Hacking from iOS 8 to iOS 9

TEAM PANGU

POC 2015

Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- Conclusion

Who We Are

- Team Pangu is known for releasing jailbreak tools for iOS 7.1, iOS 8, and iOS 9

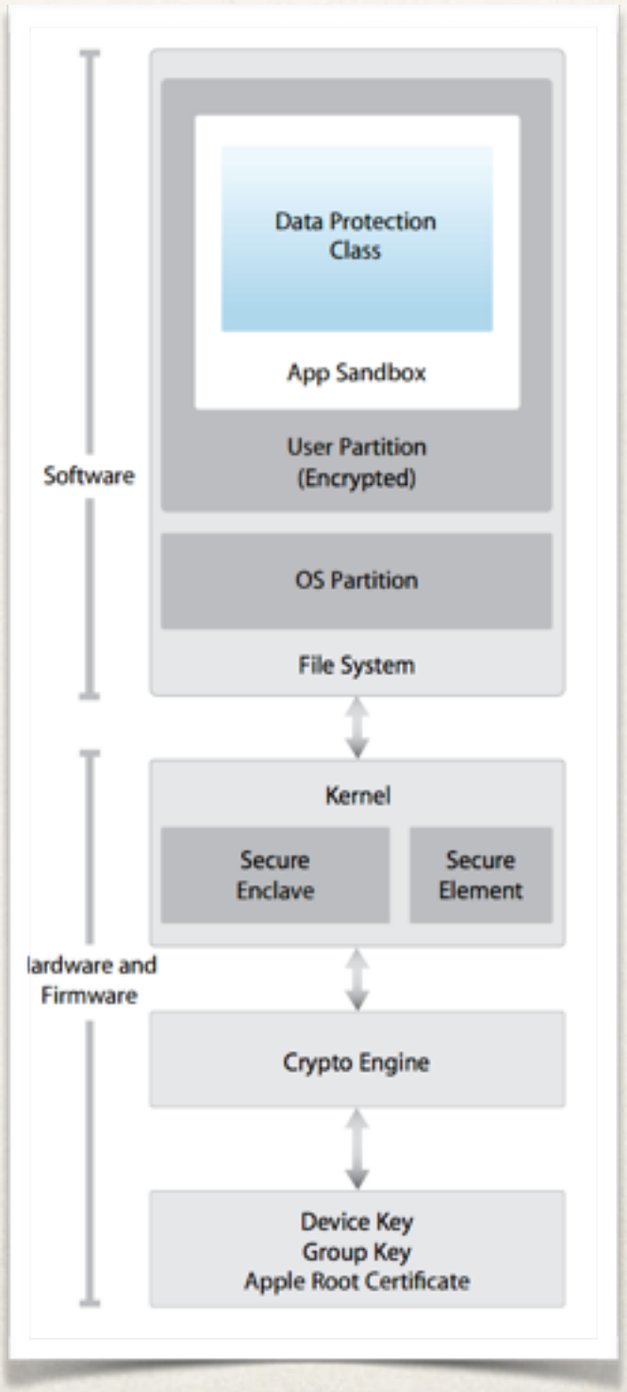
- We have broad security research interests

- Our research was present at BlackHat, CanSecWest, POC, RuxCon, etc.

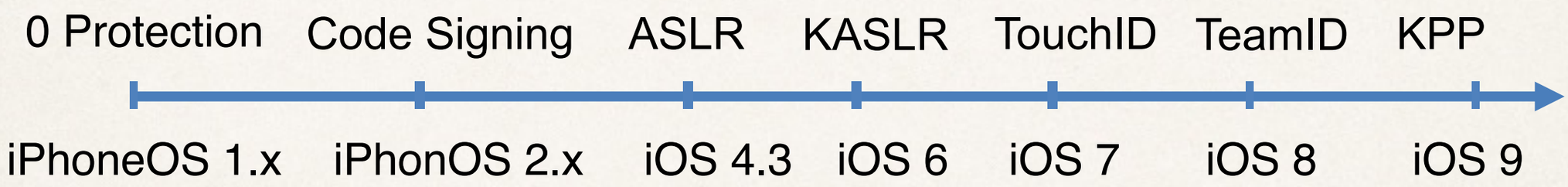
- We also co-organize a mobile security conference named MOSEC (mosec.org) with POC

iOS Security Overview

- Apple usually releases a white paper to introduce iOS security architecture
  - Isolations
  - Restricted Sandbox
  - Mandatary Code Signing
  - Exploit Mitigation (ASLR, DEP)
  - Data Protection
  - Hypervisor



Timeline of Major Security Features



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
- 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

```
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

```
    sExecPath = apple[0];
    bool ignoreEnvironmentVariables = false;
    if ( sExecPath[0] != '/' ) {
        // have relative path, use cwd to make absolute
        char cwdbuff[MAXPATHLEN];
        if ( getcwd(cwdbuff, MAXPATHLEN) != NULL ) {
            // maybe use static buffer to avoid calling malloc so early
            char* s = new char[strlen(cwdbuff) + 1 + strlen(sExecPath) + 1];
            strcpy(s, cwdbuff);
            strcat(s, "/");
            strcat(s, sExecPath);
            sExecPath = s;
        }
    }
    // Remember short name of process for later logging
    sExecShortName = ::strrchr(sExecPath, '/');
    if ( sExecShortName != NULL )
        ++sExecShortName;
    else
        sExecShortName = sExecPath;
    sProcessIsRestricted = processRestricted(mainExecutableMH);
    if ( sProcessIsRestricted ) {
#if SUPPORT_LC_DYLD_ENVIRONMENT
        checkLoadCommandEnvironmentVariables();
#if SUPPORT_VERSIONED_PATHS
        checkVersionedPaths();
#endif
#endif
        pruneEnvironmentVariables(envp, &apple);
        // set again because envp and apple may have changed or moved
        setContext(mainExecutableMH, argc, argv, envp, apple);
    }
    else
        checkEnvironmentVariables(envp, ignoreEnvironmentVariables);
```

By default, ignoreEnvironmentVariables is false

checkEnvironmentVariables will not ignore DYLD environment variables

dyld on iOS 8.3

- ignoreEnvironmentVariables is set True according to v108

```
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

```
if ( csops(0, 0, &csStatus, (void *)4) )
{
  v129 = -1;
  dyld::throwf((dyld *)"failed to get code signing flags", (const char *)0xFFFFFFFF);
}
v15 = (char *)dword_1FE26464;
v108 = *(_DWORD *)&csStatus;
```

dyld on iOS 8.3

- v108 & 0x1004 == 4096
- 0x0004 means that the program has get-task-allow entitlement

```
/* 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          0x0000800 /* tell dyld to treat restricted */
#define CS_ENFORCEMENT            0x0001000 /* require enforcement */
#define CS_REQUIRE_LV             0x0002000 /* require library validation */
#define CS_ENTITLEMENTS_VALIDATED   0x0004000
```

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

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

```
if ( vFFFF4084 & 1 )
{
  v96 = dyld::my_stat(
          (dyld *)"/System/Library/Caches/com.apple.dyld/enable-dylibs-to-override-cache",
          (struct stat *)v127,
          (stat *)v82);
  v97 = 0;
  if ( !v96 )
  {
    v98 = 0;
    if ( v134 < 0x400 )
      v98 = 1;
    if ( (signed int)v135 > 0 )
      v97 = 1;
    if ( !v135 )
      v97 = v98;
  }
  LOBYTE(dyld::sDylibsOverrideCa  ) = v97;
}
```

This value is read from 0xFFFF4084, 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

```
// create image by mapping in a mach-o file
ImageLoaderMachOCompressed* ImageLoaderMachOCompressed::instantiateFrom
                                                    uint64_t
                                                    unsig    nt segCount, unsigned int libCount,
                                                    c        struct linkedit_data_command* codeSigCmd, const LinkContext& context)
{
    ImageLoaderMachOCompressed* image = ImageLoaderMach    mpressed::inst

    try {
        // record info about file
        image->setFileInfo(info.st_dev,    fo.st_ino, info.st_mtime);

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

        // mmap segments
        image->mapSegments(fd, offsetInFat, lenInFat, info.st_size, cont

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

Bind code signature with the vnode of the dylib file

Map segments of the dylib into memory

Trigger page faults to test code signatures

Reduced TOCTOU Time Window in iOS 9

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

```
// create image by mapping in a mach-o file
ImageLoaderMachOCompressed* ImageLoaderMachOCompressed::instantiateFrom
                                                    uint64_t of
                                                    unsigned in
                                                    const struc
{
    ImageLoaderMachOCompressed* image = ImageLoaderMachOCompressed::ins

    try {
        // record info about file
        image->setFileInfo(info.st_dev, info.st_ino, info.st_mtime);

        // if this image is code signed, let kernel vali       gnature before mapping any pages from image
        image->loadCodeSignature(codeSigCmd, fd       etInFat, 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

```
ImageLoader::setFileInfo(v46, v24, v23, v25);
v48 = 2;
ImageLoaderMachO::loadCodeSignature((int)v46, a12, v45, a5, a6, a14);
v48 = 3;
v37 = v43;
v38 = a5;
v39 = a6;
v40 = a14;
ImageLoaderMachO::validateFirstPages(v46, a12, v45, v42);
v26 = *(_QWORD *)(a9 + 80);
v48 = 4;
v37 = a7;
v38 = a8;
*(_QWORD *)&v39 = v26;
v41 = a14;
ImageLoaderMachO::mapSegments(v46, v45, a5, a6);
v48 = 5;
ImageLoaderMachOCompressed::registerEncryption(v46, a13, a14);
v48 = 6;
ImageLoaderMachO::crashIfInvalidCodeSignature(v46);
```

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

```
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;
      }
    }
  }
}
```

Changes in Loading Launchd Daemons

- Launchd on iOS 9 will first invoke a trivial API in xpcd_cache.dylib to ensure the present of executable permission

```
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

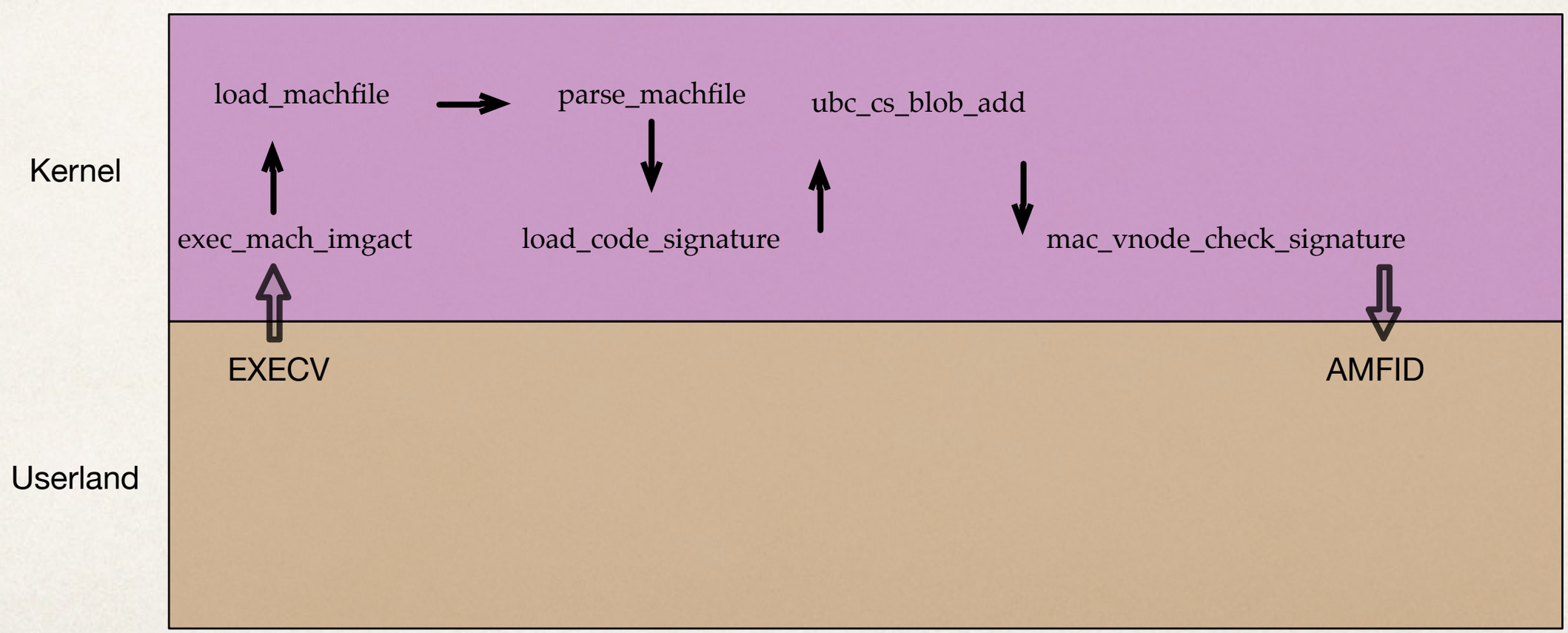
Changes in loading launchd daemons

- Non-platform binary cannot be launched

```
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

Changes in Loading Main Executable

- In iOS 8.3, the kernel proactively compares the SHA1 of the Mach-O header with the SHA1 in corresponding cs_blob

```
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", vp->v_name);
    blobs = ubc_get_cs_blobs(vp);

    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",
                    vp->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;
    } ? end while off<size&&ret==LOAD_S... ?
} ? end if got_code_signatures ?
```

More Changes in Loading Main Executable

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

```
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 0xffffff80098fde28): SError esr: 0xbf575401 far: 0xffffff8009898000
- Panic when Page table is modified

  panic(cpu 0 caller 0xffffff80214fde28): SError esr: 0xbf575407 far: 0xffffff8021498000

Agenda

- iOS Security Overview
- Security Changes from iOS 8 to iOS 9
- Kernel Vulnerability Exploited in Pangu 9
- Kernel Exploit Chain
- 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

```
//---------------------------------------------------------------------------
// 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

```
//--------------------------------------------------------------------
// 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)->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

```
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
- 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
MOVS            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

```
LDR             X0, [X19,#0xE8] ; X0=_device
LDR             X8, [X0]
LDR             X8, [X8,#0x630] ; vtable+0x630
LDR             X1, [SP,#0x28] ; X1=scalar[0]
MOV             W3, #0
MOV             W4, #0
MOV             W5, #0
ADD             X6, SP, #0x10
MOV             X2, X20
BLR             X8              ; trigger
```

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 0xffffff80

```
__ZNK11OSMetaClass12getMetaClassEv
MOV             R0, #(unk_8045FF20 - 0x8030BD34)
ADD             R0, PC ; unk_8045FF20
ADDS            R0, #0x30
BX              LR
```

```
__ZNK11OSMetaClass12getMetaClassEv
ADRP            X8, #unk_FFFFFF800BDA0040@PAGE
ADD             X8, X8, #unk_FFFFFF800BDA0040@PAGEOFF
ADD             X0, X8, #0x340
RET
```

Exploit Type Confusion to Leak Heap Address

- OSMetaClass::release(void)
  - R0/X0=self pointer -> leak low 32bit of an object address
- Not enough for arm64
  - High 32bit value is 0xffffff80 or 0xffffff81

```
__ZNK11OSMetaClass7releaseEv
BX                  LR
```

```
__ZNK11OSMetaClass7releaseEv
RET
```

Exploit Type Confusion to Leak Heap Address for ARM64

- OSMetaClassBase::isEqualTo(OSMetaClassBase const*)

- R1/X1 is under control

  - Calling the function twice can decide the high 32bit value of the heap address

```
__ZNK15OSMetaClassBase9isEqualToEPKS_
CMP                 X0, X1
CSET                W0, EQ
RET
```

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 vtable = object address - call offset + 8
  - Set vtable+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
- 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