

Auditing and Exploiting Apple IPC

ianbeer

About me:

- Security Researcher with Project Zero
- Won pwn4fun last year with a JavaScriptCore bug and some kernel bugs
- That macbook air now runs ubuntu :)
- Over the last year reported ~60 OS X sandbox escapes/priv-escs (10 still unpatched)
- Some accidentally also present on iOS

This talk:

- Overview of (almost) all IPC mechanisms on iOS/OS X
- Quick look at Mach Message fundamentals
- Deep-dive into XPC services
- Exploiting XPC bugs
- fontd IPC and exploiting fontd bugs
- Mitigations and the future

IPC Zoo

socketpair semaphores
signals domain sockets
fifo shm

AppleEvents
Pasteboard

CFMessage
Port

Distributed
Notifications

NSXPC

A

B

CFPort

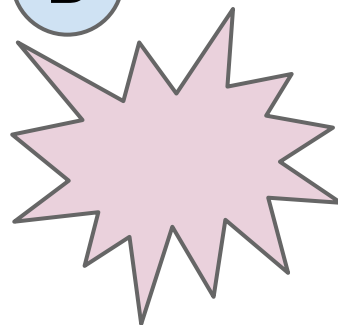
MIG

XPC

D
O

Mach Messages

XNU



Why care about IPC?

Sandboxing

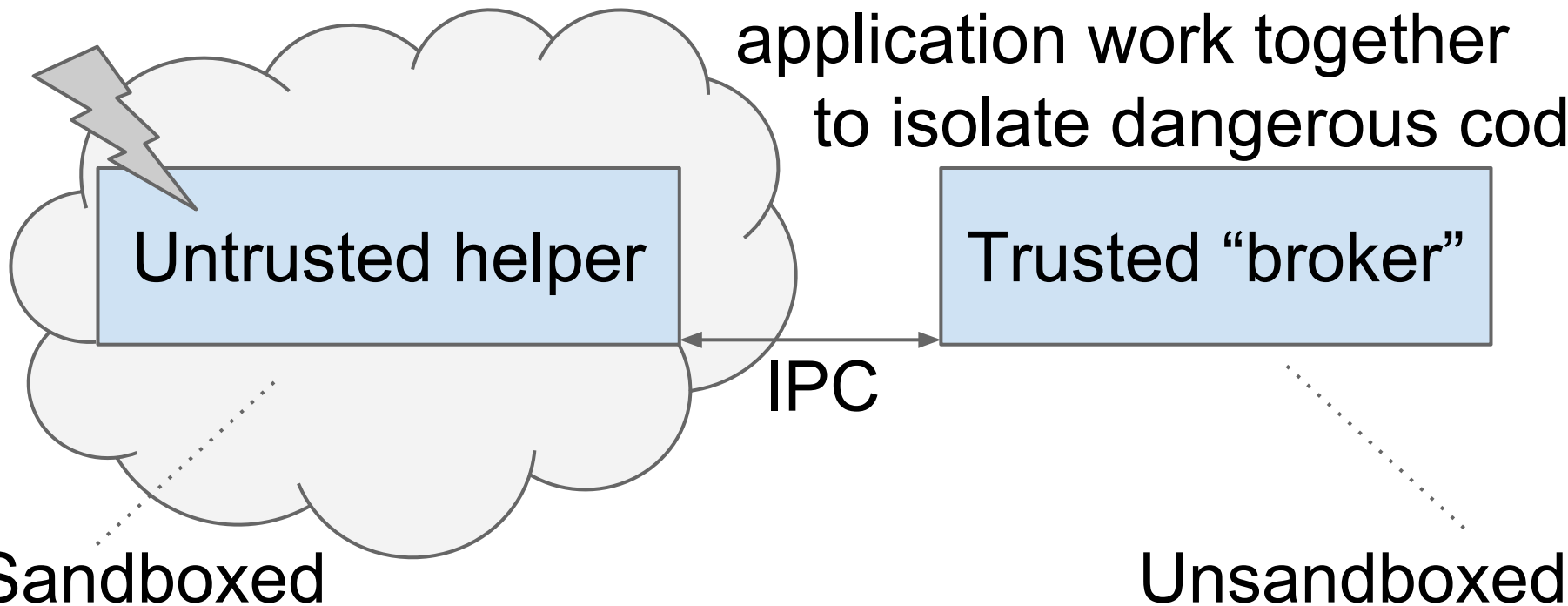
You *probably* get initial code execution in some kind of sandbox in userspace...

- renderer/plugin process
- quicklook-satellite
- ntpd
- appstore app

Plenty of stuff is still
unsandboxed on OS X
though
(...Adobe Reader...)

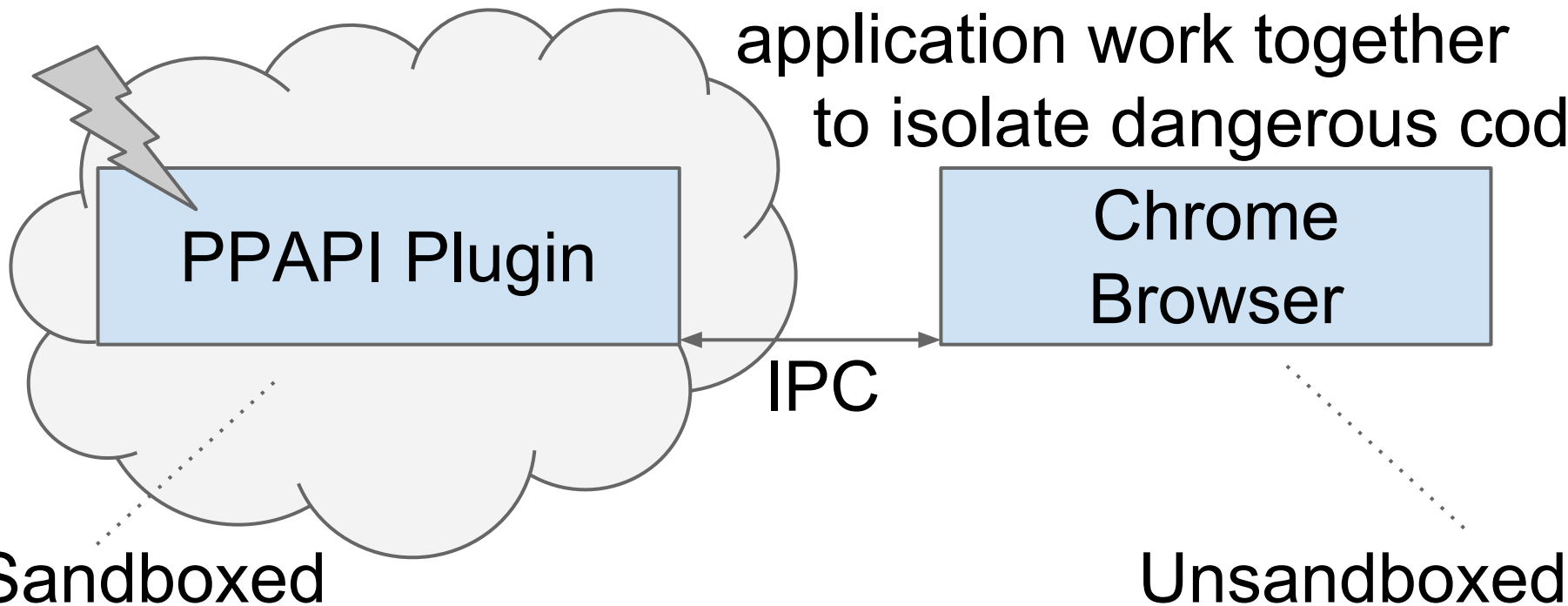
Sandbox escape models

Privilege separation: Two parts of the same application work together to isolate dangerous code



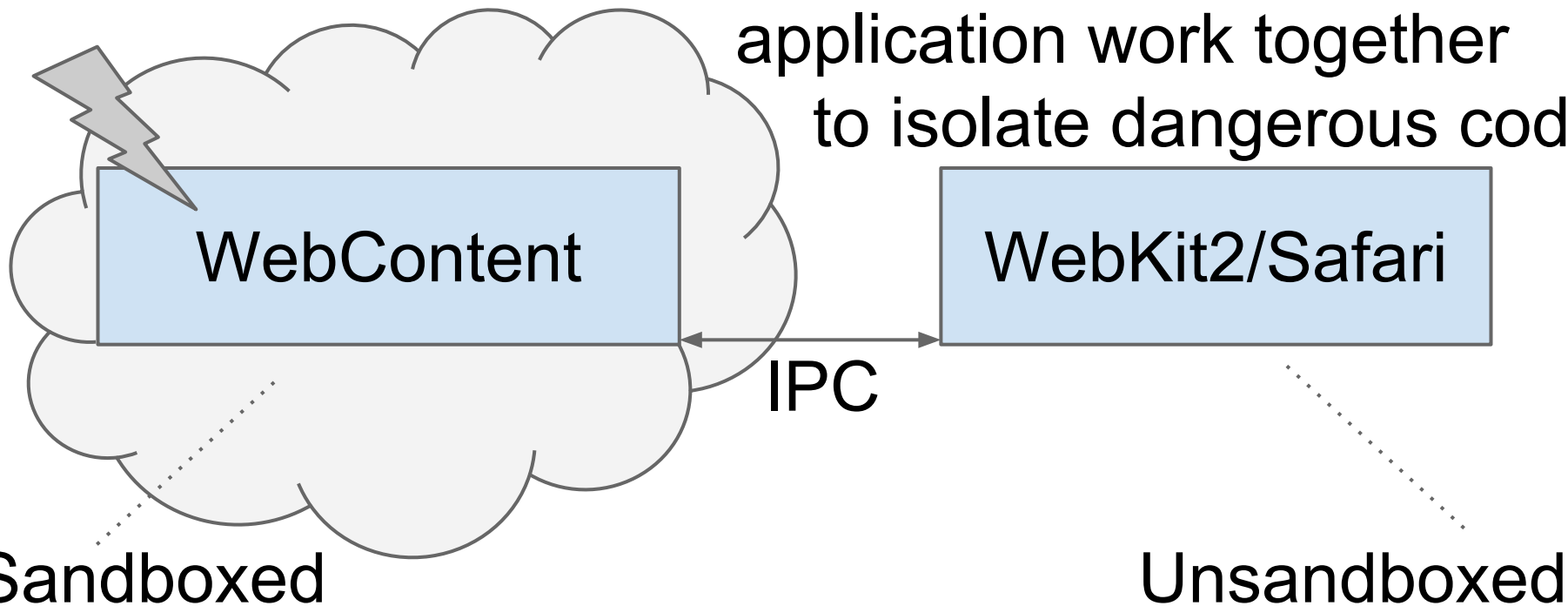
Sandbox escape models

Privilege separation: Two parts of the same application work together to isolate dangerous code



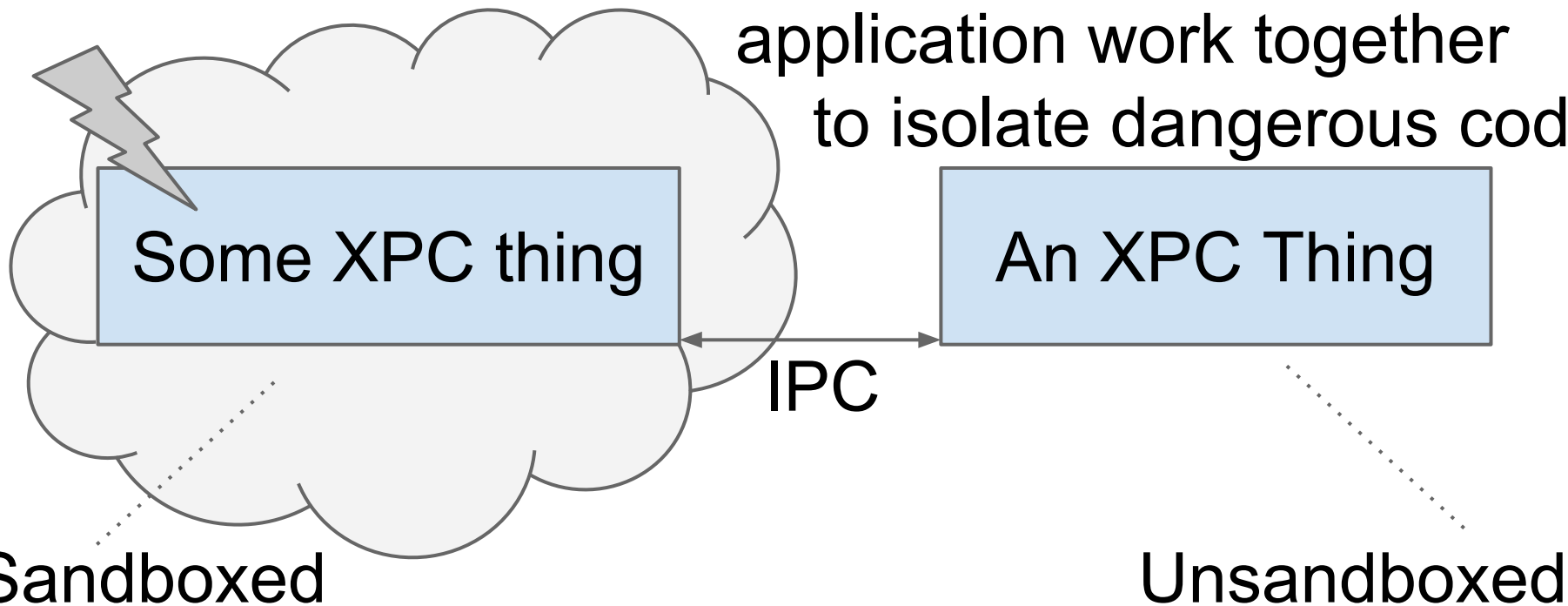
Sandbox escape models

Privilege separation: Two parts of the same application work together to isolate dangerous code



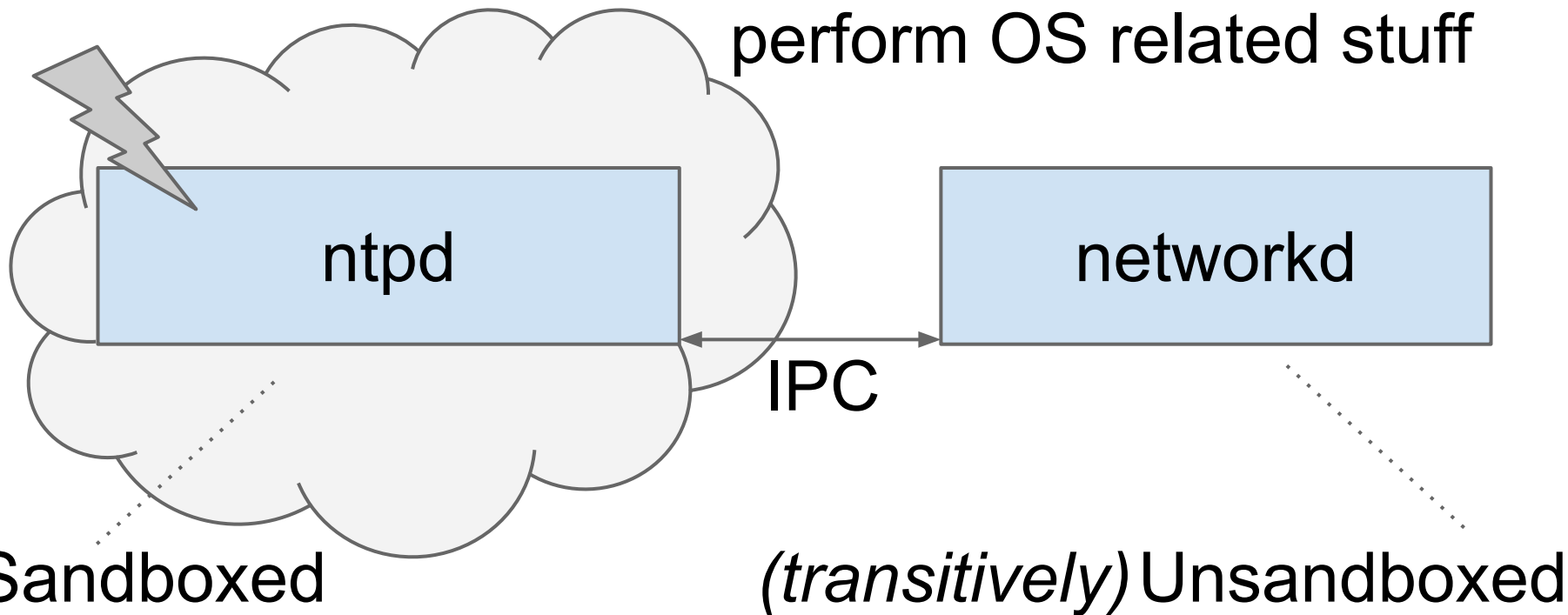
Sandbox escape models

Privilege separation: Two parts of the same application work together to isolate dangerous code



Sandbox escape models

System Services: OS provided IPC services which perform OS related stuff



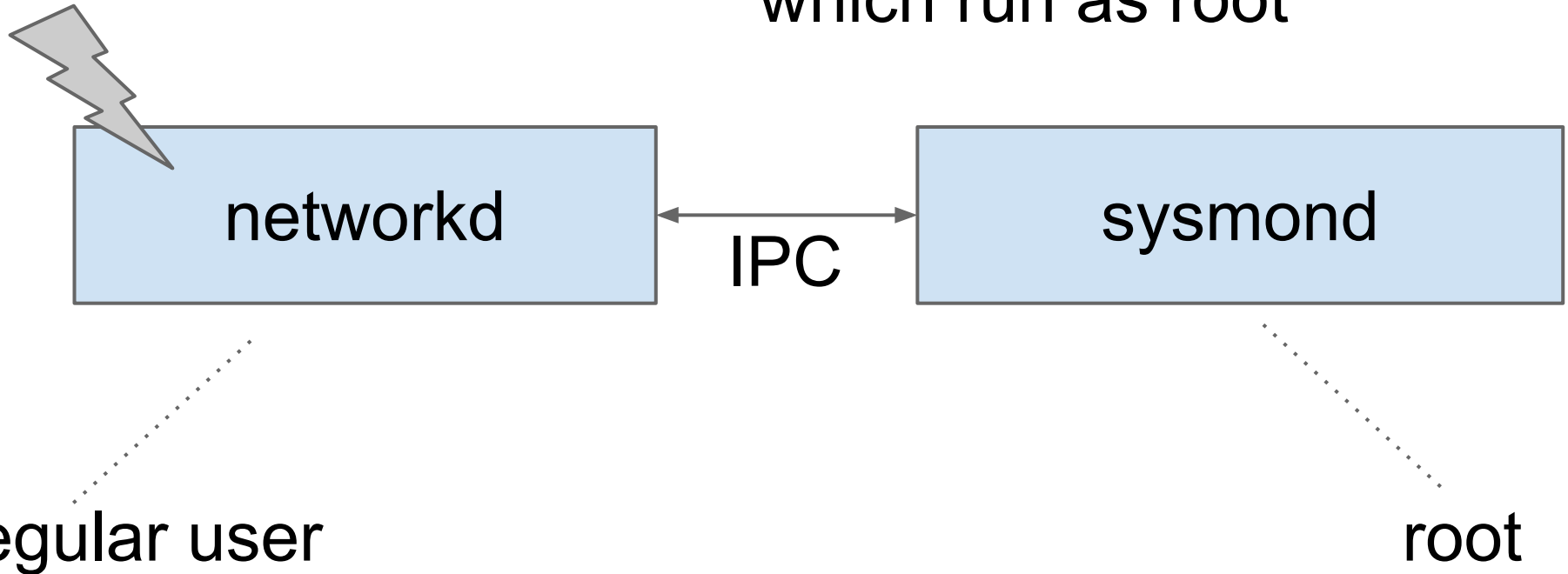
Privilege Escalation

OS X: root == kernel code execution

iOS: not that easy, but still, more attack surface

Privilege escalation model:

Root System Services: OS provided IPC services which run as root



it takes two to IPC

low-level mach messages and bootstrapping

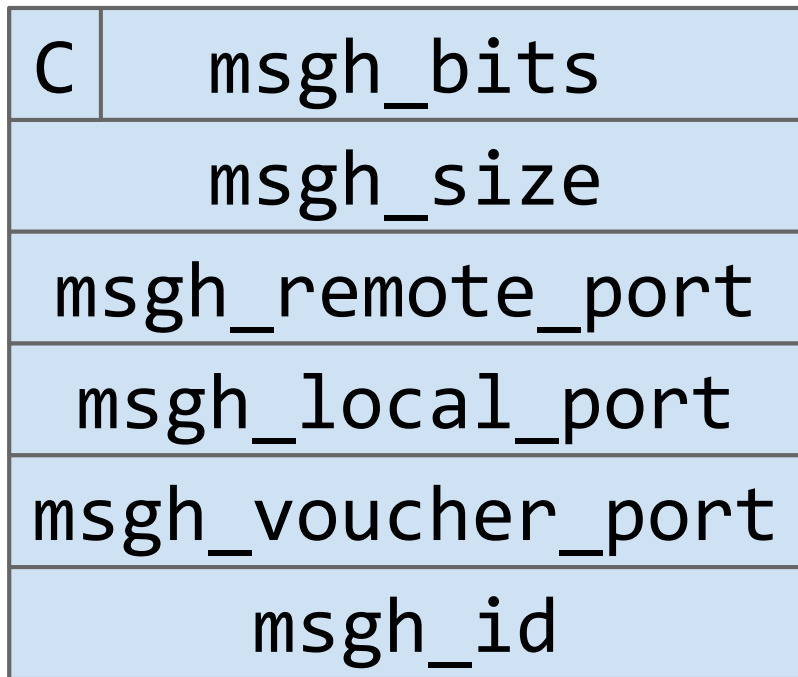
Building Mach Messages

Structure of a Mach Message:

complex flag indicates whether this message contains descriptors

`mach_msg_header_t:`

sending: ignored
receiving: message size excluding audit trailer



sending: optional reply port
receiving: local port message received on

sending: destination port to send to
receiving: optional reply port

ignored by Mach code; used by MiG as message identifier

new in Yosemite

Structure of a Mach Message:

`mach_msg_header_t`

only present if complex
flag set

`msg_header_descriptor_count`

`mach_msg_descriptor_t`

repeated `msg_header_descriptor_count`
times

...

inline data

`msg_header_trailer_type`

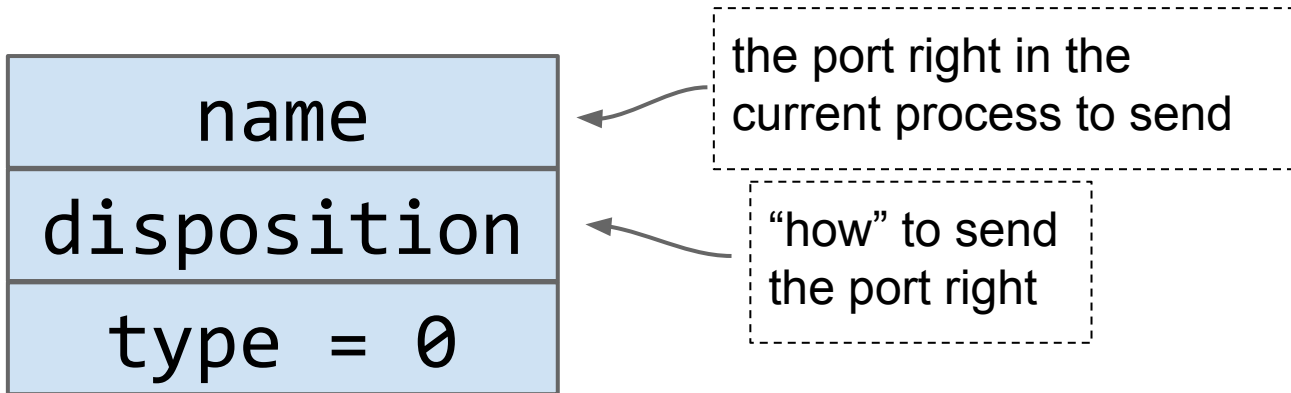
`msg_header_trailer_size`

trailers are requested by receiver
and appended by kernel; only
authenticity check is that they're
not included in `msg_header_size`.
audit trailer contains sender pid

...

Port Descriptors

`mach_msg_port_descriptor_t:`



OOL Descriptors

`mach_msg_ool_descriptor64_t:`

address
size
deallocate
copy
type = 1

send: address of vm region to send
receive: address where received
region has been mapped

should the region be
deallocated with `vm_deallocate`
when the message is sent?

launchd

launchd

- pid 1
- launchd manages system services
- All processes can talk to launchd
- provides the mechanisms to look up system services and connect to them
- system service == a send right to a mach port
 - launchd only cares about the initial connection, not the protocol

connecting to launchd services

```
mach_port_t connect_to_service(const char* service_name) {  
    mach_port_t bs_port, service_port;  
    kern_return_t err;  
  
    task_get_bootstrap_port(mach_task_self(), &bs_port);  
    err = bootstrap_look_up(bs_port, service_name, &service_port);  
    if (err == KERN_SUCCESS) {  
        return service_port;  
    } else {  
        return MACH_PORT_NULL;  
    }  
}
```

LaunchDaemons & LaunchAgents

- `/System/Library/Launch*` config files allow static registration of service names

```
<dict>
    <key>Label</key>
    <string>com.apple.nfsd</string>
    <key>ProgramArguments</key>
    <array>
        <string>/sbin/nfsd</string>
    </array>
</dict>
</plist>
```

bootstrap_checkin()

- Ask launchd for the mach port for the service name reserved in the Launch* plist:

```
bootstrap_check_in(bootstrap_port,  
                    "service_name",  
                    &servicePort);
```

follow xrefs to find
message handling code :)



bootstrap_register()

Deprecated (but still used) dynamic launched service registration:

```
bootstrap_register(bootstrap_port,  
                  "my_service",  
                  service_port);
```

follow xrefs to find
message handling code :)



launchctl

- tool to manage launchd
- since launchd has been rewritten, so has launchctl, so most documentation out-of-date!
- but start with: `sudo launchctl print system`

building a list of root services

Use launchctl; here's an incomplete list:

com.apple.ocspd	com.apple.wifi.anqp	com.apple.securitydservice
com.apple.launchd.peruser.0	com.apple.security.syspolicy	com.apple.wdhelper
com.apple.cfprefsd.daemon	com.apple.FontWorker	com.apple.DiskArbitration.diskarbitrationd
com.apple.taskgated	com.apple.FontWorker.ATS	com.apple.systemstatsd
com.apple.suhelperd	com.apple.installd	com.apple.networkd_privileged
com.apple.revisiond	com.apple.FileCoordination	com.apple.logind
com.apple.diskmanagementd	com.apple.ProgressReporting	com.apple.apsd
com.apple.alf	com.apple.cvmsServ	com.apple.network.IPConfiguration
com.apple.sysmond	com.apple.KernelExtensionServer	com.apple.SystemConfiguration.configd
com.apple.metadata.mds.index	com.apple.tccd.system	
com.apple.metadata.mds.xpc	com.apple.coreservices.launchservicesd	
com.apple.metadata.mds	com.apple.system.opendirectoryd.libinfo	
com.apple.metadata.mds.xpcs	com.apple.system.opendirectoryd.membership	
com.apple.cmio.VDCAssistant	com.apple.system.opendirectoryd.api	
com.apple.usbd	com.apple.system.DirectoryService.libinfo_v1	
com.apple.airportd	com.apple.system.DirectoryService.membership_v1	
com.apple.wifi.anqp	com.apple.private.opendirectoryd.rpc	

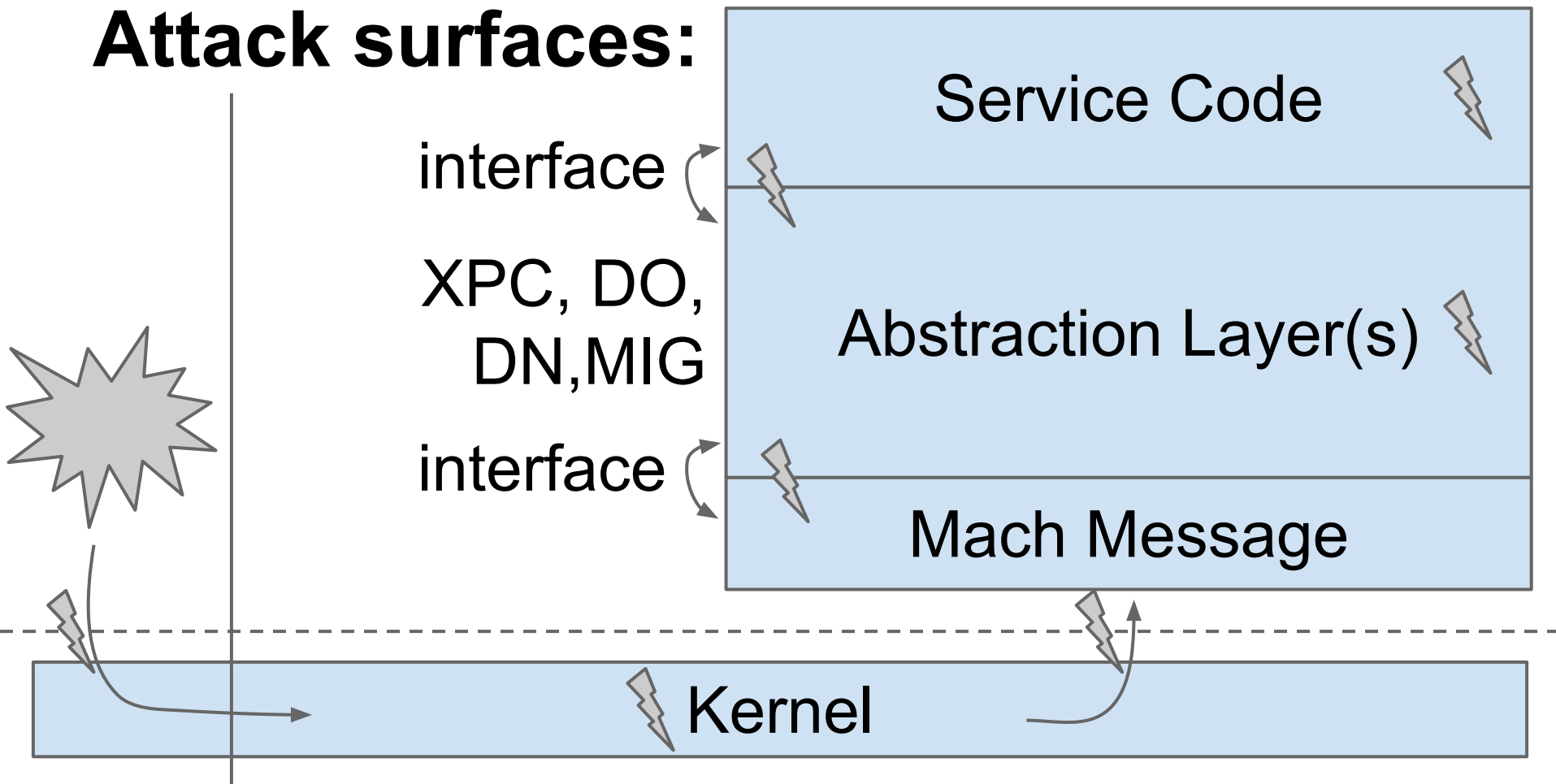
building a list of root services...

com.apple.SystemConfiguration.NetworkInformation	com.apple.AOSNotification.aps-production
com.apple.SystemConfiguration.PPPController-priv	com.apple.AOSNotification
com.apple.network.EAPOLController	com.apple.AOSNotification.aps-development
com.apple.SystemConfiguration.SCNetworkReachability	com.apple.AOSNotification.aps-demo
com.apple.SystemConfiguration.DNSConfiguration	com.apple.CoreServices.coreservicesd
com.apple.SystemConfiguration.PPPController	com.apple.SecurityServer
com.apple.networking.captivenetworksupport	
com.apple.SleepServices	
com.apple.warmd.server	
com.apple.sandboxd	
com.apple.coresymbolicationd	
com.apple.FSEvents	
com.apple.distributed_notifications@1v3	
com.apple.distributed_notifications@0v3	
com.apple.familycontrols	
com.apple.familycontrols.authorizer	
com.apple.system.notification_center	
com.apple.system.logger	
com.apple.PowerManagement.control	
com.apple.iohideventsystem	

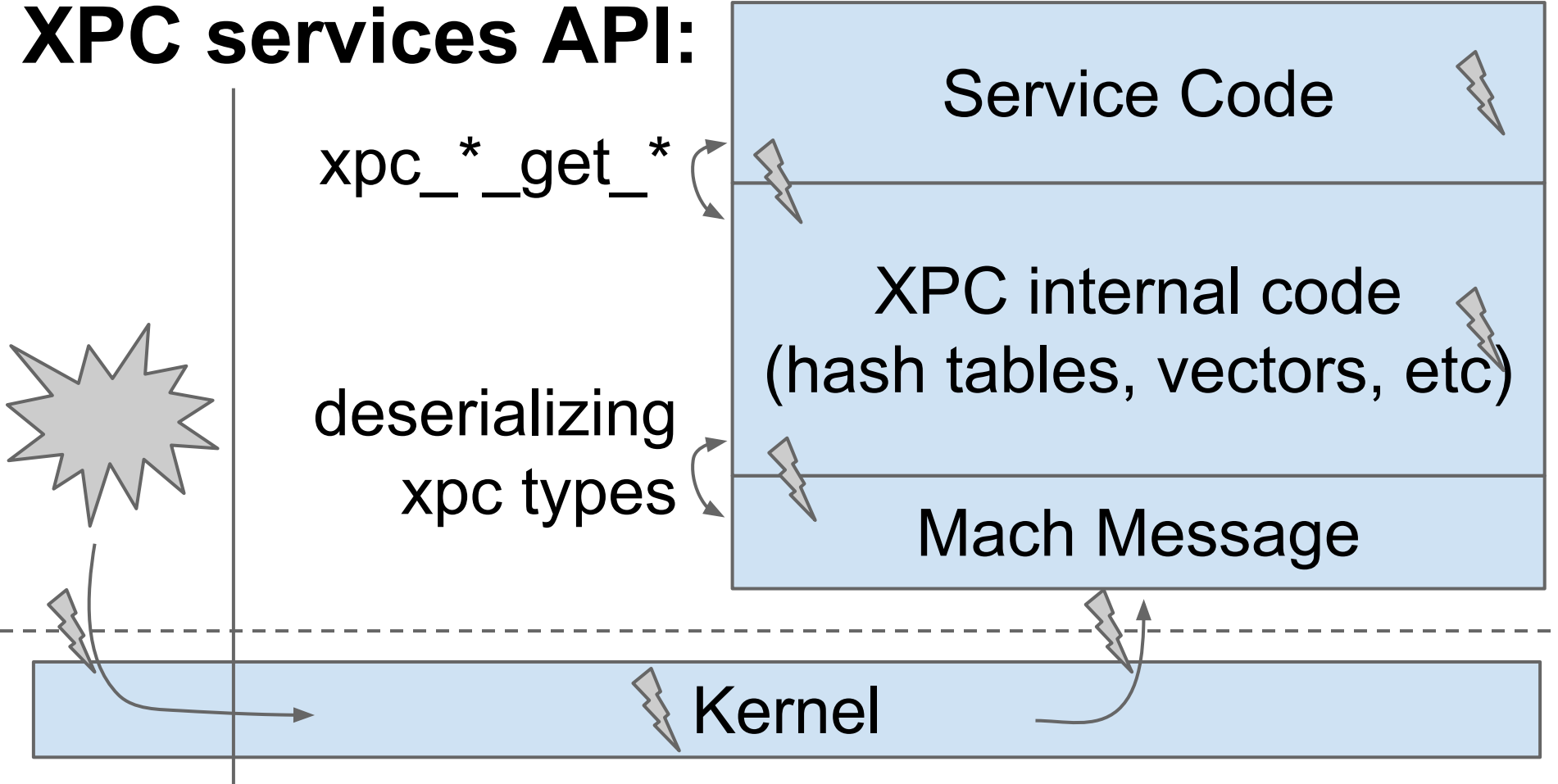
Building useful services

IPC services

Attack surfaces:



XPC services API:



XPC Internals

XPC Services Overview

- **not** built on MiG
- schema-less message passing abstraction
- messages are strongly-typed dictionaries
- data-types:
 - xpc_dictionary_t
 - xpc_array_t
 - xpc_string_t
 - xpc_(u)int64_t
 - xpc_uuid_t
 - xpc_data_t
 - xpc_date_t
 - xpc_bool_t
 - ...

Example XPC Message:

```
msg = { "type"           = 6,  
        "connection_id" = 1,  
        "state"          = { "power_slot": 0 },  
        "parameters"     = { "duration" = 0,  
                              "start"    = 0,  
                              "connection entry list" = [  
                                { "hostname": "example.com" }  
                              ],  
        }  
}
```

The wire format isn't quite as nice as this...

XPC Wire Format: Simple Dictionary

Write test program to send XPC messages

```
(lldb) break set --name _xpc_serializer_get_dispatch_mach_msg
(lldb) continue
(lldb) finish
(lldb) x/22xw $rax+0x40 ;this is the mach message
```

```
dict {"key": "value"}
```

0x00000013	0x00000040	0x00000000	0x00000000	; mach_msg_header_t
0x00000000	0x10000000	0x58504321	0x00000004	; fixed_header XPC! 0x4
0x0000f000	0x00000018	0x00000001	0x0079656b	; dict_type byte_len n_entries "key\x00"
0x00009000	0x00000006	0x756c6176	0x00000065	; string_type byte_len "value\x00"
0x00000000	0x00000000	0x00000000	0x00000000	
0x00000000	0x00000000			

XPC Wire Format: Bigger Dictionary

```
dict {"key": "value", "auint64": 0x41414141...}
```

```
0x00000013 0x00000054 0x00000000 0x00000000
```

```
0x00000000 0x10000000 0x58504321 0x00000004
```

```
0x0000f000 0x0000002c 0x00000002 0x6e697561 ; n_entries "auint64\x00"
```

```
0x00343674 0x00004000 0x41414141 0x41414141 ; uint64_type uint64_value
```

```
0x0079656b 0x00009000 0x00000006 0x756c6176
```

```
0x00000065 0x00000000 0x00000000 0x00000000
```

```
0x00000000 0x00000000
```

XPC Wire Format: Dictionary with Data

```
dict {"key": "value",  
      "auint64": 0x41414141...  
      "data": \x41\x42\x43\x44 } //short data is inline  
0x00000013 0x00000068 0x00000000 0x00000000  
0x00000000 0x10000000 0x58504321 0x00000004  
0x0000f000 0x00000040 0x00000003 0x6e697561 ; n_entries  
0x00343674 0x00004000 0x41414141 0x41414141  
0x0079656b 0x00009000 0x00000006 0x756c6176  
0x00000065 0x61746164 0x00000000 0x00008000 ; "data\x00" data_type  
0x00000004 0x44434241 ; data_byte_len data_payload
```

XPC Wire Format: Dictionary with port

```
dict {"key": xpc_connection(NULL)}
```

0x80000013	0x00000044	0x00000000	0x00000000	;	MACH_MSGH_BITS_COMPLEX
0x00000000	0x10000000	0x00000001	0x00001003	;	msg_id descriptor_count
0x00000000	0x00110000	0x58504321	0x00000004	;	port_desc_type port_move_send
0x0000f000	0x0000000c	0x00000001	0x00434241		
0x00013000				;	xpc_connection_type

XPC Deserialization Code

```
_xpc_TYPE_deserialize(xpc_serializer_t*);
```

```
xpc_serializer_t + 0x48
```

```
= pointer to data
```

```
xpc_serializer_t + 0x50
```

```
= remaining data length
```

Deserializers seem reasonably robust, impose sensible limits etc

XPC Object Creation:

```
_xpc_object_create(OBJC_CLASS* type,  
                  uint32_t extra);
```

extra bytes to allocate
for object fields



XPC Object Internals:

`xpc_{(u)int64_t, double, date}`

+0x28: 8 byte value

Simple objects,
1 8-byte data field

XPC Object Internals:

`xpc_string_t`

`+0x28: string length`

`+0x30: pointer to strdup'ed chars`

XPC Object Internals:

xpc_uuid_t

+0x28: first 8 UUID bytes

+0x30: second 8 UUID bytes

XPC Object Internals:

xpc_data_t

+0x28: dispatch_once count
+0x30: *dispatch_object_t
+0x38: offset
+0x40: dispatch data size
+0x48: mapped_already flag

XPC Object Internals:

`xpc_array_t`

`+0x2c: array length`

`+0x30: calloc'ed xpc_object_t buffer`

XPC Object Internals:

xpc_dictionary_t

+0x60: 11 hash_buckets[6]

XPC Object Internals:

xpc dictionary linked-list entries:

```
struct ll {  
    struct ll* forward;  
    struct ll* backward;  
    xpc_object_t* object;  
    uint64_t flags;  
    char key[0]; // allocated inline  
}
```

Knowing the
internals of this
structure is
super-helpful for
exploitation

XPC Services API: safe version

```
xpc_{dictionary, array}_get_{TYPE}()
```

Checks that the entry is of the expected type;
returns a NULL value if not

XPC Services API: unsafe version

`xpc_{dictionary, array}_get_value()`

returns an `xpc_object_t`,
which is really:

```
typedef void * xpc_object_t;
```

Remember, xpc is schema-less,
an attacker can send any xpc type

Type Confusion in XPC:

The use of `void*` means the compiler won't warn about bad uses of `xpc_object_t`

But is that interesting?

Avoiding Type Confusion in XPC:

Either:

- ~~XPC API entrypoints must check types~~

Before Yosemite, no entrypoints checked types

- ~~API consumers must check types~~

some did, some didn't ;)

Implications of XPC type confusion

If API consumer code doesn't check types, we can force a controlled, incorrect, `xpc_*` type to be passed to an `xpc_` API.

Implications depend on:

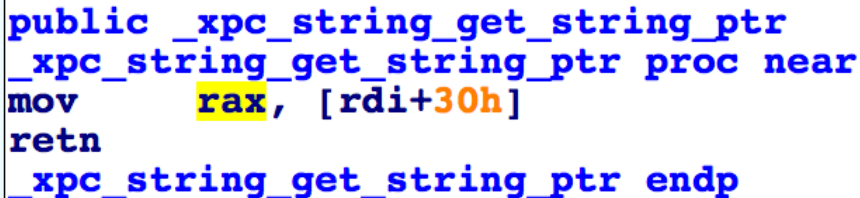
- What fields overlap with what
- How are those fields are used

XPC type confusion example

str can be
of any type

attacker-controlled
dictionary

```
xpc_object_t str = xpc_dictionary_get_value(msg, "foo");  
printf("%s\n", xpc_string_get_string_ptr(str));
```



```
public _xpc_string_get_string_ptr  
_xpc_string_get_string_ptr proc near  
mov     rax, [rdi+30h]  
retn  
_xpc_string_get_string_ptr endp
```

simply treats the
value at +0x30 as
a c-string pointer!

Cool, can we do more?

XPC object overlap

offset	uint64	string	array	uuid	data
+0x28	value	length	length	value[0:8]	dispatch_count
+0x30	---	char*	xpc_object_t*	value[8:16]	dispatch_object_t*

This has been strdup-
ed, so no NULL bytes
means tougher to use

Can confuse a pointer with 8
completely controlled bytes :)

What is a dispatch_object_t?

- Objective-C object
- Objective-C method called on it
- nemo already covered this!

Example vulnerable code:

attacker passes an XPC_UUID



```
xpc_object_t obj = xpc_dictionary_get_value(msg, "data");  
const void* data = xpc_data_get_bytes_ptr(obj);
```

Will treat second 8 bytes as an
Objective-C object pointer :)



There is actually one more hurdle:
the byte at +48 has to be 0, but the
XPC UUID is smaller than that...

Dictionary deserialization

The heap object following the UUID will be the UUID's dictionary LL entry:

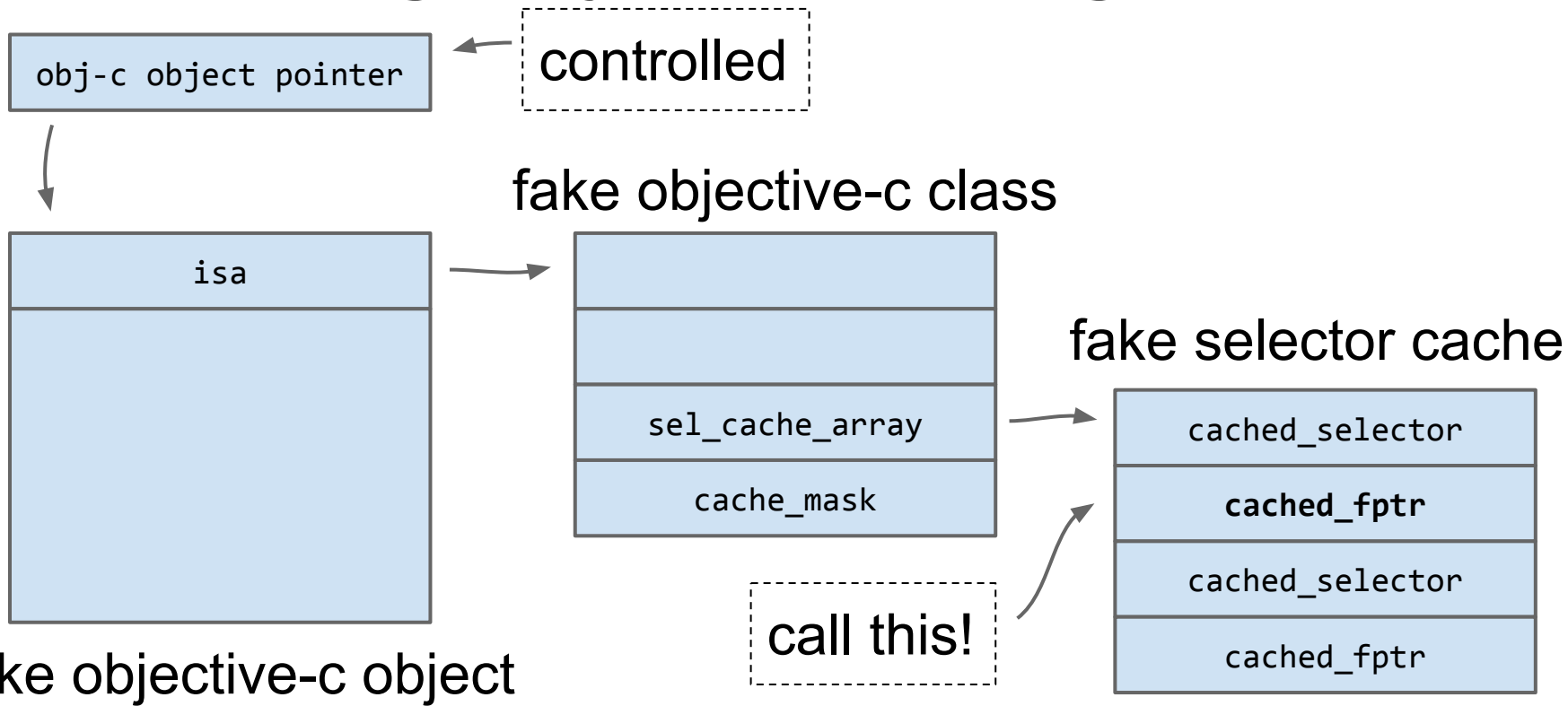
```
struct ll {  
    struct ll* forward;  
    struct ll* backward;  
    xpc_object_t* object;  
    uint64_t flags;  
    char key[0];  
}
```

The least-significant byte of that entry's backward pointer will be the `already_mapped` flag

easy :) ensure that the most recently deserialized LL entry in this hash bucket was > 512 bytes which will make the allocation 256-byte aligned

XPC type confusion exploitation techniques

Exploiting Objective-C bugs



What/Where

- Need known data at a known location
- Lame heap spray!
- Depressingly effective :(
- nemo has told you about fancier techniques :)

Heap spraying with XPC

```
// fill a page (hs) with the data you want
size_t heap_spray_pages = 0x40000; // 1GB
size_t heap_spray_bytes = heap_spray_pages * 0x1000;
char* heap_spray_copies = malloc(heap_spray_bytes);
for (int i = 0; i < heap_spray_pages; i++){
    memcpy(heap_spray_copies+(i*0x1000), hs, 0x1000);
}

xpc_dictionary_set_data(msg, "heap_spray", heap_spray_copies,
heap_spray_bytes);

// find your data at 0x120200000 in the target :)
```

**Are there really services
with that very specific
pattern?**

Yes, lots!

networkd XPC type confusion bug

<https://code.google.com/p/google-security-research/issues/detail?id=130>

breaks you out of ntpd and safari sandboxes

sysmond XPC type confusion bug

<https://code.google.com/p/google-security-research/issues/detail?id=121>

user -> root priv-esc

Finding all the bugs

- This bug class can be pretty easily described and found using Abstract Interpretation
- Wrote a hacky AI framework for x64 (~600 lines of python)
- Ran it over all executables
- Found many more bugs :) Apple since patched xpc_data entrypoints

Apple patches

- Minimal

fontd

to MiG or not to MiG...

Fontd

The fontd process actually hosts two services:

`com.apple.FontObjectsServer`

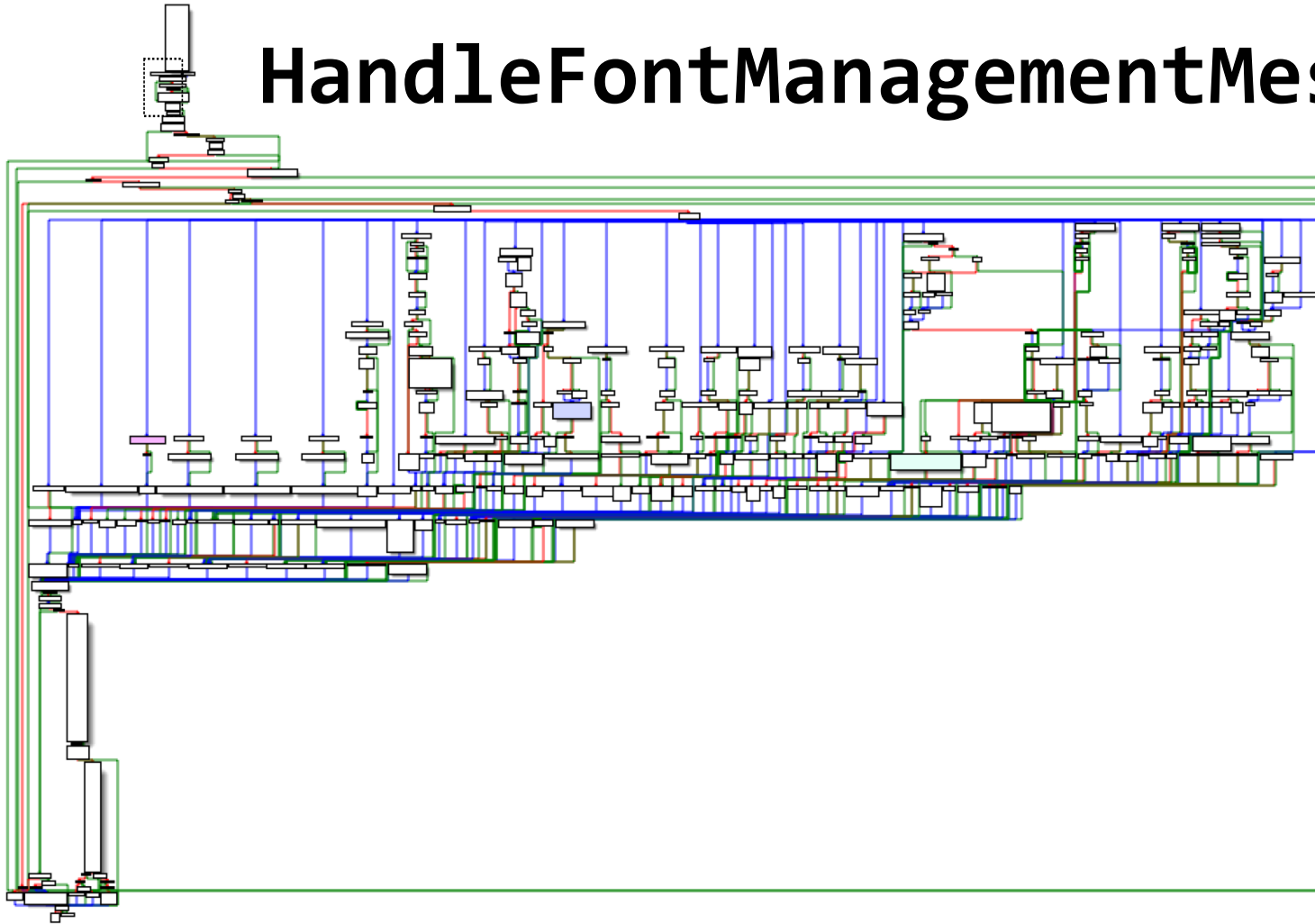
`com.apple.FontServer`

reachable from a lot of interesting sandboxes

com.apple.FontObjectsServer

- Doesn't use MiG
- Hand-rolled mach message parsing atop CFMachPort
- Crazy legacy code paths (supports sender and receiver having different endian-ness?!)
- Implemented in `libATSServer.dylib`

HandleFontManagementMessage:



unspaghettifying: IDAPython

```
import idaapi

jmp_table_addr = 0x85964      # where's the jump table?
jmp_table_cases = 47         # how big is it?
jmp_table_labels = 0x96120    # where are the labels?
label_len = 0x30             # how big are they?

for i in range(jmp_table_cases):
    case_addr = ((jmp_table_addr + Dword(jmp_table_addr + (i*4))) & 0xffffffff)
    label_str = GetString(jmp_table_labels + (i*label_len))
    comment = GetCommentEx(case_addr, 0)
    if comment is None:
        comment = ""
    else:
        comment += '\n'
    comment += label_str + " case:" + str(i)
    MakeComm(case_addr, comment)
```

FontObjectsServer method names:

kFORendezvousMessage

kFODBSynchMessage

kFOSynthesizeTablesMessage

kFOActivateFontsMessage

kFODeactivateFontsMessage

kFOActivateFontsFromMemoryMessage

kFODeactivateFontsInContainerMessage

kFOGetContainerMappingMessage

kFOGetAnnexDataMessage

kFOGetFileTokenFlatFSRefMessage

kFOResolveFileTokenMessage

kFOComputeFontSpecsMessage

kFOMarkFontAsBadMessage

kFOEnableFontProtectionMessage

kFOScanFontDirectoriesMessage

kFOUserDirInfoMessage

kFOShutdownServerMessage

kFOPingServerMessage

kFOAddToFontNamesCacheMessage

kFOFindUnicodeEncodingMessage

kFOGetFCacheDataMessage

kFOMapSharedMemoryMessage

kFOFindFontIDFromNameMessage

kFOGetKnownDirsInfoMessage

kFORegisterQueryPortMessage

kFOUnregisterQueryPortMessage

kFOSynthesizeFontFamilyResourcesMessage

kFOGetPSFontEncodingMessage

kFOEnableFontMessage

kFODBDumpForFileTokenMessage

FontObjectsServer method names:

kFOActivateFontsWithInfoMessage

kOFAStrStreamMessage

kOFAStrStrikeMessage

kOFAGeneralMessage

kOFACacheSynchMessage

kOFACacheProcessUsageMessage

kOFACacheFindMessage

kFOEnableFinderNotificationsMessage

kFOEnableUINotificationsMessage

kFOGetPersistentDataMessage

kFOSavePersistentDataMessage

kFOGetFontProtectionMessage

kFOGetFontTraitsMessage

kFOSetFontFlagsMessage

kXTURLActionMessage

kXTGenDBCompleteMessage

kXTURLActionClientMessage

More IDAPython: make a switch tab

based on <https://github.com/aaronportnoy/toolbag/blob/master/user/bin/switchViewer.py>

```
import idutils
```

```
import idaapi
```

```
import idc
```

```
class SwitchTab(idaapi.simplecustviewer_t):
```

```
    def __init__(self, table_addr, targets):
```

```
        self.table_addr = table_addr
```

```
        self.targets = targets
```

```
        self.Create()
```

```
        self.Show()
```

```
    def Create(self):
```

```
        idaapi.simplecustviewer_t.Create(self, "0x%x switch destinations" % self.table_addr)
```

```
        comment = idaapi.COLSTR("; Double-click to follow", idaapi.SCOLOR_BINPREF)
```

```
        self.AddLine(comment);
```

```
        for t in self.targets:
```

```
            line = idaapi.COLSTR("0x%x:" % t, idaapi.SCOLOR_REG)
```

```
            self.AddLine(line)
```

```
        return True
```

```
def OnDbClick(self, shift):
    line = self.GetCurrentLine()
    if "0x" not in line:
        return False
    target = int(line[2:line.find(':')], 16)
    idc.Jump(target)
    return True
```

```
jmp_addr = ScreenEA()
switch_info = idaapi.get_switch_info_ex(jmp_addr)
if switch_info == None:
    print "that isn't a jump-table jump"
else:
    # number of cases
    num_cases = switch_info.get_jtable_size()
    print '0x%08x: switch (%d cases)' % (jmp_addr, num_cases)
    for t in idautils.CodeRefsFrom(jmp_addr, 1):
        print "0x%x" % t
    SwitchTab(jmp_addr, idautils.CodeRefsFrom(jmp_addr, 1))
```

a first FontObjectsServer bug:

```
loc_845C7:                ; kXTURLActionMessage case:44
lea      rdi, [r14+18h]
call     __ZL26DoHandleXTURLActionMessageP14XTURLActionMsg ; DoHandleXTURLActionMessage(XTURLActionMsg *)
mov      ebx, eax
mov      rdi, [r14+18h]
test     rdi, rdi
jz       short loc_845E0
```

r14 points to the received mach message,
so rdi will point to controlled data...

a first FontObjectsServer bug:

```
push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r13
push    r12
push    rbx
sub     rsp, 4E8h
mov     r15, cs:___stack_chk_guard_ptr
mov     rax, [r15]
mov     [rbp+var_30], rax
mov     rax, [rdi]
test    rax, rax
jz      short loc_861C4
```

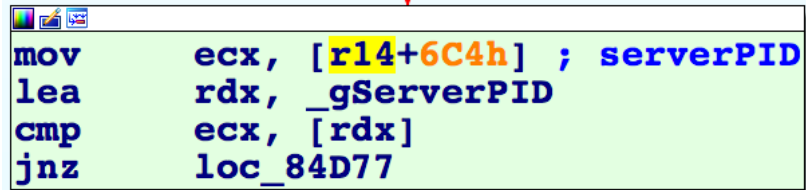
rdi points to controlled data

so we control
rax here...

```
mov     rbx, rdi
mov     rdi, rax
call    _CFRetain
mov     rdi, rbx
```

this will msgSend
CFRetain to rax?!

message format weirdness:



```
mov     ecx, [r14+6C4h] ; serverPID
lea     rdx, _gServerPID
cmp     ecx, [rdx]
jnz     loc_84D77
```

Dumb generational
fuzzer unlikely to make it
past this...

But manual analysis
gets past this trivially...





















com.apple.FontServer

- The other service hosted by fontd
- MiG-based
- Implemented in `libFontRegistryServer.dylib`
- Custom CF object serialization format :)
- Also allow by a bunch of interesting sandboxes:
 - Chrome renderer
 - Safari

Finding MiG entrypoints without .defs

If there are some symbols, MiG functions nearly always use a common prefix:

Function name

	__XAddFontProvider
	__XCopyAvailableFontFamilyNames
	__XCopyAvailableFontNames
	__XCopyAvailableFonts
	__XCopyAvailableFontsSandboxed
	__XCopyDuplicateFonts
	__XCopyFamilyNamesForLanguage
	__XCopyFontDirectories
	__XCopyFontForCharacter
	__XCopyFontForCharacterSandboxed
	__XCopyFontWithName
	__XCopyFontWithNameSandboxed
	__XCopyFontsMatchingRequest
	__XCopyFontsMatchingRequestSandboxed
	__XCopyLocalizedNameForFonts
	__XCopyLocalizedPropertiesForFonts
	__XCopyPropertiesForAllFonts
	__XCopyPropertiesForFont
	__XCopyPropertiesForFontMatchingRequest
	__XCopyPropertiesForFontMatchingRequestSandboxed

with no symbols at all:

Look for this structure in the `__DATA:__const:`

```
/* Description of this subsystem, for use in direct RPC */
const struct _notify_ipc_subsystem {
    mig_server_routine_t    server; /* Server routine */
    mach_msg_id_t          start; /* Min routine number */
    mach_msg_id_t          end;    /* Max routine number + 1 */
    unsigned int            maxsize; /* Max msg size */
    vm_address_t            reserved; /* Reserved */
    struct routine_descriptor /*Array of routine descriptors */
        routine[38];
} _notify_ipc_subsystem = {
    notify_ipc_server_routine,
    78945668,
    78945706,
    (mach_msg_size_t)sizeof(union __ReplyUnion__notify_ipc_subsystem),
    (vm_address_t)0,
    {
        { (mig_impl_routine_t) 0,
          (mig_stub_routine_t) X_notify_server_post, 12, 0, (routine_arg_descriptor_t)0,
          (mach_msg_size_t)sizeof(__Reply__notify_server_post_t)}, // ...
    }
}
```

Reversing MiG function prototypes

- If `__MigTypeCheck` is defined (which is hopefully is!) then MiG will generate “type-checking” code
 - Null-termination check for strings
 - Number of OOL descriptors
- Will then unpack arguments + return value pointers and pass to service code

Serialization













- Probably the most fundamental property of any IPC system
- There are an almost uncountable number of object serialization implementations in OS X/iOS, and new ones are being added all the time

FontServer object serialization

- Most FontServer RPCs take serialized CF objects
- CF already has some object serialization (eg plist)
- but hey, why not write a custom one for fontd? :)

TCFResurrectContext

Implements the deserialization

-  TCFResurrectContext::Resurrect(TCFTType)
-  TCFResurrectContext::ResurrectCFArray(void)
-  TCFResurrectContext::ResurrectCFBoolean(void)
-  **TCFResurrectContext::ResurrectCFCharacterSet(void)**
-  TCFResurrectContext::ResurrectCFData(void)
-  TCFResurrectContext::ResurrectCFDictionary(void)
-  TCFResurrectContext::ResurrectCFError(void)
-  TCFResurrectContext::ResurrectCFNumber(void)
-  TCFResurrectContext::ResurrectCFSet(void)
-  TCFResurrectContext::ResurrectCFString(void)
-  TCFResurrectContext::ResurrectCFURL(void)
-  TCFResurrectContext::ResurrectCFUUID(void)

TCFResurrectContext format:

CFArray

type = 0x11
n_entries

...

CFString

type = 0x7
length
chars

...

CFData

type = 0x12
length
data

...

They're almost all very simple...

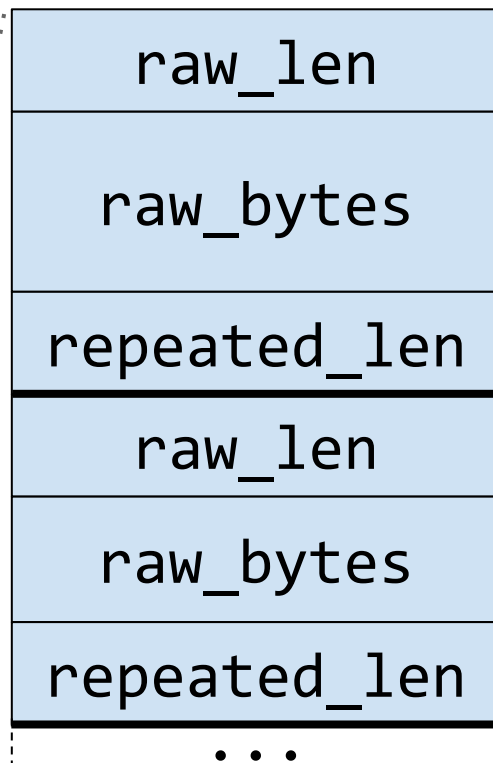
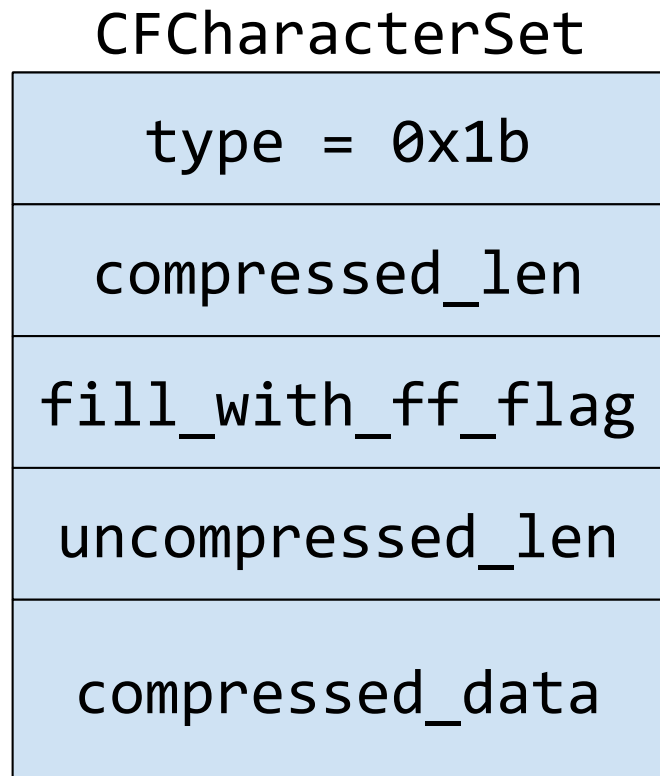
CFCharacterSet

“A CFCharacterSet object represents a set of Unicode compliant characters.”

<https://developer.apple.com/library/mac/documentation/CoreFoundation/Reference/CFCharacterSetRef/index.html>

Basically a bitmap, this should also be uninteresting...

CFCharacterSet serialization



2-byte length
of raw data in
2-byte units

fill with twice this
number of either
0xff or 0x00
bytes

No bounds
checking in
decompression :(

mov **r13**, r14 ; points 8 bytes in to the input buffer

```
loc_33D94:                ; void *
lea      rsi, [r13+2]
movzx    r12d, word ptr [r13+0]
lea      rdx, [r12+r12] ; size_t
mov      rdi, rbx        ; void *
call     _memcpy
lea      r14, [r13+r12*2+2] ; place to start in the input stream
lea      rax, [rbx+r12*2] ; place to start in the output buffer
cmp      r14, r15
jnb      short loc_33DD9
```

```
lea      r14, [r13+r12*2+4] ; input skipped ahead another two bytes
movzx    r13d, word ptr [r13+r12*2+2]
lea      rdx, [r13+r13+0] ; size_t
mov      rdi, rax        ; void *
mov      esi, [rbp+var_2C] ; int
call     _memset
add      r13, r12
lea      rax, [rbx+r13*2]
```

```
loc_33DD9:
cmp      r14, r15
mov      rbx, rax
mov      r13, r14
jb       short loc_33D94 ; continue if there's still input
```

More IPC Mechanisms

and how to find them

Distributed Objects

- very old Cocoa RPC technology
- allows “transparent” RPC by exposing local Objective-C objects via proxy objects in other processes
- calling a method on the proxy forwards the method call to the real object
- it’s actually still used!

vending an object via DO:

```
#import <objc/Object.h>
#import <Foundation/Foundation.h>

@interface VendMe : NSObject
- (oneway void) foo: (int) value;
@end

@implementation VendMe
- (oneway void) foo: (int) value;
{
    NSLog(@"%d", value);
}
@end
```

```
int main (int argc, const char * argv[]) {
    VendMe* toVend = [[VendMe alloc] init];

    NSConnection *conn;
    conn = [NSConnection defaultConnection];

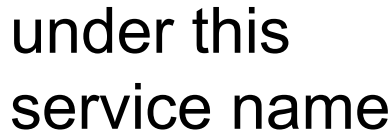
    [conn setRootObject:toVend];
    [conn registerName:@"service_name"];

    [[NSRunLoop currentRunLoop] run];
    return 0;
}
```

vend this object



under this
service name



connecting to a Distributed Object:

```
#import <Cocoa/Cocoa.h>
```

```
int main(int argc, char** argv){  
    id theProxy = [[NSConnection  
        rootProxyForConnectionWithRegisteredName:@"service_name"  
        host:nil] retain];  
    [theProxy foo:123];  
    return 0;  
}
```

create a proxy object by
connecting to the named
service

call the foo method
on the remote object
passing 123 as the
argument

DO Protocols

- restrict vendored object methods
- can use to enumerate exposed attack surface

define a protocol

```
@protocol MyProtocol  
- (oneway void) foo: (int) value;  
@end
```

```
@interface VendMe: NSObject <MyProtocol>  
...  
@end
```

implement it

use it remotely

```
[proxy setProtocolForProxy:@protocol(MyProtocol)];
```

Custom DO serialization

Scope for memory corruption :)

`NSCoding -initWithCoder:`

NSXPCConnection

- A “modern” equivalent to Distributed Objects:

```
NSXPCConnection *conn = [[NSXPCConnection alloc]  
initWithServiceName:@"service_name"];
```

connect to this service



```
conn.remoteObjectInterface =  
[NSXPCInterface interfaceWithProtocol:@protocol(MyProtocol)];
```

protocol same as DO



```
[conn resume];
```


```
[[conn remoteObjectProxy] foo:123];
```

call remote method



Vending NSXPCCConnection Objects


```
NSXPCListener *listener = [NSXPCListener serviceListener];  
id delegate = [MyDelegate new];  
listener.delegate = delegate;  
[listener resume];
```



register a delegate

that delegate's `shouldAcceptNewConnection` method:

```
- (BOOL)listener:(NSXPCListener *)listener  
  shouldAcceptNewConnection:(NSXPCCConnection *)conn {  
    conn.exportedInterface =  
    [NSXPCTInterface interfaceWithProtocol:@protocol(MyProtocol)];  
    connection.exportedObject = [VendMe new];  
    [connection resume];  
    return YES;  
}
```



The exported object

DistributedNotifications

- Broadcast named messages to all subscribers
- Can attach optional CFDictionary with the usual CF data types
- You don't know who actually sent the notification, don't trust them!
 - (especially if you're running as root...)
- Pretty widely used

Sending a Distributed Notification:

CFMutableDictionaryRef **dictionary** =

CFDictionaryCreateMutable(NULL,

0,

&kCFTTypeDictionaryKeyCallBacks,

&kCFTTypeDictionaryValueCallBacks);

CFDictionary will be copied
to all subscribers

CFDictionaryAddValue(dictionary, @"a_key", @"a_value");

CFNotificationCenterPostNotificationWithOptions(

CFNotificationCenterGetDistributedCenter(),

CFSTR("my.notification.name"),

NULL,

dictionary,

kCFNotificationDeliverImmediately | kCFNotificationPostToAllSessions);

Post this notification
name with that
dictionary

Receiving a Distributed Notification:

```
CFNotificationCenterAddObserver(CFNotificationCenterGetDistributedCenter(),  
    NULL,  
    MyNotificationCallback,  
    CFSTR("my.notification.name"),  
    NULL,  
    CFNotificationSuspensionBehaviorDeliverImmediately);
```

register this
callback function

for this
notification name

```
void MyNotificationCallback(CFNotificationCenterRef center,  
    void *observer,  
    CFStringRef name,  
    const void *object,  
    CFDictionaryRef userInfo);
```

attacker
controlled
CFDictionary
passed to
callback

Defense-in-depth

stronger sandboxing on OS X

Mach message “firewall”

- Want more granular sandboxing than launchd provides
- See `launchd_interception_server.cc` in chromium
- But, broken in Yosemite:
 - launchd rewrite
 - no more bootstrap namespaces
- Everything is now XPC based

Final notes

- Improve userspace 64-bit ASLR!
 - heap spraying shouldn't be this effective
- Provide a mechanism for more granular sandboxing of Mach services
- Ubuntu runs really nicely on Apple hardware!

More Info:

<https://www.mikeash.com/pyblog/friday-qa-2009-01-16.html>

<http://nshipster.com/inter-process-communication/>

[http://adcdownload.apple.](http://adcdownload.apple.com/wwdc_2012/wwdc_2012_session_pdfs/session_241_cocoa_interprocess_communication_with_xpc.pdf)

[com//wwdc_2012/wwdc_2012_session_pdfs/session_241_coc
oa_interprocess_communication_with_xpc.pdf](http://adcdownload.apple.com/wwdc_2012/wwdc_2012_session_pdfs/session_241_cocoa_interprocess_communication_with_xpc.pdf)

“Mac OS X and iOS Internals - To The Apple's Core” - J. Levin

“Mac OS X Internals: A Systems Approach” - A. Singh

<https://code.google.com/p/google-security-research/issues/>