Jailbreaking

Apple Watch
whoami

- Security researcher at Lookout
- Lead researcher on Pegasus exploit chain
- Focused on advanced exploitation techniques
- Fried Apple team co-founder
- iOS/tvOS/WatchOS jailbreak author

July 27-30, 2017
What is Apple Watch?

- Released in 2015
- Apple S1/S2 processor
- ARMv7k 32 bit architecture
- 512 MB RAM
- WatchOS
Why to pwn a watch?

- It's fun 😊
- Access to sensitive user data
- Run tools like radare or frida on a watch
- iPhone attack vector
Apple Watch security

- Secure boot chain
- Mandatory Code Signing
- Sandbox
- Exploit Mitigations
- Secure Enclave Processor (2-nd generation only)
- Data Protection
Possible attack vectors

- Malformed USB descriptor (cable required)
- Malformed email, message, photo, etc
- Application extension based
Attack plan

- Leak kernel base
- Dump whole kernel
- Find gadgets and setup primitives
- Disable security restrictions
- Run ssh client on a watch
Bugs of interest

- CVE-2016-4656 - osunserialize bug
- CVE-2016-4669 - mach_port register
- CVE-2016-7644 - set_dp_control_port
- CVE-2017-2370 - voucher extract recipe
Leaking kernel slide

- CVE-2016-4655 and CVE-2016-4680
- Object constructor missing bounds checking
- OSNumber object with high number of bits
- Object length used to copy value from stack
- Kernel stack memory leaked
- Can be triggered from an app’s sandbox
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
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;
}
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, ...) {
...

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
    }
    // We control this value
    // Will be returned to userland
    // Points to stack based buffer
}

July 27-30, 2017
CVE-2016-4656 exploitation

- Kernel mode UAF in OSUnserializeBinary
- OSString object deallocated
- retain() called on deallocated object
- Fake object with fake vtable → code exec
- Problem: kernel dump required
- Bonus: we can deref any address via fake vtable

July 27-30, 2017
OSObject * OSUnserializeBinary(const char *buffer, size_t bufferSize, ...) {

newCollect = isRef = false;
...

case kOSSerializeDictionary:
    \textbf{o} = newDict = OSDictionary::withCapacity(len);
    newCollect = (len != 0);
    break;
...

    if (!isRef)
    {
        \textbf{setAtIndex}(objs, objsIdx, \textbf{o});
        if (!ok) break;
        objsIdx++;
    }

Save object to \textit{objs} array

```c
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
```
Object saved to objs array destroyed
```

Deallocated object retained
```
Deallocated object retained
```

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

- No WatchOS kernel dumps in public
- No keys for WatchOS 2.x kernels
- Idea: read kernel as OSString chunks
- vtable offset required to fake OSString
- vtable stored in __DATA.__const in kernel
Getting vtable - __DATA.__const leak

- __DATA.__const address is in Mach-O header
- Kernel base + 0x224 == __DATA.__const
- Deref and branch address via fake vtable

---

```c
; Sections

section <"__nl_symbol_ptr", "__DATA", 0x80001158, 0, 0, 0, 0, 0>
section <"__mod_init_func", "__DATA", 0x80001158, 0, 0, 9, 0, 0>
section <"__mod_term_func", "__DATA", 0x8000119C, 0, 0, 0xA, 0, 0>
section <"__const", "__DATA", 0x800011E0, 0, 0, 0, 0, 0>
section <"__data", "__DATA", 0x80001224, 0, 0, 0, 0, 0>
section <"__data", "__DATA", 0x80001224, 0, 0, 0, 0, 0>
section <"__data", "__DATA", 0x80001268, 0, 0, 0, 0, 0>
```
Getting vtable - known offset

- Get vtable offset from similar XNU build
- Known delta from __DATA.__const start
- Tune address with +/- delta

```
__DATA:__ const:803ECE8C __ZTV80SString
__DATA:__ const:803ECE8C ; `vtable for 'OSString
__ZTV80SString DCB 0
__DATA:__ const:803ECE8C __ZTV80SString DCB 0
__DATA:__ const:803ECE8D DCB 0
__DATA:__ const:803ECE8E DCB 0
__DATA:__ const:803ECE8F DCB 0
DATA: const:803ECE90 DCB 0
```

July 27-30, 2017
Getting vtable - known offset

- Get vtable offset from similar XNU build
- Known delta from __DATA.__const start
- Tune address with +/- delta

FAIL

```
_DATA: __const: 803ECE8C __ZTV80SString
_DATA: __const: 803ECE8C __ZTV80SString
_DATA: __const: 803ECE8C __ZTV80SString
_DATA: __const: 803ECE8C __ZTV80SString
_DATA: __const: 803ECE8D __ZTV80SString
_DATA: __const: 803ECE8E __ZTV80SString
_DATA: __const: 803ECE8F __ZTV80SString
_DATA: const: 803ECE90 DCB 0
```
Getting vtable – OSString layout

OSString 32 bit

<table>
<thead>
<tr>
<th>Address</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>vtable ptr + 0x8</td>
</tr>
<tr>
<td>0x4</td>
<td>retain count</td>
</tr>
<tr>
<td>0x8</td>
<td>flags</td>
</tr>
<tr>
<td>0xC</td>
<td>length</td>
</tr>
<tr>
<td>0x10</td>
<td>string ptr</td>
</tr>
</tbody>
</table>

OSString 64 bit

<table>
<thead>
<tr>
<th>Address</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>vtable ptr + 0x8</td>
</tr>
<tr>
<td>0x8</td>
<td>retain count</td>
</tr>
<tr>
<td>0x8</td>
<td>flags</td>
</tr>
<tr>
<td>0x10</td>
<td>length</td>
</tr>
<tr>
<td>0x18</td>
<td>string ptr</td>
</tr>
</tbody>
</table>
OSString layout

OSString vtable pointer

OSObject::retain() offset

32
Getting vtable – next free node trick

- vtable ptr is first 8 bytes of a object
- OSString size is 0x20 (64 bit)
- retain() off is vtable start + 0x20 (64 bit)
- Next node ptr as deallocated object vtable
- Idea - retain() OOB to next node in freelist
- If next node is OSString – branch object vtable
kalloyc.32 freelist

Freelist head

Next node pointer

32
Getting vtable – next free node trick

- Heap spray OSString objects
- Make dozen OSDictionaries with OSString
- Trigger OSDictionary deallocation
- retain() -> deref next free chunk pointer
- Free chunk is surrounded by OSStrings
- retain() -> OOB branch to next OSString node
Heap spray and OOB branch to vtable

- Branch out of bounds to next node
- Allocated OSString object
- Deallocated OSString object
Getting vtable – next free node trick

- Heap spray OSString objects
- Make few OSDictionary by with OSString
- Trigger OSDictionary deallocation
- retain() -> deref next free chunk pointer
- Free chunk is surrounded by OSStrings
- retain() -> OOB branch to next OSString node

... on 32 bits 😞
Getting vtable – dump over panic

- **OSString** vtable reference in **OSUnserialize** 😊
- We can deref any address as fake vtable ptr

```assembly
OSUnserializeBinary
  PUSH     {R4-R7,LR}
  ADD      R7, SP, #0xC
  PUSH.W   {R8,R10}
  MOV      R5, R0
  MOVVS    R0, #0x14 ; this
  MOV      R8, R1
  BL       __ZN8OSObjectnwEm ; OSObject::operator new(ulo;
  MOVW     R1, #:lower16:(`__ZTV8OSString - 0x8031A9C0);
  MOV      R4, R0
  MOV      R1, #:upper16:(`__ZTV8OSString - 0x8031A9C0);
  MOV      R0, #(`__ZN8OSString10gMetaClassE - 0x8031A9C2)
  ADD      R1, PC ; \"vtable for \"OSString
```
Getting vtable – dump over panic

- Crash in OSUnserializeBinaryXML
- Get LR register value from panic
- Use fake vtable to deref LR value
- Get panic from a watch
- We just dump 4 bytes of a kernel 😊
Getting vtable – dumping kernel

- Use address to leak as vtable_addr - 0x10
- We need to tune by `retain()` offset in vtable
- Crash and get panic log
- Copy panic from Watch to iPhone -> Mac
- Parse panic, grab 4 bytes of a kernel TEXT
- Update address with 4 bytes delta and repeat
Getting vtable – final steps

- Crash in OSUnserializeXML
- Leak opcode until ‘BL OSUnserializeBinary’
- Leak OSUnserializeBinary opcodes
- Finally get OSStr ing vtable offset
Getting vtable – final steps

OSUnserializeBinary address

OSUnserializeBinary offset

MOV
STR
BL

R0, R10
R2, [SP, #0x88+var_34]

__Z19OSUnserializeBinaryPKcmPP8OSString ; OSUnserializeBinary
loc_8031533A

MOV
MOV
BL
MOVW
MOV
MOV
ADD

R8, R1
__ZN8OSObjectnwEm ; NSObject::operator new(ulong)
R1, #:lower16(__ZTV8OSString - 0x80311340) ; `vtable for 'OSString
R4, R0
R1, #:upper16(__ZTV8OSString - 0x80311340) ; `vtable for 'OSString
R0, #(__ZN8OSString10gMetaClassE - 0x80311342) ; OSString::gMetaClass
R1, PC ; `vtable for 'OSString ; `vtable for 'OSString

July 27-30, 2017
Getting vtable – final steps

- 5 minutes for recover watch after crash
- 5 minutes to fetch panic from watch
- 2 minutes to copy to Mac and parse
- No way to automate a process
- It takes me two weeks to dump vtable
Next step – full kernel dump

- Now use fake OSString obj to read kernel
- Read data via IORegistryEntryGetProperty
- Leak kernel header, calculate kernel size
- Dump full kernel to userland by chunks
Next step – kernel symbolication

- Find and list all kexts
- Find sysent and resolve syscalls
- Find and resolve mach traps
- Resolve IOKit objects vtable
Next step – setting up primitives

- Scan kernel dump for gadgets
- Set up exec primitive
- Set up kernel read & write primitives
Next step – kernel structs layout

- Look for `proc_*` functions
- Restore `proc` structure layout
- Dump memory, check for known values
Next step – kernel structs layout

- memmem string \byte pattern
- + xref + instruction analysis
- Resolve syscalls table, mach traps table
- Simple instruction emulation
Getting root and sandbox bypass

- Patch `setreuid` (no KPP 😊)
- `patch ucred in proc structure in kernel`
- `patch sandbox label value in ucred`
Getting kernel task

- Patch `task_for_pid()`
- Or save kernel `self` in task bootstrap port
- Read it back via `task_get_special_port()`
- Restore original bootstrap port value

July 27-30, 2017
Disable codesign checks

- Patch _debug to 1
- patch _nl_symbol_ptr (got) entries
- Patch amfi variables
  - cs_enforcement_disable
  - allow_invalid_signatures
Remount rootfs

- Patch __mac_mount
- Change flags in rootfs vnode and mount RW
- Patch lwvm is_write_protected check
- Patch PE_i_can_has_debugger in lwvm
Spawning ssh client

- Compile dropbear for ARMv7k
- Compile basic tools package for ARMv7k
- **Problem**: More sandbox restrictions
- Kill WatchOS specific sandbox operations
ssh connection problem...

- WatchOS interfaces

```
"awdl0/ipv6" = "fe80::c837:8aff:fe60:90c2";
"lo0/ipv4" = "127.0.0.1";
"lo0/ipv6" = "fe80::1";
"utun0/ipv6" = "fe80::face:5e30:271e:3cd3";
```
Watch <-> iPhone port forwarding

```cpp
NSDictionary *comm = @{
    @"Command" : @"StartForwardingServicePort",
    @"ForwardedServiceName" : @"com.apple.syslog_relay",
    @"GizmoRemotePortNumber" : [NSNumber numberWithUnsignedShort: pt],
    @"IsServiceLowPriority" : @0,
};

AMDSServiceConnectionSendMessage(serviceConnection, (__bridge CFPropertyListRef)(comm), kCFPropertyListXMLFormat_v1_0);

AMDServiceConnectionReceiveMessage(serviceConnection, &response, (CFPropertyListFormat*)&format);

NSNumber *iphone_port = response["CompanionProxyServicePort"];
```

Thanks to Luca 😊

July 27-30, 2017
ssh connection over bluetooth

[!] Setting up bluetooth proxy on a watch
[+] Device connected, binding port 22 on watch to an iPhone port 50308
[+] Port binded, now use port 50308 on your iPhone device
[+] Setting up iproxy with local port 5444 and iPhone port 50308
[+] Done. Now ssh to local port 5444 to access watch
[!] Waiting for connection

maxbazaliy~» ssh root@localhost -p 5444
The authenticity of host '[localhost]:5444 ([127.0.0.1]:5444)' can't be established.
ECDSA key fingerprint is SHA256:SCM/doXH/pnJVn6dnHz6An/ZbEYAPMWREQlx2ucplgY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:5444' (ECDSA) to the list of known hosts.
root@localhost's password:
-she-3.2# uname -a
Darwin Apple-Watch 15.4.0 Darwin Kernel Version 15.4.0: Fri Feb 19 13:32:35 PST 2016; root:xnu-3248.41.4~27/RELEASE_ARM_S7002 Watch1,2

July 27-30, 2017
maxbazaliy->~/Desktop/Watch_jailbreak_demo


Watch as a spyware target

- Watch have access to SMS, Calls, Health
- Photos and emails synced to Watch
- Fetch GPS location from the phone
- Microphone usage
- Apple Pay 😊
Dumping Messages, Contacts, Emails

- Just dump from sqlite DB or de-serialize data
  - private/var/mobile/Library/AddressBook/
  - private/var/mobile/Library/NanoMailKit/
  - private/var/mobile/Library/SMS/

- Hook on fly on device sync\notification
Dumping Calendar, Passes, pair info

- Just dump from sqlite DB or de-serialize data
  - private/var/mobile/Library/Health/
  - private/var/mobile/Library/Caches/
  - private/var/mobile/Library/Application Data/

- Hook on fly on device sync\notification
Dumping Health, Caches, Photos

- Just dump from sqlite DB or de-serialize data
  - private/var/mobile/Library/Health/
  - private/var/mobile/Library/Caches/
  - private/var/mobile/Library/Application Data/

- Hook on fly on device sync notification
Call recording

- AudioToolbox.framework exists but not public
- Add observer on CTTelephonyCenter
- Catch kCTCallStatusChange in a callback
- Hook AudioUnitProcess function
- Create file via ExtAudioFileCreateWithURL
- Use ExtAudioFileWrite to dump call data
References

- Stefan Esser - iOS 10 - Kernel Heap Revisited
- Luca Todesco - com.apple.companion_proxy client
- Lookout - Technical Analysis of the Pegasus Exploits on iOS
- Siguza - tfp0 powered by Pegasus
@mbazaliy