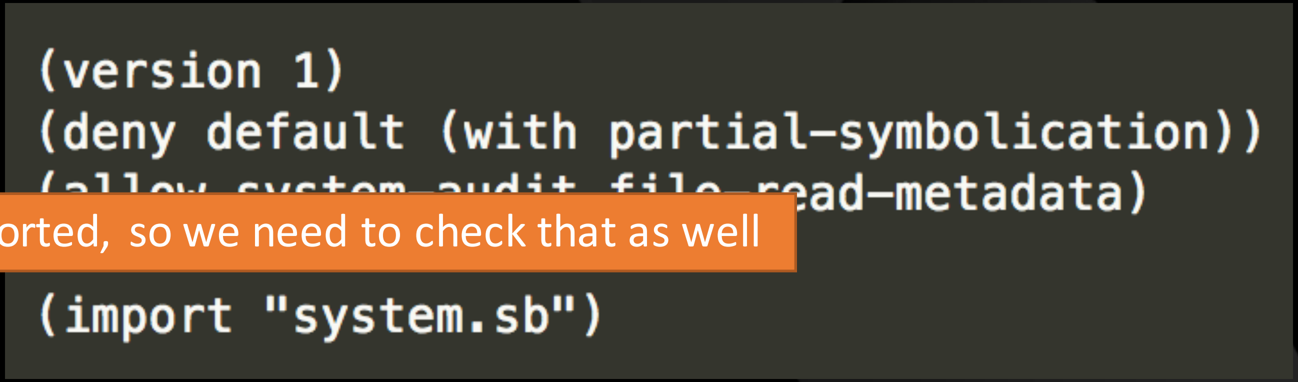


```java
void enforceNotIsolatedCaller(String caller) {
    if (UserHandle.isIsolated(Binder.getCallingUid())) {
        throw new SecurityException("Isolated process not allowed to call " + caller);
    }
}

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();
    }
}
```

Auditing and Case Studies

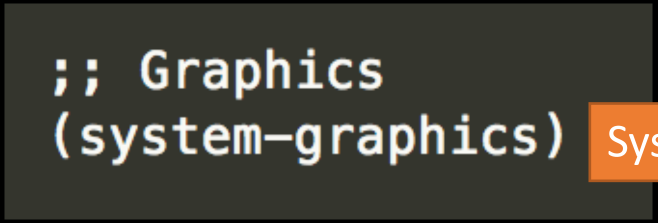
How to audit a sandbox profile?

- Just look at the definitions and see what attack surfaces are allowed!
- We will try with the WebContent sandbox on OS X.



```
(version 1)
(deny default (with partial-symbolication))
(allow system-audit file-read-metadata)

(import "system.sb")
```

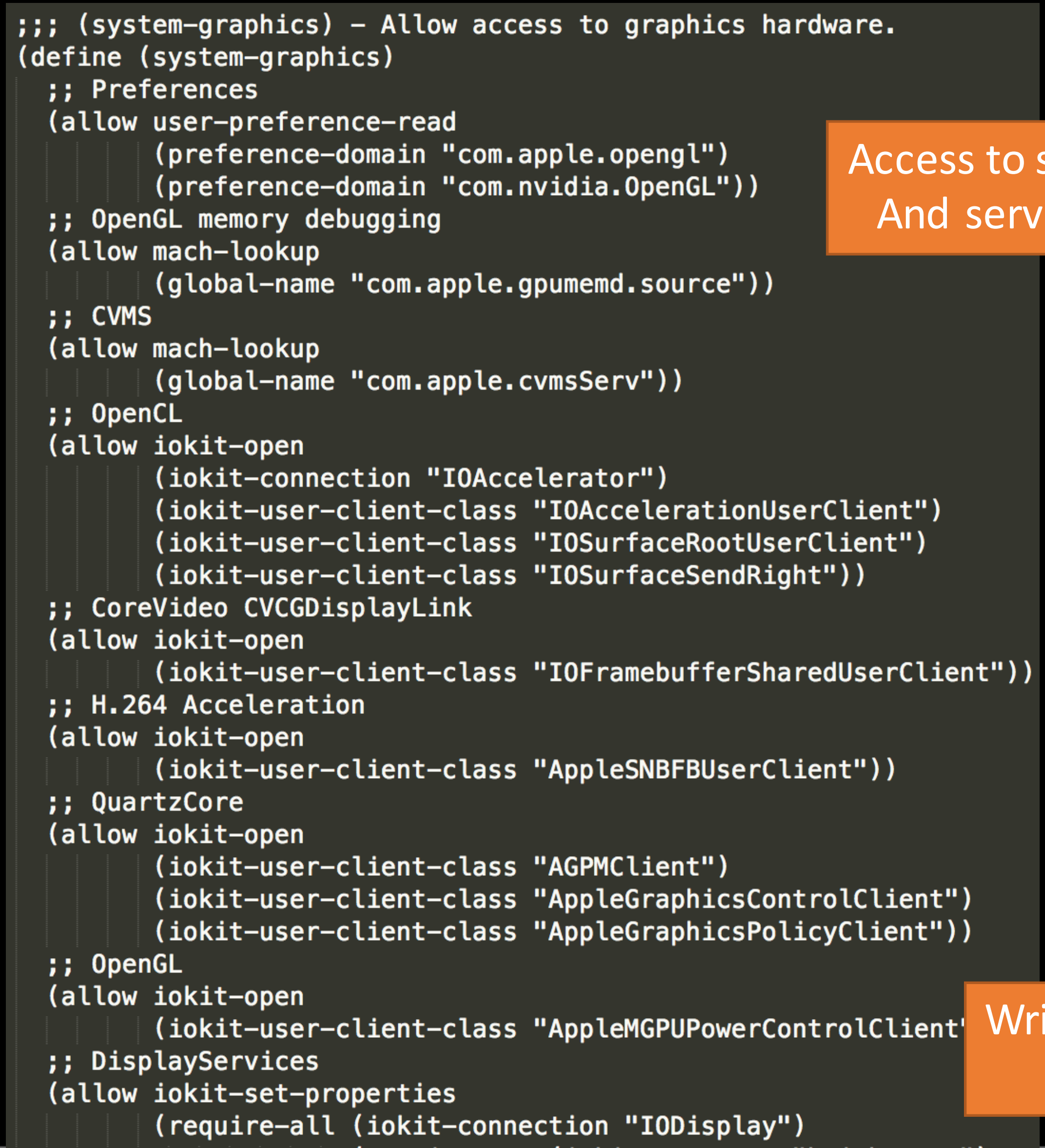
system.sb is imported, so we need to check that as well



```
;; Graphics
(system-graphics)
```

System-graphics is defined in system.sb, let’s check it

How to audit a sandbox profile? (2)



```
;;; (system-graphics) - Allow access to graphics hardware.
(define (system-graphics)
  ;; Preferences
  (allow user-preference-read
         (preference-domain "com.apple.opengl")
         (preference-domain "com.nvidia.OpenGL"))
  ;; OpenGL memory debugging
  (allow mach-lookup
         (global-name "com.apple.gpumemd.source"))
  ;; CVMS
  (allow mach-lookup
         (global-name "com.apple.cvmsServ"))
  ;; OpenCL
  (allow iokit-open
         (iokit-connection "IOAccelerator")
         (iokit-user-client-class "IOAccelerationUserClient")
         (iokit-user-client-class "IOSurfaceRootUserClient")
         (iokit-user-client-class "IOSurfaceSendRight"))
  ;; CoreVideo CVCGDisplayLink
  (allow iokit-open
         (iokit-user-client-class "IOFramebufferSharedUserClient"))
  ;; H.264 Acceleration
  (allow iokit-open
         (iokit-user-client-class "AppleSNBFBUserClient"))
  ;; QuartzCore
  (allow iokit-open
         (iokit-user-client-class "AGPMClient")
         (iokit-user-client-class "AppleGraphicsControlClient")
         (iokit-user-client-class "AppleGraphicsPolicyClient"))
  ;; OpenGL
  (allow iokit-open
         (iokit-user-client-class "AppleMGPUPowerControlClient'
  ;; DisplayServices
  (allow iokit-set-properties
         (require-all (iokit-connection "IODisplay")
```

Access to several IOKit User clients
And services related to graphics

Graphics seems definetely a nice attack surface,
Now we can start finding vulnerabilities in those IOKit clients by fuzzing or manual auditing, since we can interact with them from the WebContent process, where we have initial code execution, to escape the sandbox, getting kernel code execution.

Write access to several iokit properties related to graphics

How to audit a sandbox profile? (3)

- “allow-iokit-open” (iokit-connection "IOAccelerator") is definetely interesting
- iokit-connection allows the sandboxed process to open all the userclient under the target IOService(much less restrictive than iokit-user-client-class)
- In the table on the left we see the Userclients that we can obtain on the IntelAccelerator (default driver in most of the recent Apple machines)

UserClient Name	Type
IGAccelSurface	0
IGAccelGLContext	1
IGAccel2DContext	2
IOAccelDisplayPipeUserClient2	4
IGAccelSharedUserClient	5
IGAccelDevice	6
IOAccelMemoryInfoUserClient	7
IGAccelCLContext	8
IGAccelCommandQueue	9
IGAccelVideoContext	0x100

IOKit vulnerability: CVE-2016-1744

- Race condition in an externalMethod in **AppleIntelBDWGraphics**.
- Affects every recent Mac with Intel Broadwell CPU/Graphics.
- Discovered by code auditing when looking for sandbox escapes into IOKit UserClients reachable from the Safari WebProcess sandbox.
- Unfortunately it got partially patched 1-2 weeks before Pwn2Own! ☹☹☹ . A replacement was needed. ☹
- Unpatched in OSX 10.11.3, only partial fix in 10.11.4 beta6.
- Reliably exploitable.
- Finally it came out that we had a bug collision with Ian Beer of Google Project Zero, which reported the bug to Apple.

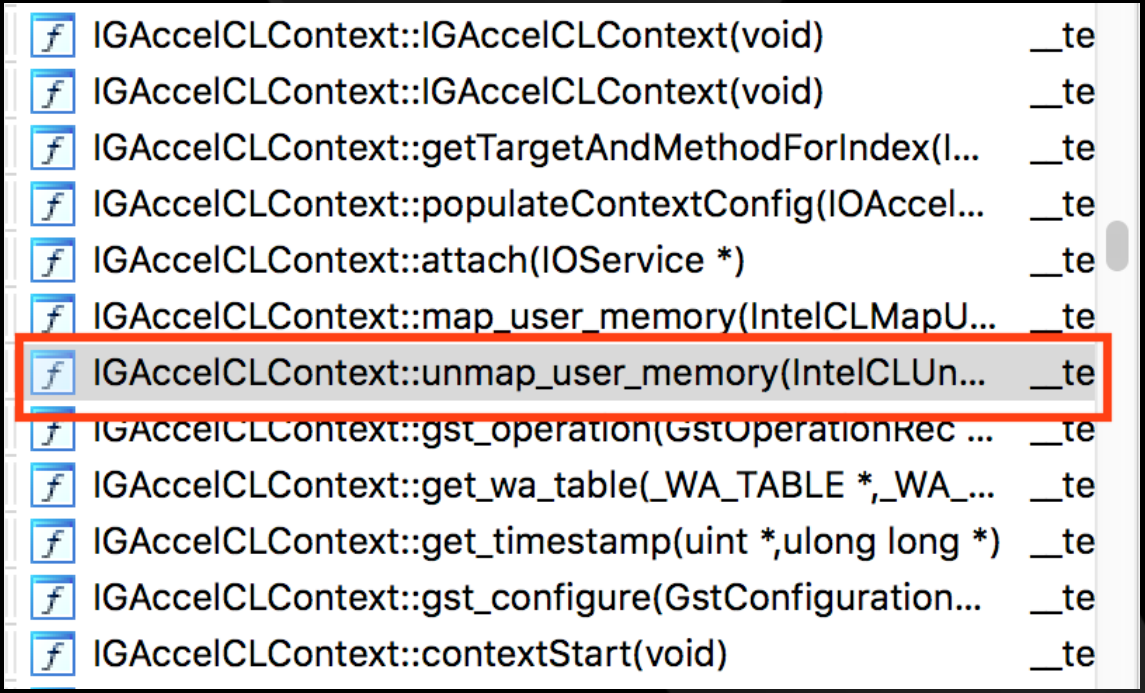
KEEN security lab

IOKit Vulnerability – CVE-2016-1744(cont.)

- Wrong/partial fix mistake responsibly disclosed to Apple.
    - Fixed in 10.11.5 beta2
    - CVE-2016-1860

IOKit vulnerability: CVE-2016-1744

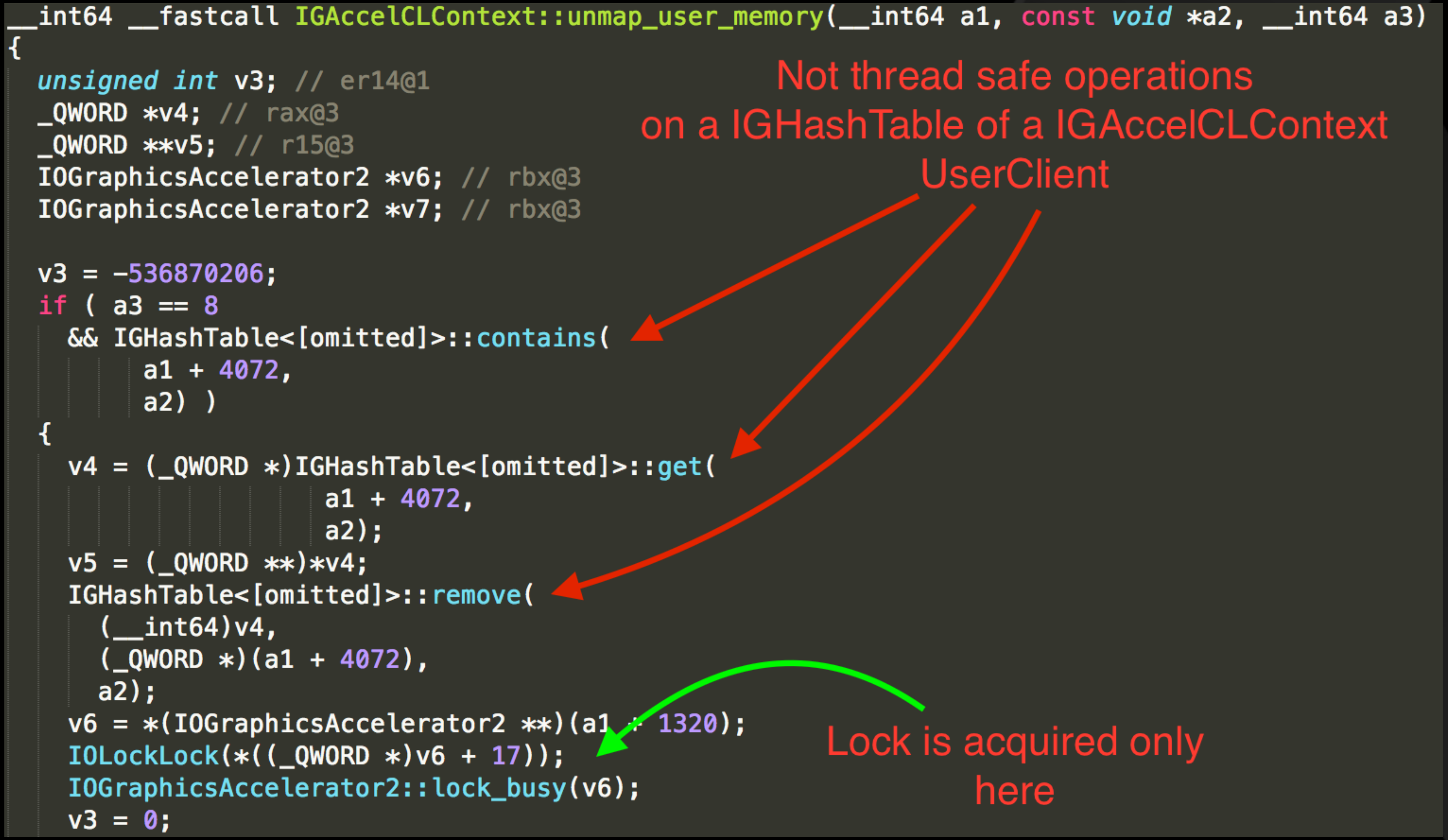
- IGAccelCLContext and IGAccelGLContext are 2 UserClients that can be reached from the WebProcess Safari sandbox.
- The locking mechanisms in these UserClients is not too good, some methods expects only a well behaved single threaded access.
- First we targeted **unmap_user_memory**



```
IGAccelCLContext::IGAccelCLContext(void)                        __te
IGAccelCLContext::IGAccelCLContext(void)                        __te
IGAccelCLContext::getTargetAndMethodForIndex(I...               __te
IGAccelCLContext::populateContextConfig(IOAccel...              __te
IGAccelCLContext::attach(IOService *)                           __te
IGAccelCLContext::map_user_memory(IntelCLMapU...                __te
IGAccelCLContext::unmap_user_memory(IntelCLUn...                __te
IGAccelCLContext::gst_operation(GstOperationRec ...             __te
IGAccelCLContext::get_wa_table(_WA_TABLE *,_WA_...              __te
IGAccelCLContext::get_timestamp(uint *,ulong long *)            __te
IGAccelCLContext::gst_configure(GstConfiguration...             __te
IGAccelCLContext::contextStart(void)                            __te
```

KEEN security lab

IOKit vulnerability: some unsafe code



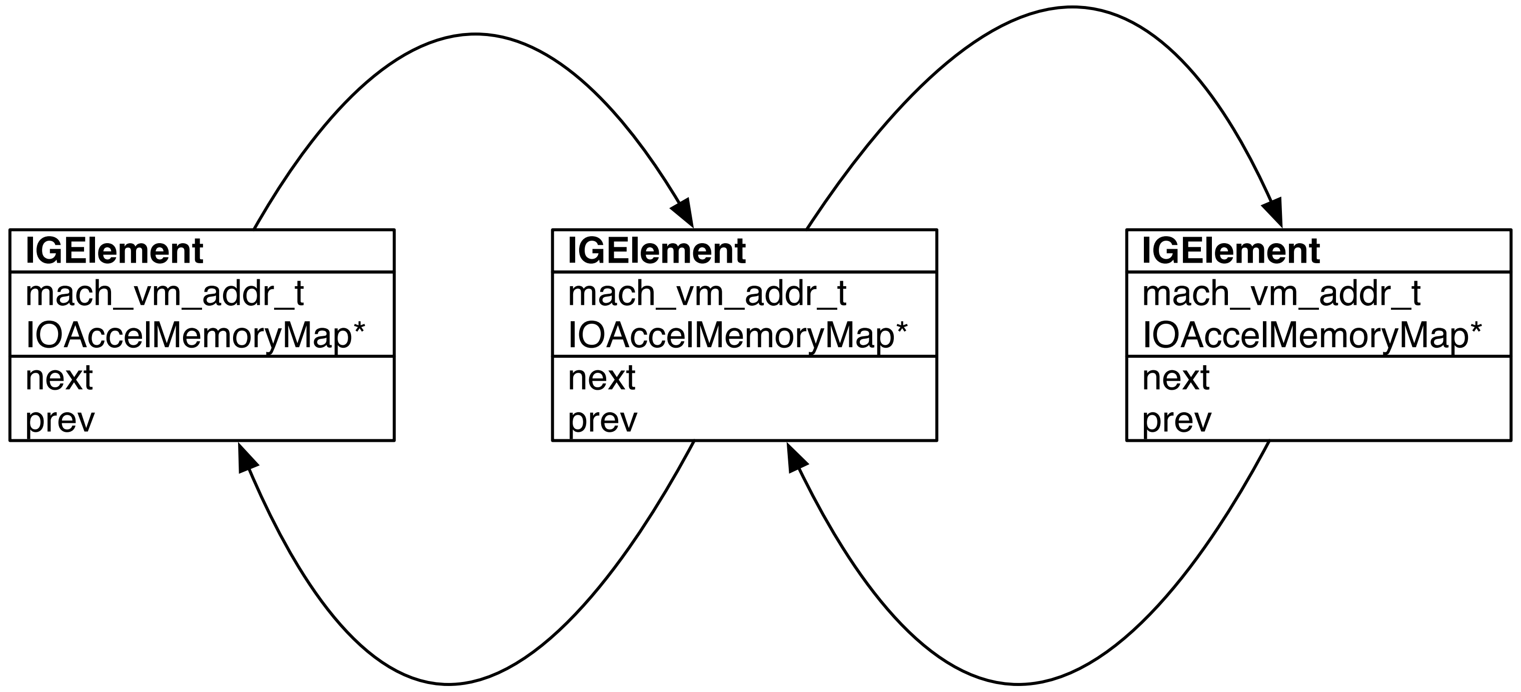
```
__int64 __fastcall IGAccelCLContext::unmap_user_memory(__int64 a1, const void *a2, __int64 a3)
{
  unsigned int v3; // er14@1
  _QWORD *v4; // rax@3
  _QWORD **v5; // r15@3
  IOGraphicsAccelerator2 *v6; // rbx@3
  IOGraphicsAccelerator2 *v7; // rbx@3

  v3 = -536870206;
  if ( a3 == 8
    && IGHashTable<[omitted]>::contains(
         a1 + 4072,
         a2) )
  {
    v4 = (_QWORD *)IGHashTable<[omitted]>::get(
                     a1 + 4072,
                     a2);
    v5 = (_QWORD **)*v4;
    IGHashTable<[omitted]>::remove(
      (__int64)v4,
      (_QWORD *)(a1 + 4072),
      a2);
    v6 = *(IOGraphicsAccelerator2 **)(a1 + 1320);
    IOLockLock(*((_QWORD *)v6 + 17));
    IOGraphicsAccelerator2::lock_busy(v6);
    v3 = 0;
```

Not thread safe operations
on a IGHashTable of a IGAccelCLContext
UserClient
Lock is acquired only
here

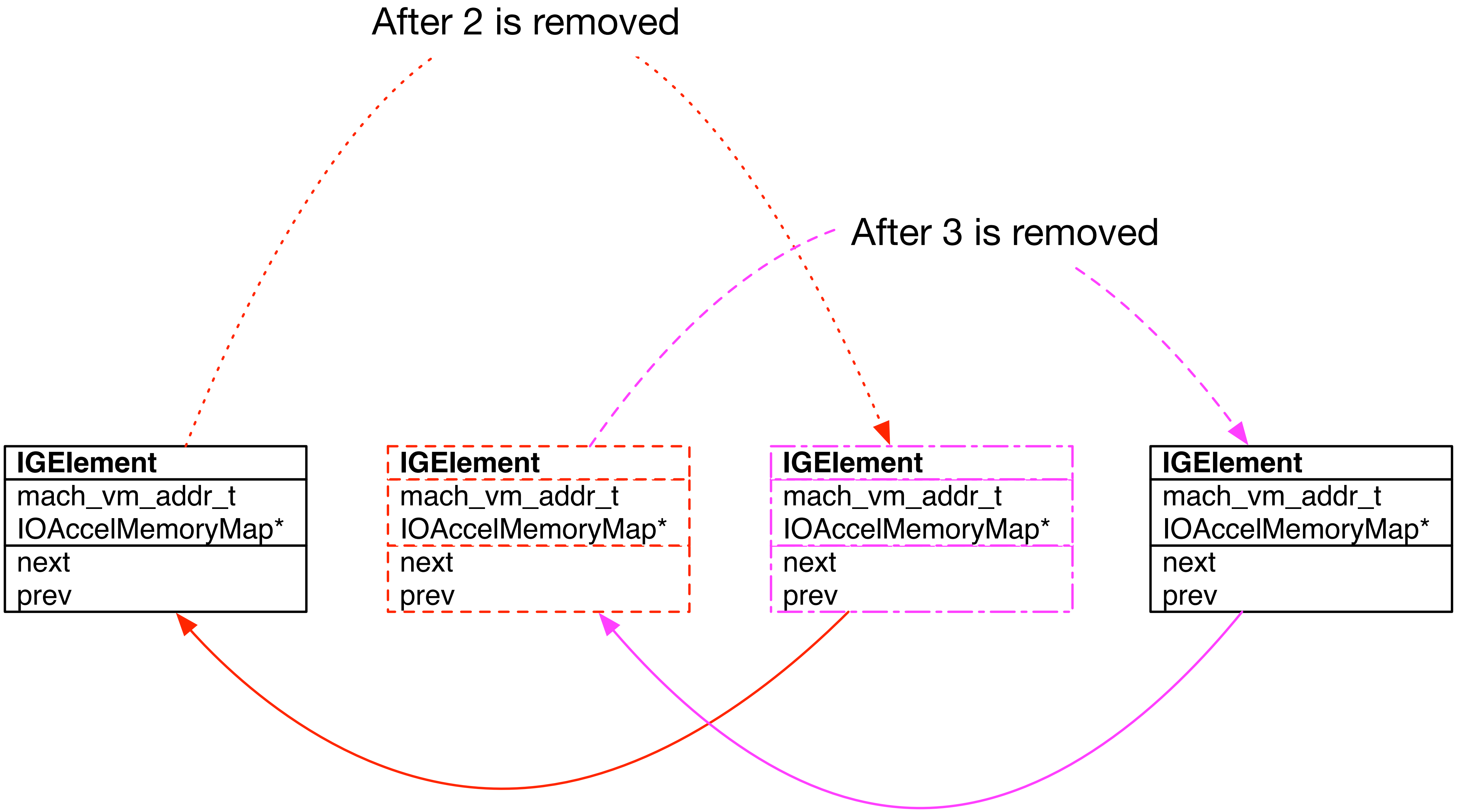
Race condition – How to trigger it?

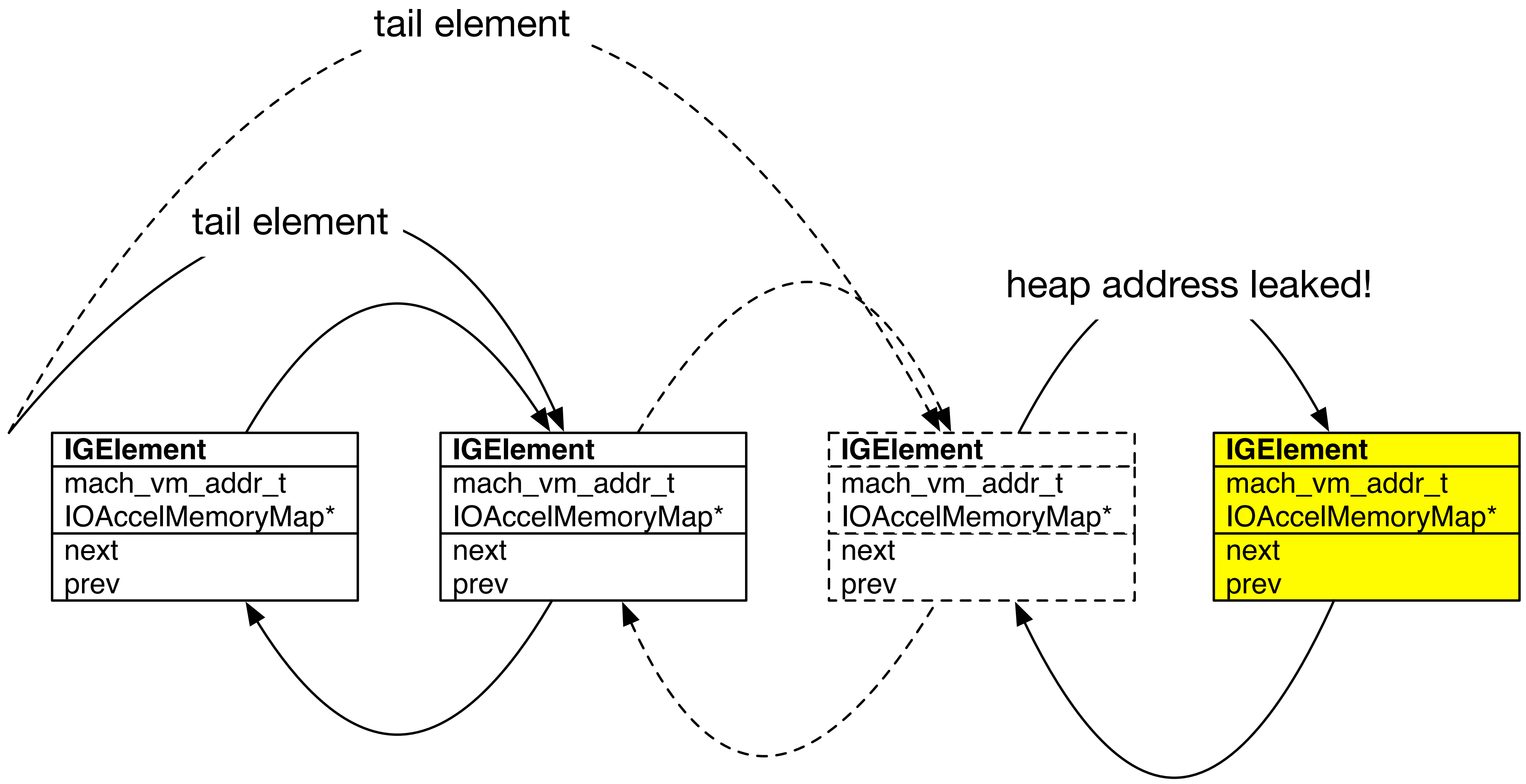
1. Open your target UserClient (IGAccelCLContext)
2. Call **map_user_memory** to insert one element into the IGHashTable
3. Call with 2 racing threads **unmap_user_memory.**
4. Repeat 2 and 3 until you are able to exploit the race window.
5. Double free on first hand
6. PROFIT!

The ideal situation is both threads passes hash table::contains, and when one is retrieving IOAccelMemoryMap* after get returns valid pointer, the other frees it and we control the pointer



However in reality more frequently they do passes contains
but thread 1 will remove it before thread 2 do get
and thread 2 hit a null pointer dereference


After 2 is removed
After 3 is removed
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev


tail element
tail element
heap address leaked!
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev
IGElement
mach_vm_addr_t
IOAccelMemoryMap*
next
prev

For further info, check our talk slides

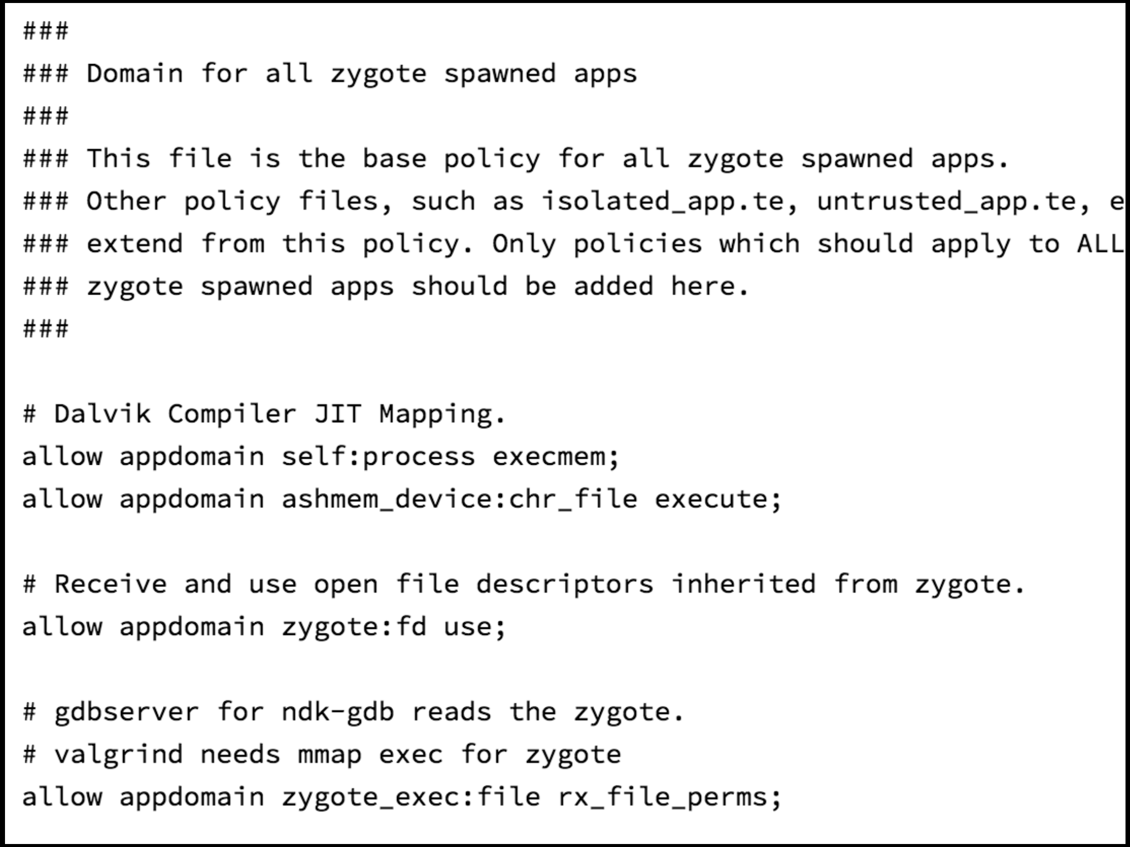
"Don't Trust Your Eye: Apple Graphics Is Compromised!"
http://bit.ly/23GR14N

The Python bites your apple: fuzzing and exploiting OSX kernel bugs
https://goo.gl/Ccgni1

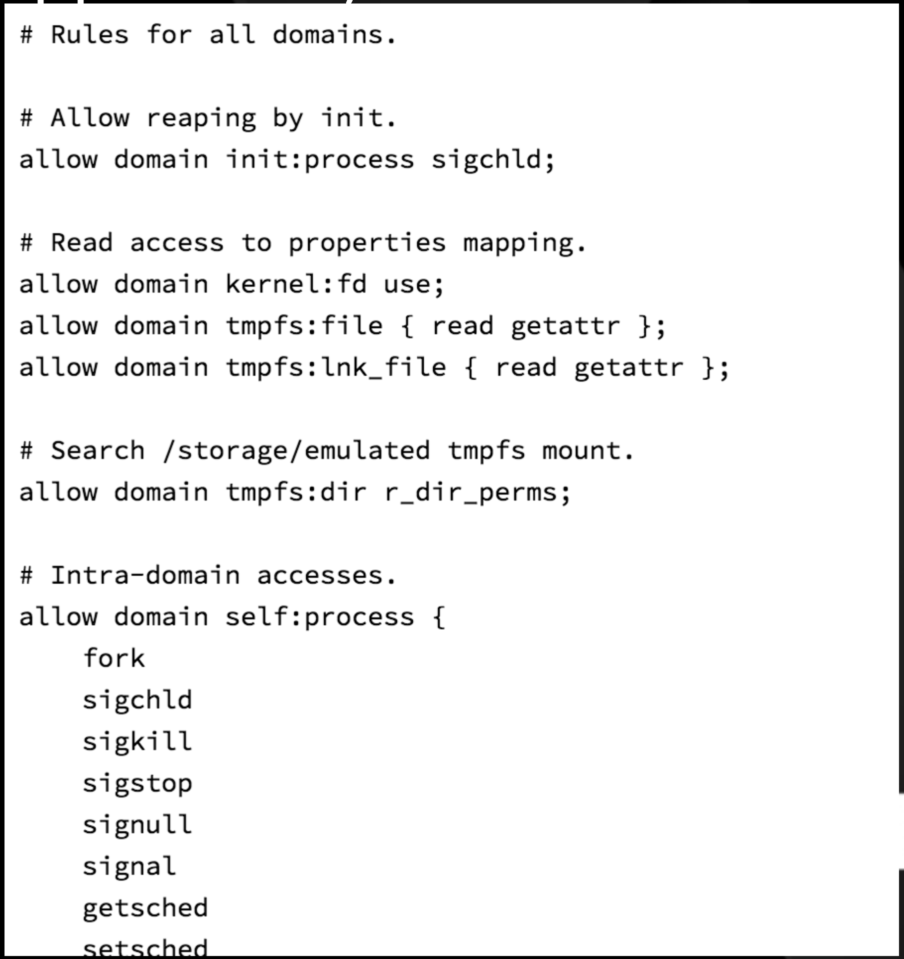
KEEN security lab

For Android Sandbox Escape

- Isolated_app inherits app.te, app.te (appdomain) inherits domain.te



```
###
### Domain for all zygote spawned apps
###
### This file is the base policy for all zygote spawned apps.
### Other policy files, such as isolated_app.te, untrusted_app.te, e
### extend from this policy. Only policies which should apply to ALL
### zygote spawned apps should be added here.
###

# Dalvik Compiler JIT Mapping.
allow appdomain self:process execmem;
allow appdomain ashmem_device:chr_file execute;

# Receive and use open file descriptors inherited from zygote.
allow appdomain zygote:fd use;

# gdbserver for ndk-gdb reads the zygote.
# valgrind needs mmap exec for zygote
allow appdomain zygote_exec:file rx_file_perms;
```



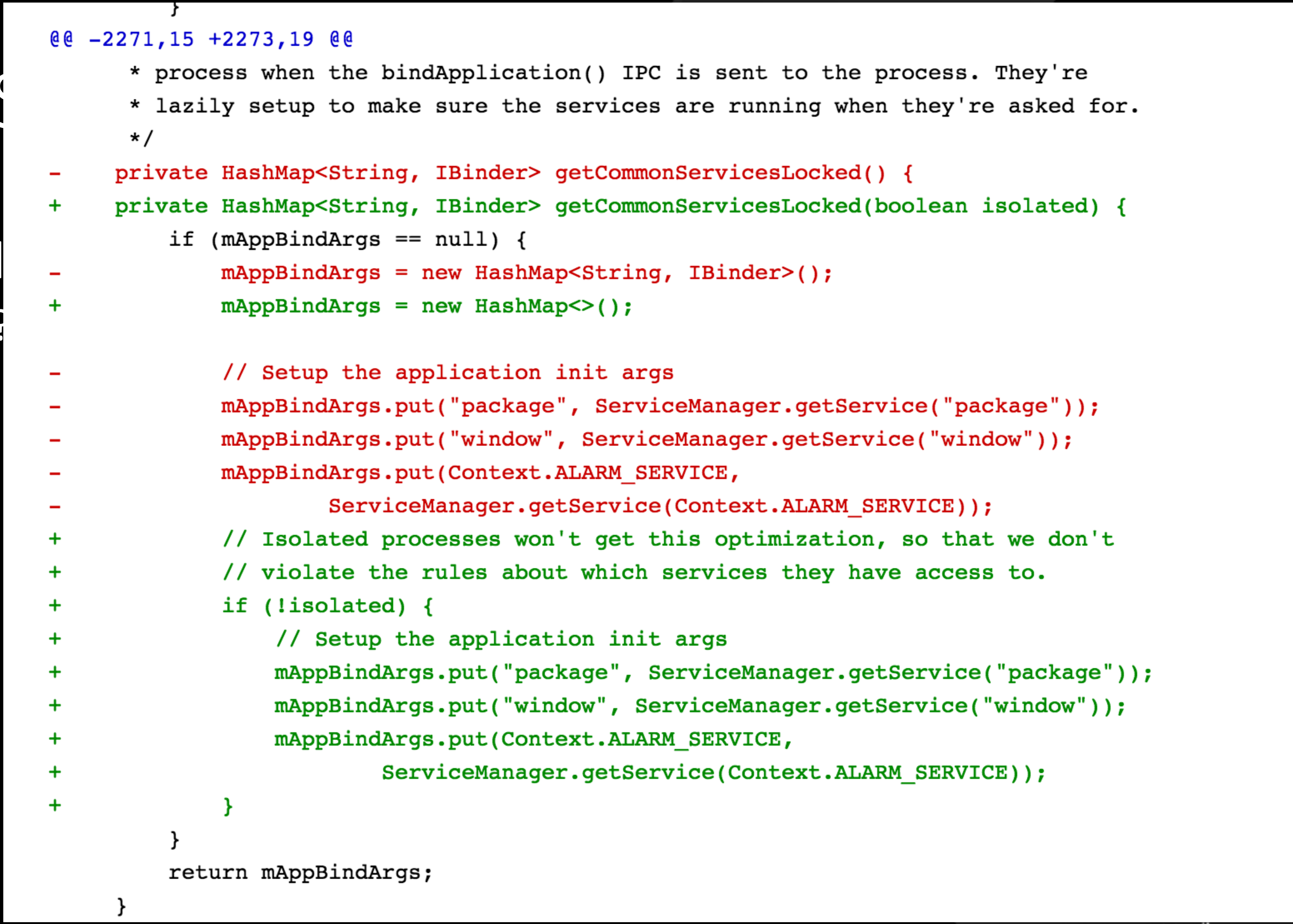
```
# Rules for all domains.

# Allow reaping by init.
allow domain init:process sigchld;

# Read access to properties mapping.
allow domain kernel:fd use;
allow domain tmpfs:file { read getattr };
allow domain tmpfs:lnk_file { read getattr };

# Search /storage/emulated tmpfs mount.
allow domain tmpfs:dir r_dir_perms;

# Intra-domain accesses.
allow domain self:process {
    fork
    sigchld
    sigkill
    sigstop
    signull
    signal
    getsched
    setsched
```

For Android Sandbox Escape(cont.)

- Attacking the binder interface is still an option
    - Exploiting vulnerable basic classes
        - SharedStorage integer overflow
        - CVE-2015-3875
- Parcel at Java level accepts deserialization on class name specified by string when processing bundle
    - A hidden path to trigger de/serialization code in system_server context

For Android S

- Attacking the bind
  - Unintend-export?



```diff
        }
@@ -2271,15 +2273,19 @@
      * process when the bindApplication() IPC is sent to the process. They're
      * lazily setup to make sure the services are running when they're asked for.
      */
-    private HashMap<String, IBinder> getCommonServicesLocked() {
+    private HashMap<String, IBinder> getCommonServicesLocked(boolean isolated) {
         if (mAppBindArgs == null) {
-            mAppBindArgs = new HashMap<String, IBinder>();
+            mAppBindArgs = new HashMap<>();

-            // Setup the application init args
-            mAppBindArgs.put("package", ServiceManager.getService("package"));
-            mAppBindArgs.put("window", ServiceManager.getService("window"));
-            mAppBindArgs.put(Context.ALARM_SERVICE,
-                    ServiceManager.getService(Context.ALARM_SERVICE));
+            // Isolated processes won't get this optimization, so that we don't
+            // violate the rules about which services they have access to.
+            if (!isolated) {
+                // Setup the application init args
+                mAppBindArgs.put("package", ServiceManager.getService("package"));
+                mAppBindArgs.put("window", ServiceManager.getService("window"));
+                mAppBindArgs.put(Context.ALARM_SERVICE,
+                        ServiceManager.getService(Context.ALARM_SERVICE));
+            }
         }
         return mAppBindArgs;
     }
```

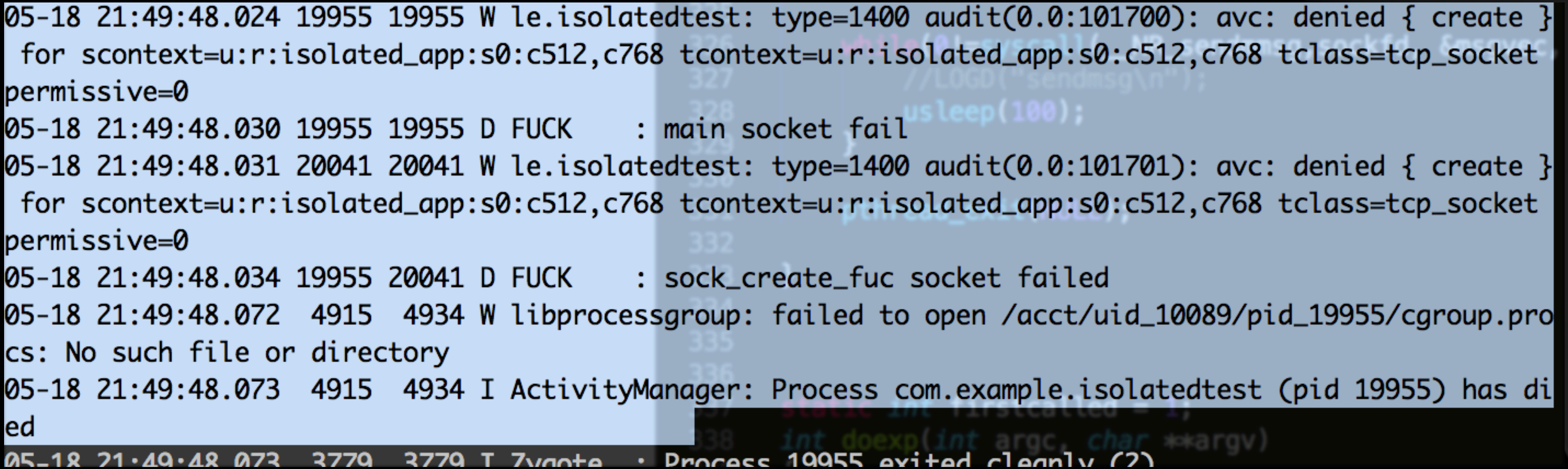
security lab

For Android Sandbox Escape(cont.)

- Attacking the binder interface is still an option
  - Exploiting vulnerable basic classes/ reachable via bundle interfaces
    - SharedStorage integer overflow
- Attacking the Chrome IPC
- Attacking WebGL
  - GL process runs in host process in Android
- Attacking the Kernel
  - CVE-2015-1805?

1805 in action

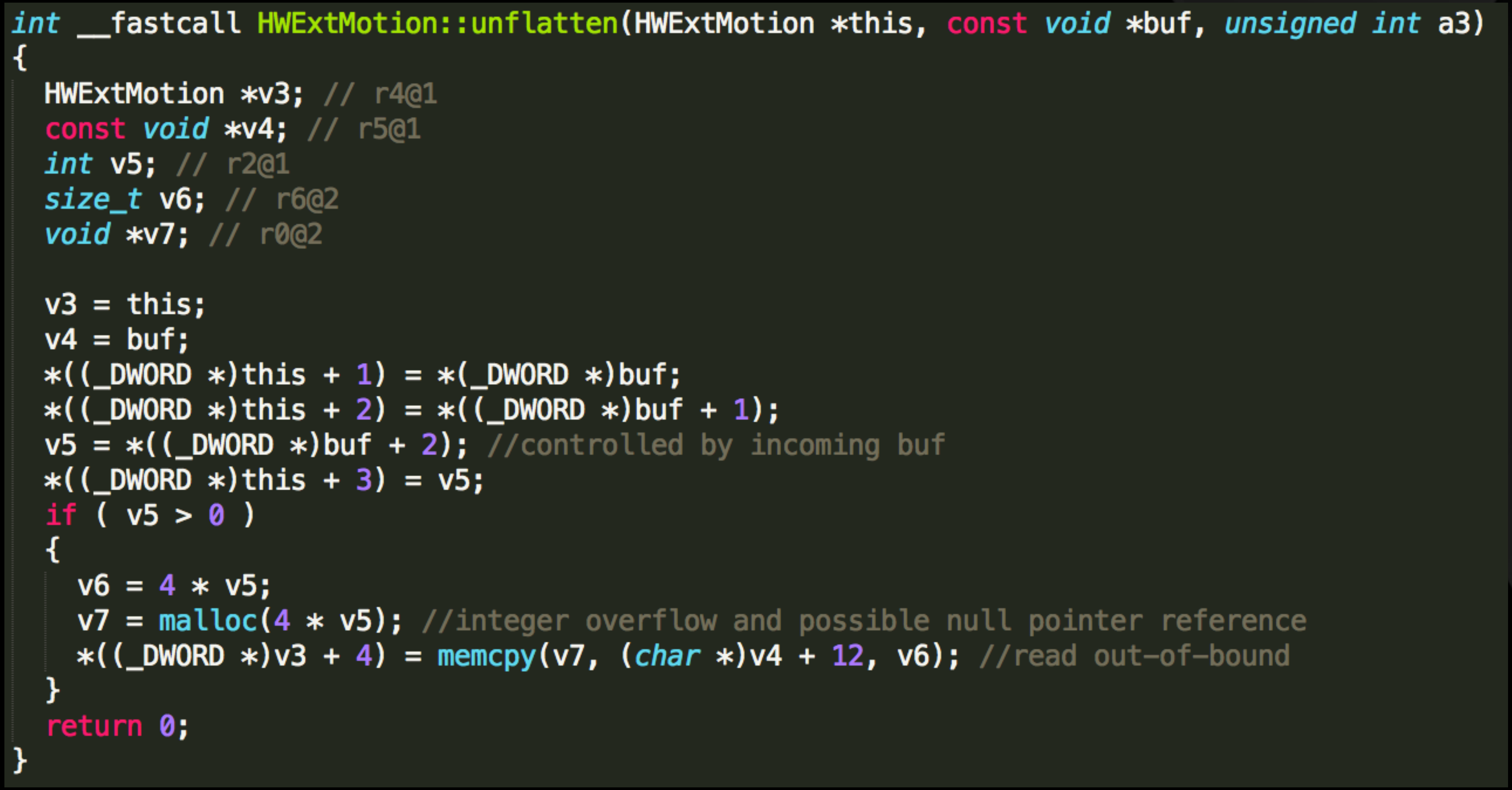
- Good news
  - No pipe policy in isolated_app
- Bad news:
  - Cannot create socket and spray kernel memory use sendmmsg ☹



```
05-18 21:49:48.024 19955 19955 W le.isolatedtest: type=1400 audit(0.0:101700): avc: denied { create }
 for scontext=u:r:isolated_app:s0:c512,c768 tcontext=u:r:isolated_app:s0:c512,c768 tclass=tcp_socket
permissive=0
05-18 21:49:48.030 19955 19955 D FUCK    : main socket fail
05-18 21:49:48.031 20041 20041 W le.isolatedtest: type=1400 audit(0.0:101701): avc: denied { create }
 for scontext=u:r:isolated_app:s0:c512,c768 tcontext=u:r:isolated_app:s0:c512,c768 tclass=tcp_socket
permissive=0
05-18 21:49:48.034 19955 20041 D FUCK    : sock_create_fuc socket failed
05-18 21:49:48.072  4915  4934 W libprocessgroup: failed to open /acct/uid_10089/pid_19955/cgroup.pro
cs: No such file or directory
05-18 21:49:48.073  4915  4934 I ActivityManager: Process com.example.isolatedtest (pid 19955) has di
ed
05-18 21:49:48.073  3779  3779 I Zygote   : Process 19955 exited cleanly (2)
```

Prevent vendor’s binder mistake

- Integer overflow in Huawei hw_ext_service running in system_server



```
int __fastcall HWExtMotion::unflatten(HWExtMotion *this, const void *buf, unsigned int a3)
{
  HWExtMotion *v3; // r4@1
  const void *v4; // r5@1
  int v5; // r2@1
  size_t v6; // r6@2
  void *v7; // r0@2

  v3 = this;
  v4 = buf;
  *((_DWORD *)this + 1) = *(_DWORD *)buf;
  *((_DWORD *)this + 2) = *((_DWORD *)buf + 1);
  v5 = *((_DWORD *)buf + 2); //controlled by incoming buf
  *((_DWORD *)this + 3) = v5;
  if ( v5 > 0 )
  {
    v6 = 4 * v5;
    v7 = malloc(4 * v5); //integer overflow and possible null pointer reference
    *((_DWORD *)v3 + 4) = memcpy(v7, (char *)v4 + 12, v6); //read out-of-bound
  }
  return 0;
}
```

Comparison

Comparison

- Both platforms share lot of traits. They both implement a sandbox policy in files that specify it and can be audited
- In general between the 2, the Chromium Android sandbox feels stronger because it exposes a smaller attack surface.
- On Android we have more layer of sandboxing:
  - Android sandbox, chrome is an application, it’s restricted by its DAC sandbox
  - IsolatedProcess, the render processes run in their own unprivileged process
  - Restrictive SELinux policy isolated_app.te

Full Sandbox Escape DEMO!

KEEN security lab

Summary and Conclusions

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.

Acknowledgments

- Liang Chen
- Qoobee
- Wushi
- All our other colleagues of KEEN Lab

Questions?

Twitter: @keen_lab

KEEN security lab

KEEN security lab

KEEN
security
lab