iOS vulnerabilities technical details

CVE-2017-6979

A race condition vulnerability in the IOSurface.kext driver allows an attacker to bypass the sanity checks for the creation of an IOSurface object.

The function *IOSurfaceRoot::createSurface* is responsible for the creation of the IOSurface object. It receives an OSDictionary, which it forwards to the function *IOSurface::init*.

IOSurface::init parses the properties and in case one of these are invalid (e.g, a width that exceeds 32 bits), returns 0, and the creation of the IOSurface is halted.

The IOSurfaceRoot object must hold a lock while calling *IOSurface::init* because *IOSurface::init* adds the IOSurface object to the IOSurfaceRoot's list of surfaces.

Figure 1 shows code that calls *IOSurface::init*:

Figure 1 - Creation of an IOSurface object and handling a failure

In case the *IOSurface::init* function fails, *IORecursiveLockUnlock* will be called.

A bogus IOSurface object will still be in the system and in the IOSurfaceRoot's list of surfaces (thus accessible to everyone).

At this particular moment, an attacker can increase the refcount of the IOSurface (creating, for instance, an IOSurfaceSendRight object attached to the surface) and prevent the bogus IOSurface object from being destroyed.

This leads to the creation and existence of an IOSurface in the kernel which the attacker controls its properties (IOSurface->width = -1 for example). Such an IOSurface object can be given to other mechanisms in the kernel which might rely on a valid width/height/one of the properties to work, thus causing heap overflows/other problems that might lead to an elevation of privileges of the attacker.

CVE-2017-6989

A vulnerability in the AppleAVE.kext kernel extension allows an attacker to drop the refcount of any IOSurface object in the kernel.

Selector numbers 4 and 7 in the AppleAVE2UserClient interface (DriverPreInit and SetSessionSettings) receives multiple IOSurface IDs from the user.
The supplied IDs are then given to the IOSurfaceRoot's function IOSurfaceRoot::lookup_surface (suggested name) that will either return NULL or the surface's kernel pointer.

The result from IOSurfaceRoot::lookup_surface will be then further used by the kernel, unless the user supplied the kernel pointer himself.

In that particular case, the kernel ignores the IOSurface ID supplied by the user (it will only validate that the ID is not 0) and will take the user-supplied pointer and use it as an IOSurface object.

Figure 2 demonstrates how the kernel skips the call to IOSurfaceRoot::lookup_surface when a user-supplied kernel pointer exists:

Figure 2 - The kernel accepts a usermode-supplied kernel pointer

Providing that the attacker did supply a valid IOSurface pointer, the refcount of that IOSurface object is not increased.

The kernel will later utilize this IOSurface object. Upon freeing resources from that AppleAVE object (by calling selector number 2 of AppleAVEUserClient), some of the refcounts of those IOSurface objects will be decreased.

Figure 3 - cleanup_stuff drops the refcount of any chosen IOSurface object provided that the user knows the kernel address of the object.

This allows an attacker to drop the refcount of any IOSurface he knows (or arbitrarily call any function with offset 0x78 from a vtable of an object).

CVE-2017-6994

An information disclosure vulnerability in the AppleAVE.kext kernel extension allows an attacker to leak the kernel address of any IOSurface object in the system.

Selector number 7 in the AppleAVE2UserClient interface (kAVEUserClientDriverPreInit) receives multiple IOSurface IDs from the user. The supplied IDs are then given to the IOSurfaceRoot's function IOSurfaceRoot::lookup_surface (suggested name) that will either return NULL or the surface's kernel pointer. The following picture demonstrates one of the IDs being given and checked later for success or failure:

Figure 4 - the kernel looks up the address of an IOSurface object by its ID which is supplied by the user

Shortly afterwards, the function copies some of these kernel pointers back to the output buffer, which is then sent directly to user mode (with the kernel pointers included):

Figure 5 - The kernel return the kernel pointer of a chosen IOSurface object

This allows an attacker to retrieve the kernel pointer of any IOSurface object, ultimately helping to bypass PAN\kASLR.

CVE-2017-6995

A type confusion vulnerability in the AppleAVE.kext kernel extension allows an attacker to send an arbitrary kernel pointer which will be used by the kernel as a pointer to a valid IOSurface object.

Selector numbers 4 and 7 in the AppleAVE2UserClient interface (DriverPreInit and SetSessionSettings) receives multiple IOSurface IDs from the user.
The supplied IDs are then given to the IOSurfaceRoot's function IOSurfaceRoot::lookup_surface (suggested name) that will either return NULL or the surface's kernel pointer.
The result from IOSurfaceRoot::lookup_surface will be then further used by the kernel, unless the user supplied the kernel pointer himself. In that particular case, the kernel will simply ignore the IOSurface ID supplied by the user (it will only validate that the ID is not NULL) and will take the user-supplied pointer and use it as an IOSurface object.
The snippet below shows how the kernel skips the call to IOSurfaceRoot::lookup_surface when a user-supplied kernel pointer exists:

Figure 6 - A user-supplied pointer is treated just as if it were a valid IOSurface kernel pointer

The function will later use that pointer as an IOSurface object. This behavior happens a lot on both selectors 4 and 7.

This allows an attacker to hijack the kernel execution and therefore execute code in the context of the kernel.

CVE-2017-6996

An attacker can leak sensitive kernel memory.

Selector numbers 4, 7 and 8 in the AppleAVE2UserClient interface (DriverPreInit, SetSessionSettings and ResetBetweenPasses) receives multiple IOSurface IDs from the user. Those selectors eventually lead to the same function (address 0xFFFFFFF006B91A98 on a compiled kernelcache on iPhone 7, iOS 10.3.1).

This function maps the IOSurface's buffers to the kernel in the function CreateBufferFromIOSurface.

From that mapping, a "FrameInfo" object is created. One of the parameters that is supplied by FrameInfo is "InfoType". If the attacker supplies InfoType 0x4569, a pointer is taken from the backed IOSurface's buffer, which the user completely controls:

Figure 7 - A completely controlled user pointer is taken from the IOSurface mapped memory

The offset 0x11D8 depends on the kernel version.
The attacker's pointer is later passed to the function MapYUVInputFromCSID:

Figure 8 - The kernel takes a user-controlled pointer and transfers it to MapYUVInputFromCSID

The function then performs multiple operations on the attacker's pointer, leading both to a memory corruption and to information disclosure.

The function allows the attacker to put a newly allocated 0x70 bytes block in any kernel-pointer:

Figure 9 - The kernel puts in a user-controlled pointer, a kernel-allocated memory

Because controllable_pointer might be a pointer to data that is controlled by an attacker, it is possible to have an arbitrary kernel read primitive here:

The attacker can simply modify *controllable_pointer after the call to initialize_IOSurfaceBufferMgr and change it to an arbitrary address. The kernel will read from that address and put its content into the mapped (and user accessible) data:

Figure 10 - The kernel takes an address from a user-controlled pointer, reads from that address, and put its content back into the IOSurface buffer

CVE-2017-6997

An attacker can free any pointer of size 0x28.

Selector numbers 4, 7 and 8 in the AppleAVE2UserClient interface (DriverPreInit, SetSessionSettings and ResetBetweenPasses) receives multiple IOSurface IDs from the user. Those selectors eventually lead to the same function (address 0xFFFFFFF006B91A98 on a compiled kernelcache on iPhone 7, iOS 10.3.1).

This function maps the IOSurface's buffers to the kernel in the function CreateBufferFromIOSurface.

From that mapping, a "FrameInfo" object is created. One of the parameters that is supplied by FrameInfo is "InfoType". If the attacker supplies InfoType 0x4569, a pointer is taken from the backed IOSurface's buffer, which the user completely controls.

If the attacker supplies no pointer, the kernel allocates data and puts it in kernel_address + 0x11D8 (the offset 0x11D8 depends on the kernel version).

Figure 11 - The kernel takes an address from a user-controlled memory

Later on, while in the InfoType 0x4569 handler, that pointer is freed **even if the attacker supplied his own pointer:**

Figure 12 - The kernel frees a user-controlled address

This allows the attacker to free any pointer of size 0x28, causing a memory corruption or denial of service in case the attacker supplies a pointer which does not belong to the kalloc.40 zone.

CVE-2017-6998

An attacker can hijack kernel code execution due to a type confusion

Selector numbers 4, 7 and 8 in the AppleAVE2UserClient interface (DriverPreInit, SetSessionSettings and ResetBetweenPasses) receives multiple IOSurface IDs from the user. Those selectors eventually lead to the same function (address 0xFFFFFFF006B91A98 on a compiled kernelcache on iPhone 7, iOS 10.3.1).

This function maps the IOSurface's buffers to the kernel in the function CreateBufferFromIOSurface.

From that mapping, a "FrameInfo" object is created. One of the parameters that is supplied by FrameInfo is "InfoType". If the attacker supplies InfoType 0x4569, a pointer is taken from the backed IOSurface's buffer, which the user completely controls.

Figure 13 - The kernel takes an address from a user-controlled memory

The offset 0x11D8 depends on the kernel version (on iPhone 7, iOS 10.3.1 it is 0x16B0). The attacker's pointer is later passed to the function address 0xFFFFFFF006B8C50C (kernelcache on iPhone 7, iOS 10.3.1):

Figure 14 - Usage of the user-controlled address

The function checks whether the first pointer within our controllable pointer is not NULL, and if so, it transfers the pointer to 0xFFFFFFF006B97C24 (iPhone 7, iOS 10.3.1):

Figure 15 - Further usage of the user's controlled address

0xFFFFFFF006B97C24 is a very short function that takes our pointer, extracts another pointer from offset 0x58 and uses it as an object. It uses its vtable, which allows an attacker to hijack the kernel flow:

Figure 16 - A vtable is fetched from the user-controlled address, and a function is being called from that vtable

CVE-2017-6999

A user-controlled pointer is zeroed.

Selector numbers 4, 7 and 8 in the AppleAVE2UserClient interface (DriverPreInit, SetSessionSettings and ResetBetweenPasses) receives multiple IOSurface IDs from the user. Those selectors eventually lead to the same function (address 0xFFFFFFF006B91A98 on a compiled kernelcache on iPhone 7, iOS 10.3.1).
This function maps the IOSurface's buffers to the kernel in the function CreateBufferFromIOSurface.
From that mapping, a "FrameInfo" object is created. One of the parameters that is supplied by FrameInfo is "InfoType". If the attacker supplies InfoType 0x4569, a pointer is taken from the backed IOSurface's buffer, which the user completely controls.
If the pointer is not NULL, its first 0x28 bytes are zeroed.
This can be seen here:

Figure 17 - A user controlled address is being NULLed

Afterwards, a pointer is taken from that buffer, which the user completely controls.
If the pointer is not NULL, its first 0x28 bytes are zeroed. This allows an attacker to nullify the first 0x28 bytes of any kernel pointer.