Tales from the iOS/macOS Kernel Trenches

Zer0con 2022

@jaakerblom

These slides were presented at Zer0con 2022 in Seoul, South Korea. They are meant for a highly technical audience. Details, clarifications and elaborations made during the presentation are not included in the slides. Some links have been added.

Abstract

The finding and exploitation of vulnerabilities in Apple's macOS and especially iOS operating systems, not least in their kernels, has long been a very hot topic in the information security community. Yet the actual details of a lot of vulnerabilities and exploitation techniques discovered never get publicly published after they are patched, meaning they often remain nothing more than a CVE number or patch note - if even that - in the public's eye. This talk will be diving into the technical details of some of the recently patched vulnerabilities and exploitation techniques for the iOS and macOS kernels that haven't made it into the spotlight.

Agenda

- Recent iOS/macOS Vulnerabilities
- Recent Exploitation Primitives and Techniques
- Exploitation
- (Recent Kernel Mitigations on iOS 15.2-15.5)
- Conclusion

Recent iOS/macOS Vulnerabilities

Recent iOS/macOS Vulnerabilities

- This talk skips many details of the bugs and techniques covered and I may move through the content quite fast
- Some parts assume deep knowledge of iOS/macOS kernel internals to understand the content
- If you have questions about something left out or that you felt was glossed over, feel free to ask

Recent iOS/macOS Vulnerabilities

- IOMobileFramebuffer (Kernel Driver): CVE-2021-30883
- IOMobileFramebuffer (Kernel Driver): CVE-2021-30983, CVE-2021-30985, CVE-2021-30991
- Kernel: CVE-2021-30937

IOMobileFramebuffer: CVE-2021-30883

Patched in: iOS 15.0.2

IOMobileFramebuffer: CVE-2021-30883

- Affects only iOS (not macOS) as IOMobileFramebuffer is an iPhone specific driver
- IOMobileFramebuffer is a highly device specific driver with many differences between different iPhone models
- This means IOMobileFramebuffer vulnerabilities typically affect only some devices
- CVE-2021-30883 affects iPhone 8/X -> iPhone 11

IOMobileFramebuffer: CVE-2021-30883

- The bug is in set_block selector of IOMobileFramebuffer
- This selector is used to adjust many settings of the screen of the iPhone, including its colors
- Accessible from both app and WebContent/Safari sandbox at the time of this patch (usable for typical 1-click chain)

IOMobileFramebuffer: CVE-2021-30883

- Apart from set_block being an interesting attack vector, it has been providing any app and WebContent the ability to change the colors of the iPhone screen
- Such changes are system wide and permanent until next reboot
- This is a side note, but I find it questionable whether apps(/websites) really have a legitimate need for the ability to system wide turn the screen completely black - user can’t do anything with the phone except reboot in such case

IOMobileFramebuffer: CVE-2021-30883

- Very shortly after 15.0.2 was released, Saar Amar of MSRC posted a write up with some technical details of this bug
- To summarize: it's an integer overflow leading to heap overflow
- A PoC causing a kernel panic showing an overflow has occurred in default/kext.kalloc.80 is included in the write up

```
memset(input, 0x41, input_size);
int *pArr = (int*)input;

pArr[0] = 0x3;          // sub-sub selector
pArr[1] = 0xffffffff;   // has to be non-zero
pArr[2] = 0x40000001;   // #iterations in the outer loop (new_from_data)
pArr[3] = 2;
pArr[8] = 2;
pArr[89] = 4;           // #iterations in the inner loop (set_table)

/* each call trigger a flow with a lot of calls to set_table(), while
   each set_table() flow will do a loop of only 4 iterations*/
for (size_t i = 0; i < 0x10000; ++i) {
    ret = IOConnectCallMethod(iomfb_uc, 78,
                        scalars, 2,
                        input, input_size,
                        NULL, NULL,
                        NULL, NULL);
}
```

IOMobileFramebuffer: CVE-2021-30883

- No public exploit for the bug has been posted after this (as far as I know)
- Let’s delve a bit deeper

IOMobileFramebuffer: CVE-2021-30883

- Even without examining the kernel code, it is possible to cause overflows in zones of other sizes than just 80 by only slightly modifying the input struct in Saar Amar’s PoC
- The result of such modifications can clearly be shown without any advanced debugging tools
- Just examining the panic logs and looking at the size of the zone corrupted by the input struct currently tested is enough

IOMobileFramebuffer: CVE-2021-30883

```
panic(cpu 4 caller 0xffffffff0083441dc): [default.kalloc.80]: element modified after free
    0: 0x4141414141414141
    8: 0x4141414141414141
```

IOMobileFramebuffer: CVE-2021-30883

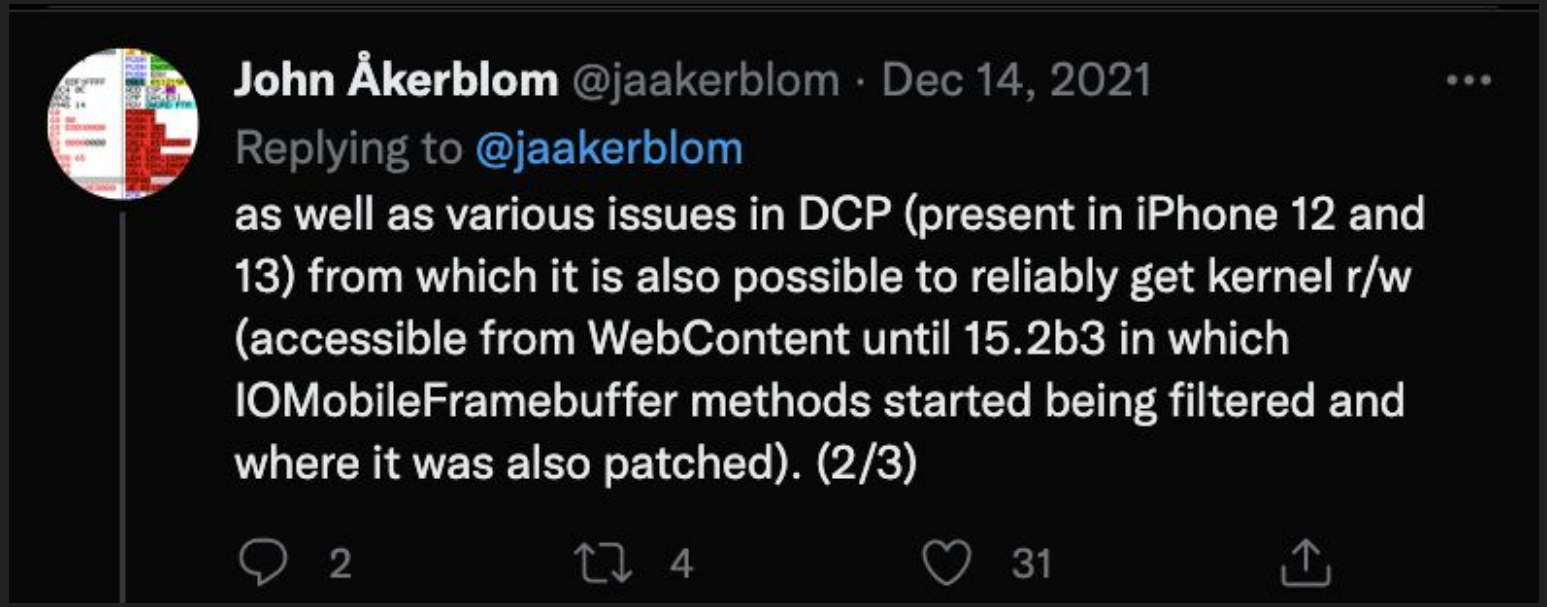
- Such testing reveals that if a too large size is chosen, the data to be overwritten seemingly cannot be controlled, at least not in the same way as for smaller sizes
- Even without much investigation, basic testing reveals the ability to cause overflows with controlled data in various default.kalloc zones, e.g default.kalloc.512
- Is this already enough to write an exploit obtaining kernel r/w, working on iOS 15, perhaps even using only publicly known techniques?
- Re-visited later in this talk if there is time

IOMobileFramebuffer:
CVE-2021-30983
CVE-2021-30985
CVE-2021-30991

Patched in: iOS 15.2

(Grouped together as CVE-2021-309XX from now)

IOMobileFramebuffer: CVE-2021-309XX



John Åkerblom @jaakerblom · Dec 14, 2021

Replying to @jaakerblom

as well as various issues in DCP (present in iPhone 12 and 13) from which it is also possible to reliably get kernel r/w (accessible from WebContent until 15.2b3 in which IOMobileFramebuffer methods started being filtered and where it was also patched). (2/3)

2 4 31

IOMobileFramebuffer: CVE-2021-309XX

- Again IOMobileFramebuffer, affecting iOS
- It is not public knowledge exactly which CVE corresponds to which of these bugs, I grouped them together as best I could (may be slightly incorrect)
- As mentioned before, IOMobileFramebuffer bugs are typically device specific
- CVE-2021-309XX affect iPhone 12 and iPhone 13 only

IOMobileFramebuffer: CVE-2021-309XX

- The bugs are reached from the set_block selector, just like CVE-2021-30883

- set_block was still accessible from Safari until iOS 15.2 beta 3, so these bugs were usable in 1-click attacks at the time they were patched

IOMobileFramebuffer: CVE-2021-309XX

- On iPhone 12 and 13, IOMobileFramebuffer is mostly just a wrapper passing data to and from the new DCP hardware present in these devices
- These bugs are actually in the firmware of DCP as opposed to in the iOS kernel image itself
- Exploiting them requires reverse engineering the DCP firmware and writing an exploit for its processor

IOMobileFramebuffer: CVE-2021-309XX

- The DCP firmware has only a limited set of mitigations
- Notably mitigations such as ASLR and PAC are not present
- Old school ROP techniques are possible

IOMobileFramebuffer: CVE-2021-309XX

- The DCP firmware provides the ability to read from and write to the kernel memory of the main application processor

- Successful exploitation of DCP leads to traditional kernel r/w primitives

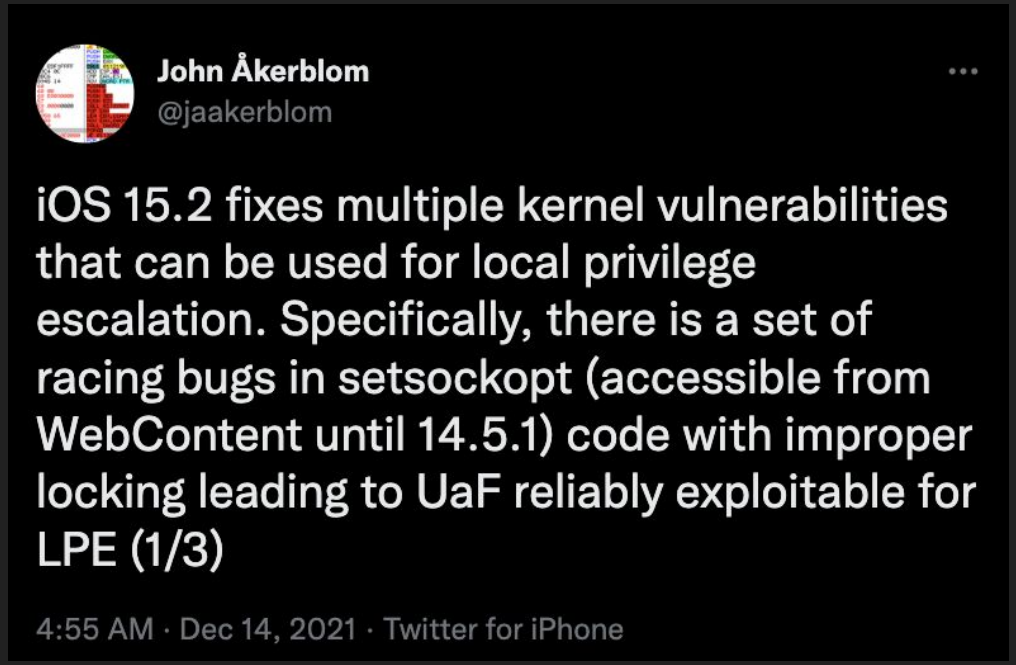
IOMobileFramebuffer: CVE-2021-309XX

- There are practically no similarities in the exploitation of these bugs and the other bugs covered in this presentation - they do not even affect the same processor
- They are also unaffected by practically all kernel mitigations (only by sandbox/entitlements)
- 盘古 used these bugs to win Tianfu cup 2021
- I will leave it to them or someone else to further cover the deeper details of these bugs and their exploitation

Kernel: CVE-2021-30937

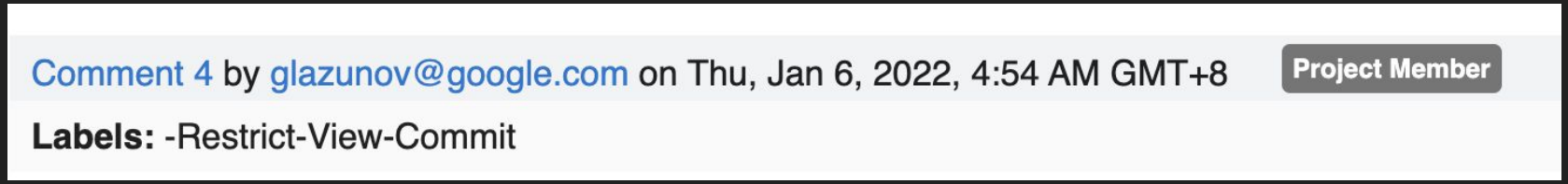
Patched in: iOS 15.2

Kernel: CVE-2021-30937



Kernel: CVE-2021-30937

- A few weeks after the release of iOS 15.2, Sergei Glazunov (who reported this bug) removed the view restriction on its Project Zero bug tracker issue



Kernel: CVE-2021-30937

- The issue page contains a basic description of the bug as well as a description of a couple of primitives that can be obtained by leveraging it
- It also includes an accompanying PoC

Kernel: CVE-2021-30937

- The report seems not to have gotten very much public attention compared to other bugs reported by Google Project Zero
- Perhaps it could be worthy of a slightly deeper look

Kernel: CVE-2021-30937

- To summarize the Project Zero report: the bug is a locking/racing issue in setsockopt leading to various UaF issues

- Since the bug is in xnu, it affects both iOS and macOS

Kernel: CVE-2021-30937

- Reachable from app and WebContent/Safari sandbox until iOS 14.5.1, thereafter only from app and other contexts
- Has been around since at least iOS 10 / macOS Sierra (confirmed by source diffing)

Kernel: CVE-2021-30937

- In his report, Sergei describes the KASAN Use-after-Free panic he triggered on macOS in the inp_join_group function of in_mcast.c in the xnu source code
- Since the bug is a UaF, we will immediately panic also on release iOS if we cause a UaF access on a page that has been unmapped

Kernel: CVE-2021-30937

- “Winning the race” and causing such a panic on unmapped memory in iOS leads to a crash landing us at the location on the following slide

- I will be summarizing the important part

```
v61 = *v43;
if ( *v43 )
{
  do
  {
    v62 = v61;
    v61 = *v61;
  }
  while ( v61 );
  while ( 1 )
  {
    while ( 1 )
    {
      v63 = v62;
      *(v62 + 0x1C) = *(v62 + 0x1D);
      v64 = *(v62 + 8);
      if ( !v64 )
        break;
      do
      {
        v62 = v64;
        v64 = *v64;
      }
      while ( v64 );
    }
    v65 = *(v62 + 0x10);
```

<- Crash here

```
    v65 = *(v62 + 0x10);
    v62 = v65 ? v65 & 0xFFFFFFFFFFFFFFFELL : 0LL;
    if ( !v62 )
      break;
    if ( *v62 != v63 )
    {
      while ( 1 )
      {
        v66 = v63;
        v63 = v62;
        if ( v66 != *(v62 + 8) )
          break;
        v67 = *(v62 + 16);
        if ( v67 )
          v62 = v67 & 0xFFFFFFFFFFFFFFFELL;
        else
          v62 = 0LL;
        if ( !v62 )
          goto LABEL_169;
      }
    }
  }
}
:
v7 = 0LL;
*(v43 + 0x10) = *(v43 + 0x11);
```

<- Continuation of left side

Kernel: CVE-2021-30937

- I originally had a lot of slides explaining what's going on here in detail as there's a lot to unpack but I've cut most slides
- I will focus on the primitives gained by actually exploiting the bug
- The most important part are these two assignments in the decompilation:

```
*(v62 + 0x1C) = *(v62 + 0x1D);
*(v43 + 0x10) = *(v43 + 0x11);
```

Kernel: CVE-2021-30937

- The UaF is in default.kalloc and these two assignments grant the ability to:

  a) Have an inline modification performed in whatever we refill the UaF buffer with

  b) Have a modification performed at **any** address of our choice, provided that we are able to refill the UaF buffer with controlled data

- I will be focusing on b) in this talk

Kernel: CVE-2021-30937

- Back to the decompilation, this is how that second modification looks like (v62 is of type (_BYTE *)):

```
*(v62 + 0x1C) = *(v62 + 0x1D);
```

- If we do a refill of the UaF buffer with controlled data, we end up controlling the v62 pointer here
- This means we effectively gain a byte copy primitive, where we can copy a byte at **any** address of our choice over the byte before it

Kernel: CVE-2021-30937

- The big caveat is that whatever is at +0x0, +0x8 and +0x10 of v62 will get dereferenced and used (bad) if these are not NULL
- There are still ways to exploit and even make use of such cases, but in this presentation I will be focusing on the targeting of something where those are indeed NULL
- In this way, we just do the byte copy modification and call it a day
- This is a VERY powerful primitive even with these constraints

Kernel: CVE-2021-30937

- The remaining essential thing to cover about this targeted byte copy primitive is how to make the required controllable default.kalloc spray

- It is a variably sized buffer originally in default.kalloc.768, growable to default.kalloc.1664, default.kalloc.4096 and larger sizes

Kernel: CVE-2021-30937

- On iOS 13 and below there is a large number of controlled spray options available for refilling this - OOL data descriptors, pipe buffers (with e.g size 4096) and IOSurface data spray to name a few
- For iOS 14 there is much less to choose from, but some fairly obvious options still present in setsockopt, such as SO_NECP_ATTRIBUTES spray

Kernel: CVE-2021-30937

- By the time of iOS 15.1.1 there really weren’t many options left
- There was however still something lurking in the necp subsystem

```
v10 = kalloc_ext(&KHEAP_DEFAULT, v5 + 0x368,
if ( !v10 )
  panic("_MALLOC: kalloc returned NULL (poten
v11 = v10;
v12 = v10 + 0x368;
v13 = copyin(*(a3 + 32), (v10 + 0x368), v5);
```

Kernel: CVE-2021-30937

- This code in necp_client_add allows us to spray allocations in default.kalloc.1664 with arbitrarily controlled data at the offset needed to control v62 from the earlier slide
- Our requirements are satisfied and when tested this indeed works as a raceable refill for our bug
- Targeted byte copy achieved :)

Kernel: CVE-2021-30937

- Finally, a note on reliability
- When we “lose” the race by not triggering the locking issue at all, nothing bad happens and we can retry
- When we trigger the bug but don’t refill in time we also will typically not panic and can retry - explanation kept out for brevity
- All of this is to say: the primitive is reliable on iOS 15

Recent (Generic) Exploitation Primitives and Techniques

Recent (Generic) Exploitation Primitives and Techniques

- Recent Generic Kernel R/W Primitives
- Recent Generic Intermediary Primitives (used to reach kernel r/w)
- Other Recent Generic Techniques

Recent Generic Kernel R/W Primitives

Recent Generic Kernel R/W Primitives

- ipc_port/tfp0 - heavily mitigated over time
- pipe buffers - PAC’d in iOS 14.2
- uio (underrated) - PAC’d in iOS 15.0
- **IOSurfaceClient** - I’ll be covering this one

IOSurfaceClient Kernel R/W

- @pattern_F_ demonstrated usage of IOSurfaceClient objects for stable repeatable kernel r/w on iOS 14 last year
- The response to this by Apple came in iOS 15.0, when they blocked the specific method selectors used publicly by pattern_F_ from the Safari sandbox in iOS 15
- However, they left other methods unblocked, some of which could still be used for kernel r/w from WebContent

IOSurfaceClient Kernel R/W

- The main idea behind most IOSurfaceClient kernel r/w implementations is to corrupt/control the array of IOSurfaceClients owned by an IOSurface userclient

- This is a pointer array of variable size (depending on how many IOSurfaceClients have been added) allocated in KHEAP_KEXT

IOSurfaceClient Kernel R/W

- Various selectors make use of the IOSurfaceClient array to do gets and sets of data through the IOSurfaceClient pointers in it

- Redirecting those gets and sets by controlling the IOSurfaceClient pointers in the array leads to kernel r/w being achieved

Controlling IOSurfaceClient array

- One obvious generic way to control the IOSurfaceClient array is to cause it to be kfree'd using some kind of free primitive and then refill it with controlled or semi-controlled data
- The free primitive part will be re-visited later, but assuming we had one, what would we refill with?
- First, let's take a brief look at the code for the setYCbCrMatrix method, usable for arbitrary write from WebContent

```
v7 = *(*(this + 0x118) + 8LL * a2);
if ( v7 )
  v6 = IOSurfaceClient::setYCbCrMatrix(v7, a3);
```

```
__int64 __fastcall IOSurfaceClient::setYCbCrMatrix(uint6
{
  return IOSurface::setYCbCrMatrix(*(this + 0x40), a2);
}
```

```
__int64 __fastcall IOSurface::setYCbCrMatrix(
{
  *(this + 0xB4) = a2;
  *(*(this + 0xC0) + 0x1CLL) = a2;
  *(this + 0x26E) = a2;
  *(*(this + 0xC0) + 0x6ALL) = a2;
  return 0LL;
}
```

Controlling IOSurfaceClient array

- One way to get kernel r/w is to control the IOSurfaceClient pointer itself and all subsequent dereferences
- In order to do so, we would need a good large controllable spray in KHEAP_KEXT
- Used to be easy, but on iOS 15 not so easy anymore with the progressive removal of more and more sprays

Controlling IOSurfaceClient array

- Alternatively, if we are able to control the pointer at +0x40 (IOSurface pointer) of the IOSurfaceClient elements in the array, we are also good as there’s enough levels of indirection to do arbitrary write

- I’ll be covering this alternative way

Controlling IOSurfaceClient array

- There is another userclient than the IOSurface userclient which is somewhat conceptually similar to it - IOGPU
- The IOGPU userclient also manages objects in arrays in a similar fashion to to how the IOSurface userclient manages the IOSurfaceClient array
- The IOGPU arrays are also of variable size and allocated in KHEAP_KEXT with the same sizes we need

Controlling IOSurfaceClient array

- In particular, IOGPU provides a selector for creating IOGPUCommandQueue objects and adding them to the corresponding array
- A string provided in the input struct when calling this selector is copied into the created IOGPUCommandQueue object, starting at offset +0x10
- The string is null terminated, but otherwise we have arbitrary control of the content meaning we have the ability to craft a pointer at +0x40 (we do not need NULL bytes)

Controlling IOSurfaceClient array

- In conclusion, we can achieve kernel r/w by using a free primitive on an IOSurfaceClient array that is still in use and then refilling it with a IOGPUCommandQueue array

- Next, I will be talking about how to achieve the required free primitive

Recent Generic Intermediary Primitives

How to kfree on iOS 15

Recent Generic Intermediary Primitives

- When it comes to looking for an arbitrary kfree primitive, the answer on iOS has always been to look no further than ipc_kmsg

- I consider this object somewhat of an exploit SDK as it has been possible to use it as the cornerstone of exploits for years

ipc_kmsg kfree in a nutshell

- ipc_kmsg contains member ikm_header
- ikm_header points to a header followed by the content of the message
- The message can contain descriptors with default.kalloc pointers
- If these pointers are faked/overwritten, free primitive is achieved, as they will later be free’d

ipc_kmsg kfree in a nutshell

- iOS 14.2 introduced PAC signature verification for kmsgs in the form of ikm_validate_sig
- This leads to a panic when receiving kmsgs that have been modified after their creation (signature mismatch)

ipc_kmsg kfree in a nutshell

- However, no call to ikm_validate_sig was added to the destroy path called when the owning port of a kmsg is destroyed
- This means ipc_kmsg free primitive has been business as usual even after iOS 14.2, including on iOS 15
- That’s all for the ipc_kmsg free primitive

Other Recent Generic Techniques

How to know where controlled kernel memory is

Other Recent Generic Techniques

- Weak kernel memory randomization has plagued iOS for years
- In fact, with iOS 14 and iOS 15 introducing more zone isolation it seems it became even worse

Other Recent Generic Techniques

- Even on iOS 15 it has been possible to simply hardcode static kernel memory addresses that will always be reliably hit by sufficiently large sprays
- ‘Fresh after boot’ reliability is trivial to test - it’s reliable on iOS 14 and iOS 15.1.1 without a doubt - this is easily verifiable
- As for phones running a long time and other ‘real world’ scenarios - I welcome anyone to disprove the reliability of this technique in such cases and would be glad if they did
- I fear this has been a very effective technique even for many ‘real world’ scenarios

Other Recent Generic Techniques

- This technique goes hand-in-hand with the techniques described previously
- We are able to know where large sprays of controlled data and/or specific objects are in memory
- Therefore we can point the IOSurfaceClient member to controlled data and know where our kmsg spray is
- A “complete” set of generic techniques for exploitation on iOS 15 have now been covered

Exploitation

Putting the parts together

Exploitation: CVE-30883

- To exploit the first bug covered in this presentation, CVE-2021-30883, it is sufficient to use to the static hardcoded address trick and overflow over an IOSurfaceClient array
- Writing a fake IOSurfaceClient pointer to controlled data there leads to kernel r/w
- That’s it really, the rest is just heap shaping

Exploitation: CVE-30937

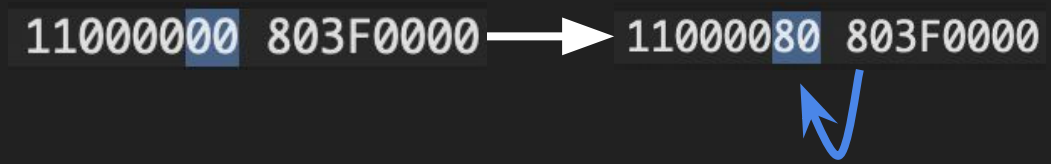
- To exploit the setsockopt bug, CVE-2021-30937, we would succeed if we managed to corrupt a kmsg and gained a free primitive to do the IOSurfaceClient array vs IOGPUCommandQueue array overlap trick

Exploitation: CVE-30937

- kmsg corruption can be done by turning a non-complex kmsg into a complex one (i.e setting bits 0x80000000 - MACH_MSGH_BITS_COMPLEX)
- In that case, the controllable inline data of a message without any real descriptors will be interpreted as if it was in fact real descriptors
- It is then trivial to fake descriptors with targets to free

Exploitation: CVE-30937

- The header starts with the message bits followed by the controllable size
- One way to achieve the corruption is to copy one 0x80 byte from the controllable size to the message bits



- MACH_MSGH_BITS_COMPLEX (0x80000000) has now been set and the free primitive will occur when the port owning the kmsg is destroyed

Exploitation: CVE-30937

- There was also the caveat about +0, +0x8 and +0x10 “needing” to be NULL
- The message bits and size fields are at the very start of the allocation for large kmsgs
- When aligning the +0x1C offset of the bug to the message bits, +0, +0x8 and +0x10 all fall in the previous allocation
- These will always be NULL when we spray kmsgs after each other, so we simply avoid this extra logic

Exploitation: CVE-30937

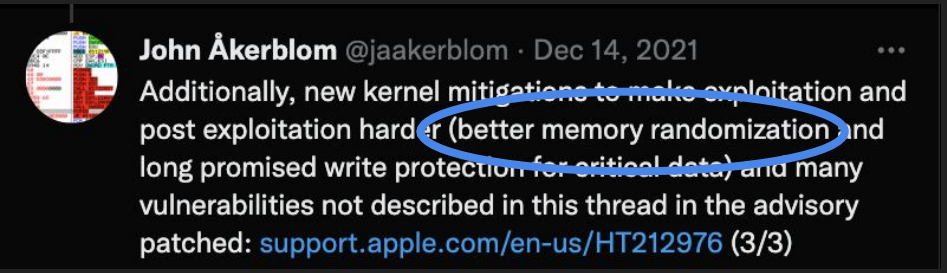
- All main parts needed for an exploit of CVE-30937 have been covered
- Putting them together indeed works and leads to kernel r/w :)
- I might open source an implementation of this soon - no promises (edit: available here)
- This bug can be used to jailbreak iOS 10 - iOS 15.1.1

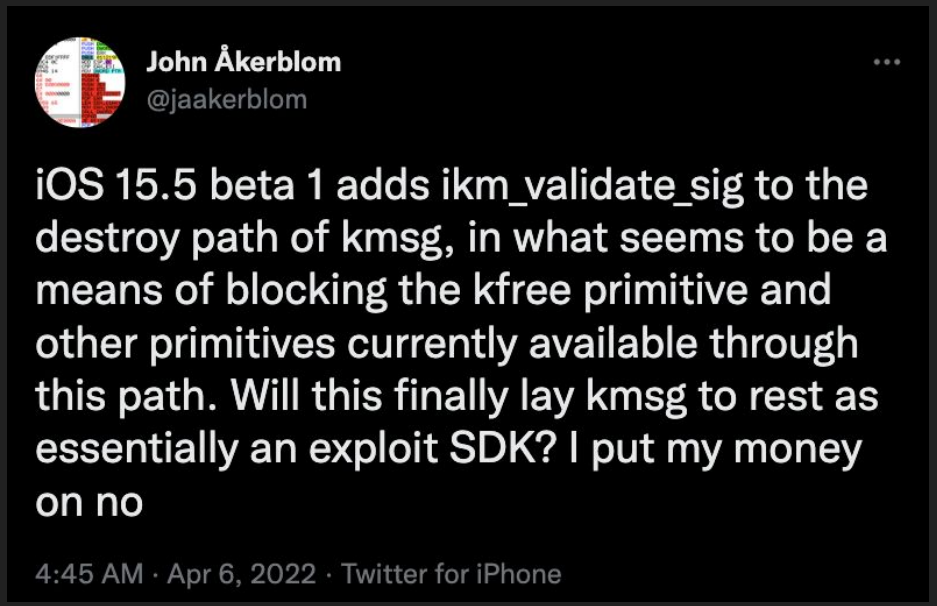
(Recent Kernel Mitigations on iOS 15.2-15.5)

(Recent Kernel Mitigations on iOS 15.2-15.5)

- I put this in parentheses as covering recent mitigations is not the main purpose of this talk
- However, I will briefly mention some mitigations relevant to the primitives and techniques previously covered

(Recent Kernel Mitigations on iOS 15.2-15.5)





(Recent Kernel Mitigations on iOS 15.2-15.5)

- In iOS 15.2, more kernel memory randomization was finally added, drastically reducing the reliability of statically hardcoding kernel memory address predictions
- This means for example a hardcoded address that would always land in KHEAP_DATA_BUFFERS on 15.1.1 could now e.g land in KHEAP_KEXT instead
- Whether zones start at their beginning or end is also random (50-50), making overflows and OOB vulnerabilities trickier to exploit

(Recent Kernel Mitigations on iOS 15.2-15.5)

- In iOS 15.3, mitigations for IOSurfaceClient were added
- PAC signing was added to the IOSurfaceClients in the IOSurfaceClient array
- Various back references checks also added to prevent IOSurfaceClient/IOSurface faking

(Recent Kernel Mitigations on iOS 15.2-15.5)

- In iOS 15.5 beta 1, PAC signing validation was finally added for the destroy path of kmsg
- The kfree primitive covered in this talk now triggers a kernel panic
- Finally, there were many more mitigations in these versions, but that’s it for this talk

Conclusion

Conclusion

- Some recent iOS/macOS kernel vulnerabilities/techniques that previously haven’t seen much/any spotlight were presented
- The bar for iOS kernel exploitation, at least as far as gaining kernel r/w is concerned, has arguably been relatively low in recent years
- However, Apple has started adding significant mitigations at a faster pace than before, killing many existing techniques
- Only time will tell the long term effectiveness of these in preventing real world attacks

Thanks to

- Phạm Hồng Phi (@4nhdaden)
- Ki Chan Ahn (@externalist)
- Zer0con organizers and sponsors