Pangu 9 Internals

Tielei Wang & Hao Xu & Xiaobo Chen
Team Pangu
Agenda

- iOS Security Overview
- Pangu 9 Overview
- Userland Exploits
- Kernel Exploits & Kernel Patching
- Persistent Code Signing Bypass
- Conclusion
Who We Are

- A security research team based in Shanghai, China
- Have broad research interests, but known for releasing jailbreak tools for iOS 7.1, iOS 8, and iOS 9
- Regularly present research at BlackHat, CanSecWest, POC, RuxCon, etc.
- Run a mobile security conference named MOSEC ([http://mosec.org](http://mosec.org)) with POC in Shanghai
iOS Security Overview

- Apple usually releases a white paper to explain its iOS security architecture
- Secure Booting Chain
- Mandatory Code Signing
- Restricted Sandbox
- Exploit Mitigation (ASLR, DEP)
- Data Protection
- Hypervisor and Secure Enclave Processor
Agenda

- iOS Security Overview
- Pangu 9 Overview
- Userland Exploits
- Kernel Exploits & Kernel Patching
- Persistent Code Signing Bypass
- Conclusion
What Jailbreak is

“iOS jailbreaking is the removing of software restrictions imposed by iOS, Apple's operating system, on devices running it through the use of software exploits”

–Wikipedia

✦ Jailbreak has to rely on kernel exploits to achieve the goal, because many software restrictions are enforced by the kernel
Kernel Attack Surfaces

Difficulty of Gaining the Privilege

Amount of Kernel Attack Surface Gained

- root with special entitlements
- root, no sandbox
- mobile, no sandbox
- mobile, less restrictive sandbox
- mobile, container sandbox
Our Preference

<table>
<thead>
<tr>
<th>Difficulty of Gaining the Privilege</th>
<th>Amount of Kernel Attack Surface Gained</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobile, no sandbox</td>
<td>root, no sandbox</td>
</tr>
<tr>
<td>mobile, less restrictive sandbox</td>
<td>mobile, container sandbox</td>
</tr>
<tr>
<td>root with special entitlements</td>
<td>root with special entitlements</td>
</tr>
</tbody>
</table>

- Mobile, no sandbox: Low difficulty, low attack surface.
- Mobile, less restrictive sandbox: Higher difficulty, moderate attack surface.
- Mobile, container sandbox: Highest difficulty, highest attack surface.
- Root, no sandbox: Intermediate difficulty, moderate attack surface.
- Root with special entitlements: Highest difficulty, highest attack surface.
Initial Idea and Practice in Pangu 7

- Inject a dylib via the DYLD_INSERT_LIBRARIES environment variable into a system process
- Pangu 7 (for iOS 7.1) leveraged the trick to inject a dylib to timed
- The dylib signed by an expired license runs in the context of timed and exploits the kernel
Team ID Validation in iOS 8

- To kill the exploitation technique, Apple introduced a new security enforcement called Team ID validation in iOS 8.
- Team ID validation is used to prevent system services (aka platform binary) from loading third-party dylibs, with an exceptional case.
- Team ID validation does not work on the main executables with the `com.apple.private.skip-library-validation` entitlement.
Pangu 8’s Exploitation

✧ neagent is a system service which happens to have the entitlement

✧ Pangu 8 mounts a developer disk into iOS devices, and asks debugserver to launch neagent, and specify the DYLD_INSERT_LIBRARIES environment variable

✧ As a consequence, our dylib runs in the context of neagent and exploits the kernel
More Restrictions since iOS 8.3

- iOS 8.3 starts to ignore DYLD environment variables unless the main executable has the get-task-allow entitlement

- Since neagent does not have the get-task-allow entitlement, DYLD_INSERT_LIBRARIES no longer works for neagent
Pangu 9’s Challenge

- Userland
  - We still need to inject a dylib into a system service with less restrictive sandbox profile
- Kernel
- KPP bypass
Agenda

- iOS Security Overview
- Pangu 9 Overview
- **Userland Exploits**
- Kernel Exploits & Kernel Patching
- Persistent Code Signing Bypass
- Conclusion
Userland Exploits

- Arbitrary file read/write as mobile via an XPC vulnerability
- Arbitrary code execution outside the sandbox
REVIEW AND EXPLOIT NEGLECTED ATTACK SURFACES IN IOS 8

The security design of iOS significantly reduces the attack surfaces for iOS. Since iOS has gained increasing attention due to its rising popularity, most major attack surfaces in iOS such as mobile safari and IOKit kernel extensions have been well studied and tested. This talk will first review some previously known attacks against these surfaces, and then focus on analyzing and pointing out those neglected attack surfaces. Furthermore, this talk will explore how to apply fuzzing testing and whitebox code auditing to the neglected attack surfaces and share interesting findings. In particular, this talk will disclose POCs for a number of crashes and memory corruption errors in system daemons, which are even triggerable through XPC (a lightweight inter-process communication mechanism) by any app running in the container sandbox, and analyze and share the POC for an out-of-boundary memory access 0day in the latest iOS kernel.

PRESENTED BY
Tielei Wang & HAO XU & Xiaobo Chen
XPC

- Introduced in OS X 10.7 Lion and iOS 5 in 2011
- Built on Mach messages, and simplified the low level details of IPC (Inter-Process Communication)
xpc_connection_t listener = xpc_connection_create_mach_service("com.apple.xpc.example", NULL, XPC_CONNECTION_MACH_SERVICE_LISTENER);

xpc_connection_set_event_handler(listener, ^(xpc_object_t peer) {
    // Connection dispatch
    xpc_connection_set_event_handler(peer, ^(xpc_object_t event) {
        // Message dispatch
        xpc_type_t type = xpc_get_type(event);
        if (type == XPC_TYPE_DICTIONARY){
            // Message handler
        }
    });
    xpc_connection_resume(peer);
});

xpc_connection_resume(listener);
XPC Client

```c
xpc_connection_t client = xpc_connection_create_mach_service("com.apple.xpc.example", NULL, 0);

xpc_connection_set_event_handler(client, ^(xpc_object_t event) {
    //connection err handler
});

xpc_connection_resume(client);

xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_double(message, "value1", 1.0);

xpc_object_t reply = xpc_connection_send_message_with_reply_sync(client, message);
```
Vulnerability in Assetsd

- Container apps can communicate with a system service named com.apple.PersistentURLTranslator.Gatekeeper via XPC
- assetsd at /System/Library/Frameworks/AssetsLibrary.framework/Support/ runs the service
Path Traversal Vulnerability

- Assetsd has a method to move the file or directory at the specified path to a new location under `/var/mobile/Media/DCIM/`

- Both `srcPath` and `destSubdir` are retrieved from XPC messages, without any validation

```c
v6 = (void *)PLStringFromXPCTreeNode(a3, "srcPath");
v7 = (void *)PLStringFromXPCTreeNode(v4, "destSubdir");
if ( !objc_msgSend(v7, "length") )
{
    \#ELSE_12:
    v8 = 0;
    goto LABEL_13;
}

v8 = 0;
if ( !objc_msgSend(v6, "length") )
{
    v9 = (void *)NSHomeDirectory();
    v10 = objc_msgSend(v9, "stringByAppendingPathComponent:", CFSTR("Media/DCIM");
    v11 = objc_msgSend(v10, "stringByAppendingPathComponent:", v7);
    v21 = 0;
    v12 = objc_msgSend(&OBJC_CLASS_NSFileManager, "alloc");
    v13 = objc_msgSend(v12, "init");
    v14 = objc_msgSend(v13, "autorelease");
    if ( (unsigned int)objc_msgSend(v14, "moveItemAtPath:toPath:error!", v6, v11, &v21) & 0xFF )
    {
```
Exploit the Vulnerability

- Use ".//" tricks in srcPath/destSubdir can lead to arbitrary file reads/writes as mobile

```c
xpc_connection_t client =
xpc_connection_create_mach_service("com.apple.PersistentURLTranslator.Gatekeeper", NULL, 0);
    xpc_connection_set_event_handler(client, ^(void(xpc_object_t response) {
        //NSLog(@"here: %@",response);
    });
xpc_connection_resume(client);
xpc_object_t dict = xpc_dictionary_create(NULL, NULL, 0);

NSString *dstPath = [@"../../../../../.."] stringByAppendingPathComponent:dest];

xpc_dictionary_set_string(dict, "srcPath", [src UTF8String]);
xpc_dictionary_set_string(dict, "destSubdir", [dstPath UTF8String]);
xpc_dictionary_set_int64(dict, "transactionID", 4);
xpc_dictionary_set_int64(dict, "operation", 4);
xpc_object_t reply = xpc_connection_send_message_with_reply_sync(client, dict);
```
More Severe Attack Scenario

- Arbitrary file reads result in severe privacy leaks
- Arbitrary file writes can be transformed into arbitrary app installation, system app replacement, and so on
- Please refer to *MalwAirDrop: Compromising iDevices via AirDrop*, Mark Dowd, Ruxcon 2015 for more details
- Exploitable by any container app
From Arbitrary File Reads/Writes to Arbitrary Code Execution

- Recall that DYLD_INSERT_LIBRARIES only works for the executables with the get-task-allow entitlement
- Who has this entitlement?
No One Holds get-task-allow in iOS 9

• We checked entitlements of all executables in iOS 9, and found no one had the get-task-allow entitlement

• But we found a surprise in developer disk images

```bash
INT80s-MBP:DeveloperDiskImage INT80$ codesign -d --entitlements - ./usr/libexec/vpnagent Executable=/Volumes/DeveloperDiskImage/usr/libexec/vpnagent
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>get-task-allow</key>
  <true/>
  <key>keychain-access-groups</key>
  <array>
    <string>com.apple.identities</string>
    <string>apple</string>
    <string>com.apple.certificates</string>
  </array>
</dict>
</plist>
```
Make Vpnagent Executable on iOS 9

- Mount an old developer disk image (DDI) that contains vpnagent
  - MobileStorageMounter on iOS 9 is responsible for the mount job
- Although the old DDI cannot be mounted successfully, MobileStorageMounter still registers the trustcache in the DDI to the kernel
  - Trustcache of a DDI contains (sort of) hash values of executables in the DDI
  - Trustcache is signed by Apple
- MobileStorageMounter will notify the kernel that vpnagent is a platform binary
  - Old vpnagent can run on iOS 9 without causing code signing failure
Debug Vpnagent

- Mount a normal DDI to enable debugserver on iOS 9
- How the kernel enforces the sandbox profile
  - If the executable is under `/private/var/mobile/Containers/Data/`, the kernel will apply the default container sandbox profile
  - Otherwise, the kernel applies the seatbelt-profile specified in the executable’s signature segment
- Leverage the XPC vulnerability to move vpnagent to some places that debugserver has access to and the kernel does not apply the default sandbox
vpnagent does not have the com.apple.private.skip-library-validation entitlement, so it would not be able to load third party dylib, right?
Bonus of get-task-allow

- Debugging and code signing have a conflict
  - E.g., setting a software breakpoint actually is to modify the code, which certainly breaks the signature of the code page
  - To enable debugging, the iOS kernel allows a process with the get-task-allow entitlement to continually run even if a code signing invalidation happens
Bonus of get-task-allow

- We reuse the code signature of a system binary in our dylib. As a result, when loading the dylib, the kernel believes that vpnagent just loads a system library.

- Team ID Passed

- Code signing validation is *softly* disabled after the kernel finds that the vpnagent with the get-task-allow entitlement is under debugging.

- Code Signing Validation Passed
Put It All Together

- Mount an old DDI to make vpnagent be a platform binary
- Mount a correct DDI to make debugserver available
- Exploit the XPC vulnerability to move a copy of vpnagent to some places that debugserver has access
- Debug the copy of vpnagent, and force it to load our dylib that reuses the code signature segment of a system binary
Agenda

- iOS Security Overview
- Pangu 9 Overview
- Userland Exploits
- **Kernel Exploits & Kernel Patching**
- Persistent Code Signing Bypass
- Conclusion
Attack iOS Kernel

- Gain arbitrary kernel reading & writing
- KASLR / SMAP / ...
- Patch kernel to disable amfi & sandbox
- KPP (Kernel Patch Protection)
Kernel Vulnerability for iOS 9.0

- CVE-2015-6974
- A UAF bug in IOHID
- Unreachable in container sandbox (need to escape sandbox)
- One bug to pwn the kernel
- Details were discussed at RUXCON and POC
Kernel Vulnerability for iOS 9.1

- CVE-2015-7084
- A race condition bug in IORegistryIterator
- Reachable in container sandbox
- One bug to pwn the kernel
- Reported to Apple by Ian Beer
- Exploited by @Lokihardt in his private jailbreak
- Some details at http://blog.pangu.io/race_condition_bug_92/
Kernel Vulnerability for iOS 9.3.3

- CVE-????-???? CVE-2016-4654 (fixed in iOS 9.3.4 this morning)
- A heap overflow bug in IOMobileFrameBuffer
- Reachable in container sandbox
- One bug to pwn the kernel
- Fixed in iOS 10 beta 2
- Details will be discussed in future
Defeat KPP

- What does KPP protect
  - r-x/r-- memory inside kernelcache
  - Code and Const
  - Page tables of those memory

- What does KPP not protect
  - rw- memory inside kernelcache
  - Heap memory
Defeat KPP

- Take a look at Mach-O header of com.apple.security.sandbox
- __TEXT is protected by KPP
- __DATA is not protected by KPP
- __got stores all stub functions address
Defeat KPP

- Both amfi and sandbox are MAC policy extensions
  - Call `mac_policy_register` to setup all hooks
  - Functions pointers are stored in `mac_policy_conf.mpc_ops`
  - Before iOS 9.2 it’s stored in `__DATA.__bss` which is `rw-`
    - Set pointers to `NULL` to get rid of the special hook
  - In iOS 9.2 it’s moved to `__TEXT.__const`
Defeat KPP

- How does amfi check if debug flag is set or not?
- It calls a stub function of PE_i_can_has_debugger
- Stub function pointers are stored in __DATA.__got
- It’s easy to cheat amfi that debug is allowed
Defeat KPP

- KPP is triggered very randomly when the device is not busy
- Patch/Restore works well if the time window is small enough
Agenda

- iOS Security Overview
- Pangu 9 Overview
- Userland Exploits
- Kernel Exploits & Kernel Patching
- Persistent Code Signing Bypass
- Conclusion
Attack Surfaces for Persistent

- Attack dyld
  - Dynamic library
- Attack kernel
  - Main executable file
  - Dynamic linker
  - dyld_shared_cache
- Attack file parsing
  - Config file/javascript/…
Load dyld_shared_cache

- The dyld_shared_cache is never attacked before
- All processes share the same copy of dyld_shared_cache
  - It’s only loaded once
- dyld checks the shared cache state and tries to load it in mapSharedCache
  - _shared_region_check_np to check if cache is already mapped
- Open the cache and check cache header to make sure it’s good
- Generate slide for cache
- _shared_region_map_and_slide_np to actually map it
The Kernel Maps the Cache

```c
static int __attribute__((noinline)) _shared_region_check_np(uint64_t *start_address)
{
    if ( gLinkContext.sharedRegionMode == ImageLoader::kUseSharedRegion )
        return syscall(294, start_address);
    return -1;
}

static int __attribute__((noinline)) _shared_region_map_and_slide_np(int fd, uint32_t count, const struct shared_file_mapping_np *mappings, uint32_t slide, uint64_t* slide_start, uint32_t slide_size)
{  
    // register code signature blob for whole dyld cache
    if ( codeSignatureMappingIndex != -1 ) {  
        fsignatures_t siginfo;
        siginfo.fs_file_start = 0;  // cache always starts at beginning of file
        siginfo.fs_blob_start = (void*)mappings[codeSignatureMappingIndex].sfm_file_offset;
        siginfo.fs_blob_size = mappings[codeSignatureMappingIndex].sfm_size;
        int result = fcntl(fd, F_ADDFILESIG, &siginfo);
        // <rdar://problem/12891874> don't warn in chrooted case because mapping syscall is about to fail too
        if ( (result == -1) && gLinkContext.verboseMapping )
            dyld::log("dyld: code signature registration for shared cache failed with errno=%d\n", errno);
    }
    if ( gLinkContext.sharedRegionMode == ImageLoader::kUseSharedRegion ) {
        return syscall(438, fd, count, mappings, slide, slideInfo, slideInfoSize);
    }
}
```

- 294  AUE_NULL  ALL { int shared_region_check_np(uint64_t *start_address) NO_SYSCALL_STUB; }  
- 438  AUE_NULL  ALL { int shared_region_map_and_slide_np(int fd, uint32_t count, const struct shared_file_mapping_np *mappings, uint32_t slide, uint64_t* slide_start, uint32_t slide_size) NO_SYSCALL_STUB; }
Structure of dyld_shared_cache

```c
struct dyld_cache_header {
    char        magic[16]; // e.g. "dyld_v0 i386"
    uint32_t    mappingOffset; // file offset to first dyld_cache_mapping_info
    uint32_t    mappingCount;  // number of dyld_cache_mapping_info entries
    uint32_t    imagesOffset;  // file offset to first dyld_cache_image_info
    uint32_t    imagesCount;   // number of dyld_cache_image_info entries
    uint64_t    dyldBaseAddress;  // base address of dyld when cache was built
    uint64_t    codeSignatureOffset; // file offset of code signature blob
    uint64_t    codeSignatureSize; // size of code signature blob (zero means to end of file)
    uint64_t    slideInfoOffset; // file offset of kernel slid info
    uint64_t    slideInfoSize;  // size of kernel slid info
    uint64_t    localSymbolsOffset; // file offset of where local symbols are stored
    uint64_t    localSymbolsSize; // size of local symbols information
    uint8_t     uuid[16]; // unique value for each shared cache file
    uint64_t    cacheType; // 1 for development, 0 for optimized
};

struct dyld_cache_mapping_info {
    uint64_t    address;
    uint64_t    size;
    uint64_t    fileOffset;
    uint32_t    maxProt;
    uint32_t    initProt;
};

struct dyld_cache_image_info {
    uint64_t    address;
    uint64_t    modTime;
    uint64_t    inode;
    uint32_t    pathFileOffset;
    uint32_t    pad;
};
```
Structure of dyld_shared_cache

- dyld_cache_mapping_info stores all mapping informations at header->mappingOffset
  - From file offset to virtual address
- dyld_cache_image_info stores all dylibs and frameworks information at header->imagesOffset
  - address indicates the mach-o header of the dylib
  - pathFileOffset indicates the full path of the dylib
- The whole cache file has a single signature blob
  - codeSignatureOffset / codeSignatureSize
- Jtool(http://www.newosxbook.com/tools/jtool.html) helps to decode the header
shared_region_map_and_slide_np

- shared_region_copyin_mappings
  - Copyin all dyld_cache_mapping_info
- _shared_region_map_and_slide
  - Make sure it’s on root filesystem and owned by root
- vm_shared_region_map_file
  - Maps the file into memory according to dyld_cache_mapping_info
  - Record the 1st mapping and take it’s address as base address of cache
The Vulnerability

- There is no explicit SHA1 check of the cache header
- Read only memory with file offsets out of code signature range would not be killed
- Possible to use a fake header and control the mappings
Abuse AMFID

- Now we could control the mapping of cache
- We still can not touch r-x memory
- But we could manipulate r-- / rw- memory
- libmis.dylib exports _MISValidateSignature
- Change two bytes in export table to points _MISValidateSignature to return 0
- Code signing is bypassed!
Conclusion

- The battle between jailbreaks and Apple makes iOS better, and more secure
- IPC and kernel vulnerabilities exploitable by container apps impose a huge threat to iOS security
Q&A