Hacking from iOS 8 to iOS 9

TEAM PANGU
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Who We Are

- Team Pangu is known for releasing jailbreak tools for iOS 7.1, iOS 8, and iOS 9
- We have broad security research interests
- Our research was present at BlackHat, CanSecWest, POC, RuxCon, etc.
- We also co-organize a mobile security conference named MOSEC (mosec.org) with POC
iOS Security Overview

- Apple usually releases a white paper to introduce iOS security architecture
  - Isolations
  - Restricted Sandbox
  - Mandatory Code Signing
  - Exploit Mitigation (ASLR, DEP)
  - Data Protection
  - Hypervisor
Timeline of Major Security Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhoneOS 1.x</th>
<th>iPhoneOS 2.x</th>
<th>iOS 4.3</th>
<th>iOS 6</th>
<th>iOS 7</th>
<th>iOS 8</th>
<th>iOS 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Protection</td>
<td>Code Signing</td>
<td>ASLR</td>
<td>KASLR</td>
<td>TouchID</td>
<td>TeamID</td>
<td>KPP</td>
<td></td>
</tr>
</tbody>
</table>
Many security features are undocumented
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Improved Team ID Validation

- Team ID was introduced in iOS 8
- Prevent platform binaries from loading third-party code
- iOS 9 enforces that a process either is a platform binary or has a team identifier

```c
prog_teamID = csproc_get_teamid_16(v11);
prog_platform = csproc_get_platform_binary_16();
v23 = prog_teamID == 0;
if ( !prog_teamID )
  v23 = prog_platform == 0;
if ( v23 )
{
  v17 = "[deny-mmap] main process has no team identifier in its signature";
goto LABEL_17;
}
```
DYLD Environment Variables

- DYLD environment variables affect the dynamic linker dyld in many ways
  - Output debug info (e.g., through DYLD_PRINT_*)
  - Dylib injection (e.g., through DYLD_INSERT_LIBRARIES)
- iOS 8.3 starts to ignore DYLD environment variables unless the main executable has certain entitlements
Released Source Code of dyld

By default, ignoreEnvironmentVariables is false

checkEnvironmentVariables will not ignore DYLD environment variables
dyld on iOS 8.3

- `ignoreEnvironmentVariables` is set True according to v108

```c
IgnoreEnvironmentVariables = 0;
v26 = &v115;
LOBYTE(dyld::sProcessIsRestricted) = 0;
v129 = -1;
if ((v108 & 0x1004) == 4096 )
    IgnoreEnvironmentVariables = 1;
dyld::checkEnvironmentVariables(envp, IgnoreEnvironmentVariables);
```

- Where is v108 from?
dyld on iOS 8.3

- v108 indicates the code signing status of the program
- CSOPS is used to query the code signing attributes

```c
if ( csops(0, 0, &csStatus, (void *)4) )
{
    v129 = -1;
    dyld::throwf((dyld *)"failed to get code signing flags", (const char *)0xFFFFFFF);
}

v15 = (char *)dword_1FE26464;
v108 = *(DWORD *)&csStatus;
```
dyld on iOS 8.3

- $\text{v108} \& 0x1004 = 4096$

- $0x0004$ means that the program has get-task-allow entitlement

In other words, DYLD environment variables only work for binaries that have the get-task-allow entitlement

```c
/* code signing attributes of a process */
#define CS_VALID 0x0000001 /* dynamically valid */
#define CS_ADHOC 0x0000002 /* ad hoc signed */
#define CS_GET_TASK_ALLOW 0x0000004 /* has get-task-allow entitlement */
#define CS_INSTALLER 0x0000008 /* has installer entitlement */
#define CS_HARD 0x0000100 /* don't load invalid pages */
#define CS_KILL 0x0000200 /* kill process if it becomes invalid */
#define CS_CHECK_EXPIRATION 0x0000400 /* force expiration checking */
#define CS_RESTRICT 0x000800 /* tell dyld to treat restricted */
#define CS_ENFORCEMENT 0x001000 /* require enforcement */
#define CS_REQUIRE_LV 0x002000 /* require library validation */
#define CS_ENTitlements.Validate 0x004000
```
DYLD Environment Variables

Consequence:

- neagent is the only program on iOS that is allowed to load third party signed libraries (ignoring the TeamID validation because of the com.apple.private.skip-library-validation entitlement)

- The trick to force neagent load an enterprise license signed library through the DYLD_INSERT_LIBRARIES no longer works
enable-dylibs-to-override-cache

- The present of this file was used to force loading of dynamic libraries from filesystem instead of the shared cache
- It was widely used by previous jailbreak tools to override the libmis library
- dyld in iOS 8.3 starts to ignore this flag
**enable-dylibs-to-override-cache**

- The kernel disallows to check the present of the flag

This value is read from 0xFFFFF4084, an address in the kernel and read only in userspace.
Reduced TOCTOU Time Window in iOS 9

- dyld is responsible for loading dynamic libraries and probing to test if the libraries are signed correctly.
Reduced TOCTOU Time Window in iOS 9

- dyld is responsible for loading dynamic libraries and probing to test if the libraries are signed correctly.

```c
try {
    // record info about file
    image->setFileInfo(info_st_dev, info_st_ino, info_st_mtime);

    // if this image is code signed, let kernel validate signature before mapping any pages from image
    image->loadCodeSignature(codeSigCmd, fd, offsetInFat, context);

    // mmap segments
    image->mapSegments(fd, offsetInFat, lenInFat, info.st_size, context);

    // probe to see if code signed correctly
    image->crashIfInvalidCodeSignature();
}
```

Many segment overlapping tricks were used in the past to bypass the subsequent code signing checks.
Reduced TOCTOU Time Window in iOS 9

- dyld on iOS 9 now validates the mach-o header (first pages) before mapping segments into the memory
Changes in Loading Launchd Daemons

- xpcd_cache.dylib is used to store plist files of launchd daemons
  - All plist files are encoded in the dylib and thus protected by signatures
  - Before iOS 9, by using a fake xpcd_cache.dylib (e.g., masking the __xpcd_cache segment as readonly), jailbreak tools can easily customize the launchd daemons
Changes in Loading Launchd Daemons

For example, launchd on iOS 8.4 loads the bplist in following way. Masking the __xpcd_cache segment readonly does not cause any problem

```c
if ( lstat("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", &v27))
{
v26 = 0;
v3 = dlopen("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", 2);
if (v3)
{
v4 = dlsym(v3, "_xpcd_cache");
if (v4)
{
    if (dladdr(v4, &v25))
    {
        v5 = getsectiondata(v25.dli_fbase, "_TEXT", "_xpcd_cache", &v26);
        if (v5)
            v7 = xpc_create_from_plist(v5, v26, v6);
        else
            v7 = xpc_dictionary_create(0, 0, 0);
        dword_36C54 = v7;
    }
}
}
```
Launchd on iOS 9 will first invoke a trivial API in `xpcd_cache.dylib` to ensure the present of executable permission.

```c
if ( lstat("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", &v29) )
{
  v28 = 0;
  v3 = dlopen("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", 2);
  if ( v3 )
  {
    v4 = dlsym(v3, "__xpcd_cache");
    v5 = v4;
    if ( v4 )
    {
      if ( ((int (__cdecl *)(void *, int *, int))v4)(v4, v1, v2) != 1 )
        ABEL_38:
        v26 = _os_assert_log(0, 0);
        _os_crash(v26);
        __debugbreak();
      }
      if ( dladdr(v5, &v27) )
      {
        v6 = getsectiondata(v27.dli_fbase, "__TEXT", "__xpcd_cache", &v28);
        if ( v6 )
          v8 = xpc_create_from_plist(v6, v28, v7);
        else
          v8 = xpc_dictionary_create(0, 0, 0);
      }
  }
}
```
Changes in Loading Launchd Daemons

- Launchd on iOS 9 only loads platform binaries
- Launchd uses csops to query the status of code signing attributes of the process
Changes in loading launchd daemons

- Non-platform binary cannot be launched

```c
if ( csops(v26, 0, &v45, 4) )
{
    result = (int *)(__error);
    if ( result != (int *)3 )
    {
        if ( *__error() )
        {
            v30 = __os_assumes_log();
            __os_avoid_tail_call(v30);
        }
        goto LABEL_83;
    }
} else
{
    result = v45;
    if ( !((unsigned int)v45 & 0x4000000) )
    {
        EL_83:
        sub_223C4((int)"unexpected exec of non-platform binary");
        goto LABEL_84;
    }
}
```
Changes in Loading Main Executable

- The iOS kernel is responsible for parsing and loading the main executable while creating a new process.
Changes in Loading Main Executable

- Before iOS 8.3, the kernel does not directly validate the signature of the Mach-O header of the main executable.
  - Kernel only ensures that the main executable has a correct code signature segment (i.e., the segment is signed correctly).
- Instead, the kernel leaves the validation to dyld.
  - dyld will access the Mach-O header of the main executable and thus trigger page faults, leading to final SHA1 comparison.
A Persistent Vector for Code signing Bypass before iOS 8.3

- Modify the Mach-O header of a platform binary
  - Change the LC_LOAD_DYLINKER of main executable to trick the kernel to load our fake dyld
  - Modify LC_UNIXTHREAD of our fake dyld which enables us to control all register values and point the PC value to a ROP gadget
In iOS 8.3, the kernel proactively compares the SHA1 of the Mach-O header with the SHA1 in corresponding cs_blob

```c
if (got_code_signatures) {
    unsigned tainted = CS_VALIDATE_TAINTED;
    boolean_t valid = FALSE;
    struct cs_blob *blobs;
    vm_size_t off = 0;

    if (cs_debug > 10)
        printf("validating initial pages of %s\n", wp->v_name);
    blobs = ubc_get_cs_blobs(wp);

    while (off < size && ret == LOAD_SUCCESS) {
        tainted = CS_VALIDATE_TAINTED;
        valid = cs_validate_page(blobs, NULL,
                                file_offset + off,
                                addr + off,
                                &tainted);
        if (!valid || (tainted & CS_VALIDATE_TAINTED)) {
            if (cs_debug)
                printf("CODE SIGNING: %s%d: invalid initial page at offset %lld validated:%d tainted:%d csflags:0x%x\n", wp->v_name, p->p_pid, (long long)(file_offset + off), valid, tainted, result->csflags);
            if (cs_enforcement(NULL) ||
                (result->csflags & (CS_HARD|CS_KILL|CS_ENFORCEMENT))) {
                ret = LOAD_FAILURE;
            }
            result->csflags &= ~CS_VALID;
        }
        off += PAGE_SIZE;
    }
}
```
More Changes in Loading Main Executable

- Actually in iOS 9, Apple adds more check for picking up an already registered cs_blob

```c
if ( v53->csb_cpu_type == v183 )
{
    if ( v53->csb_base_offset == file_offset )
    {
        if ( v53->csb_mem_size == *(DWORD *)(v205 + v48) )
        {
            lck_mtx_lock(v224);
            if ( v224 )
            {
                if ( *(WORD *)(v224 + 68) == 1 )
                {
                    v55 = *(DWORD *)(v224 + 76);
                    if ( v55 )
                    {
                        if ( *(DWORD *)(v55 + 28) == dword_8040AFF4 )
                        {
                            lck_mtx_unlock(v224);
                            goto LABEL_126;  // success
                        }
                    }
                }
            }
        }
    }
}
```
Kernel Patch Protection (KPP)

- Apple introduced KPP in iOS 9 for 64bit devices
- Implementation details are unclear
- It’s believed that it is related to the Secure Enclave Processor (SEP), an alternative of TrustZone on iOS devices
- Unfortunately, the SEP firmware is encrypted
KPP Observations

- KPP randomly checks the integrity of RX pages of the kernel-cache and page table
  - Persistent code patch is not feasible, because it would trigger random kernel panic

- Panic when RX page is modified
  panic(cpu 1 caller 0xffffffff80098fde28): SError esr: 0xbf575401 far: 0xffffffff8009898000

- Panic when Page table is modified
  panic(cpu 0 caller 0xffffffff80214fde28): SError esr: 0xbf575407 far: 0xffffffff8021498000
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Use-after-free in IOHIDResourceUserClient

- We found it by auditing IOHIDFamily source code
- The bug was also independently discovered by other researchers
  - @qwertyoruiop, Cererdlong, etc
- The interesting thing is this bug also affects Mac OS, but is only triggerable with root on Mac OS
  - We almost missed the bug
- Thanks @qwertyoruiop for pointing out that it is triggerable with mobile on iOS
Use-after-free in IOHIDResourceUserClient

- _device is allocated in method 0
- createDevice -> createAndStartDevice

```objective-c
// IOHIDResourceDeviceUserClient::createAndStartDevice

IOReturn IOHIDResourceDeviceUserClient::createAndStartDevice()
{
    IOReturn result;
    OSNumber * number = NULL;

    number = OSDynamicCast(OSNumber, _properties->getObject(kIOHIDRequestTimeoutKey));
    if ( number )
        _maxClientTimeoutUS = number->unsigned32BitValue();

    // If after all the unwrapping we have a dictionary. Let's create the device
    _device = IOHIDUserDevice::withProperties(_properties);
    require_action(_device, exit, result=kIOReturnNOResources);
```
Use-after-free in `IOHIDResourceUserClient`

- `_device` is released in method 1
- `terminateDevice -> OSSafeRelease`

```c
// IOHIDResourceDeviceUserClient::terminateDevice

IOReturn IOHIDResourceDeviceUserClient::terminateDevice()
{
    if (_device) {
        _device->terminate();
    }
    OSSafeRelease(_device);

    return kIOReturnSuccess;
}
```
Use-after-free in IOHIDResourceUserClient

- OSSafeRelease is **NOT** safe
  - #define OSSafeRelease(inst) do { if (inst) (inst)-&gt;release(); } while (0)
- It does not nullify the pointer after releasing it!
Use-after-free in IOHIDResourceUserClient

- _device is used again in many functions
  - E.g. method 2 takes 1 input scalar and an input struct, also the the return value is directly passed to user space
  - IOHIDResourceDeviceUserClient::_handleReport

```c
if ( arguments->scalarInput[0] )
    AbsoluteTime_to_scalar(&timestamp) = arguments->scalarInput[0];
else
    clock_get_uptime( &timestamp );

if (!arguments->asyncWakePort ) {
    ret = _device->handleReportWithTime(timestamp, report);
    report->release();
} else {
```
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Context of the UAF

- 32bit
- The UAF object is in the kalloc.192 zone
- Both R1 and R2 are under control when the UAF is triggered

```
LDR.W R0, [R4,#0x80] ; R0=_device
LDR R1, [SP,#0x60+var_40]
LDR R2, [SP,#0x60+var_3C] ; R1,R2=scalar[0]
LDR R3, [R0]
LDR.W R6, [R3,#0x3B4] ; vtable+0x3B4
MOV R3, #0
STR R3, [SP,#0x60+var_60]
STR R3, [SP,#0x60+var_5C]
MOV R3, R5
BLX R6 ; trigger
```
Context of the UAF

- 64bit
- The UAF object is in the kalloc.256 zone
- Only X1 is under control when the UAF is triggered
Transfer UAF to Type Confusion

- The UAF object zone can be easily filled with variety IOUserClient objects via calling IOServiceOpen.

- Check vtable offsets of all possible IOUserClient classes to see what functions we may call.
  - OSMetaClass::serialize(OSSerialize *)
  - OSMetaClass::getMetaClass(void)
  - OSMetaClass::release(void)
  - OSMetaClassBase::isEqualTo(OSMetaClassBase const*)
Exploit Type Confusion to Leak Kernel Slide

- OSMetaClass::getMetaClass(void)
- Return a static object inside kernel -> leak kernel base
- 32bit return value is enough for arm64 also
- High 32bit value is always 0xffffffff80
Exploit Type Confusion to Leak Heap Address

- OSMetaClass::release(void)
  - R0/X0=self pointer -> leak low 32bit of the object address
  - Not enough for arm64
    - High 32bit value is 0xffffffff80 or 0xffffffff81
Exploit Type Confusion to Leak Heap Address for ARM64

- OSMetaClassBase::isEqualTo(OSMetaClassBase const*)

- X1 is under control

- Calling the function twice can decide the high 32bit value of the heap address
Heap Spray with OSData

- What we have now - Kernel base / object address

- `io_service_open_extended` -> `OSUnserializeXML` -> spray OSData with controlled size and content

- Set `[object address] = vtable = object address - call offset + 8`

- When triggering the bug, function pointer at `[object address +8]` will be picked up

- Set `[object address+8] = gadget to call`
The Read Gadget

- 32bit
  - LDR R0, [R1]; BX LR;
- 64bit
  - LDR X0, [X1,#0x20]; RET;
The Write Gadget

- **32bit** - R1 and R2 are under control
  - STR R1, [R2]; BX LR;

- **64bit** - X1 and contents of X0 are controlled
  - LDR X8, [X0,#0x60]; STR X1, [X8,#8]; RET;
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Pangu 9 Released

* Pangu 9 for iOS 9 was released at Oct 14, 2015

* Also we noticed that some people discussed whether Pangu 9 met the Zerodium bounty requirements

Team Pangu, if it’d developed its exploits further, could have made as much as $1 million if it’d submitted its findings to Zerodium, an exploit dealer that had offered that amount to anyone who found and submitted an iOS 9 jailbreak. But as security expert Francisco Alonso told me over Twitter, Pangu would likely not have met Zerodium’s requirements, which asked for exploits to be fully remote. The iOS 9 jailbreak requires the phone to be connected to a PC via USB.
Pangu 9 vs. Bounty Requirement

- We never consider the bounty
- We release the jailbreak tool for
  - Full control of iOS devices for end users
  - Security research and jailbroken iOS development
- We think Mobile Safari is NOT a good landing point for jailbreak tools
  - It’s too dangerous if the exploits are abused, which violates our purpose of releasing a jailbreak tool
  - It will also shorten the lifetime of a jailbreak tool, because Apple will (very likely) release a fix asap
Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Public Release vs. Bounty Hunting
- Conclusion
Conclusion

- Apple puts more efforts on improving the whole security mechanisms rather than fixing individual bugs
- A lot of security features in iOS were undocumented, which make jailbreaking more and more difficult
- KPP introduced in iOS 9 makes people believe that there may be no jailbreak anymore, what we did proves that hackers will always find their way in
Thanks for Your Attention

Q&A