The Userland Exploits of Pangu 8

@PanguTeam

Outline

- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Team ID Validation by Teasing the Trust-Cache
- Bypass Code Signing Validation by Segment Overlapping
- Sandbox Escape
- Conclusion

Pangu Team

- Security research team in China
- Focused on iOS security for more than 3 years
- Release two untether jailbreaks in half a year
  - 2014.6 - Pangu Axe for iOS 7.1.x
  - 2014.10 - Xuanyuan Sword for iOS 8-8.1

Pangu Team

- Xiaobo Chen (@dm557)
- Hao Xu (@windknown)
- Tielei Wang (@INT80_pangu)
- @ogc557
- @tb557
- @zengbanxian
- Siglos (@0x557)

Outline

- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Team ID Validation by Teasing the Trust-Cache
- Bypass Code Signing Validation by Segment Overlapping
- Sandbox Escape
- Conclusion

Team ID

- Check the entitlements of binary built by latest Xcode
  - com.apple.developer.team-identifier

```
<plist version="1.0">
<dict>
        <key>application-identifier</key>
        <string>U46NZDWC3Y.com.iflytek.ringdiyclient</string>
        <key>aps-environment</key>
        <string>development</string>
        <key>com.apple.developer.team-identifier</key>
        <string>U46NZDWC3Y</string>
        <key>get-task-allow</key>
        <true/>
        <key>keychain-access-groups</key>
        <array>
                <string>U46NZDWC3Y.com.iflytek.ringdiyclient</string>
        </array>
</dict>
</plist>
```

Data Protection

- Data protection class
  - A - NSFileProtectionComplete
  - B - NSFileProtectionCompleteUnlessOpen
  - C - NSFileProtectionCompleteUntilFirstUserAuthentication
  - D - NSFileProtectionNone

Data Protection

- Lots of files in “/var” are protected with
  - Class C - NSFileProtectionCompleteUntilFirstUserAuthentication
  - Even root cannot access those files if a device is never unlocked

- Create a file in “/var/mobile/Media” and print the attributes

```
NSFileCreationDate = "2014-11-04 14:11:24 +0000";
NSFileExtensionHidden = 0;
NSFileGroupOwnerAccountID = 501;
NSFileGroupOwnerAccountName = mobile;
NSFileModificationDate = "2014-11-04 14:11:24 +0000";
NSFileOwnerAccountID = 0;
NSFileOwnerAccountName = root;
NSFilePosixPermissions = 493;
NSFileProtectionKey = NSFileProtectionCompleteUntilFirstUserAuthentication;
NSFileReferenceCount = 1;
NSFileSize = 576;
NSFileSystemFileNumber = 33495;
NSFileSystemNumber = 16777218;
NSFileType = NSFileTypeRegular;
```

Data Protection

- Apple adds a special flag for folders
  - fcntl with F_GETPROTECTIONCLASS flag to get the protection class
  - 0 for “/var/mobile/Media”

```
/*
 * dir_none forces new items created in the directory to pick up the mount point default
 * protection level. it is only allowed for directories.
 */
#define PROTECTION_CLASS_DIR_NONE 0

#define PROTECTION_CLASS_A 1
#define PROTECTION_CLASS_B 2
#define PROTECTION_CLASS_C 3
#define PROTECTION_CLASS_D 4
#define PROTECTION_CLASS_E 5
#define PROTECTION_CLASS_F 6
```

Data Protection

- It is possible to change the protection class of folder to turn off the default protection

- fcntl with F_SETPROTECTIONCLASS to set protection class = 4 which is NSFileProtectionNone

Launchd

- Move core code from launchctl to launchd
  - Kill arguments normally used by jailbreak
    - “launchctl load -D all” no longer work
- Strict loading process
  - Load all plist files from xpcd_cache.dylib
  - Assert plist files also exist in /System/Library/LaunchDaemons
  - If you want to load a service from /System/Library/LaunchDaemons, the plist file must exist in xpcd_cache

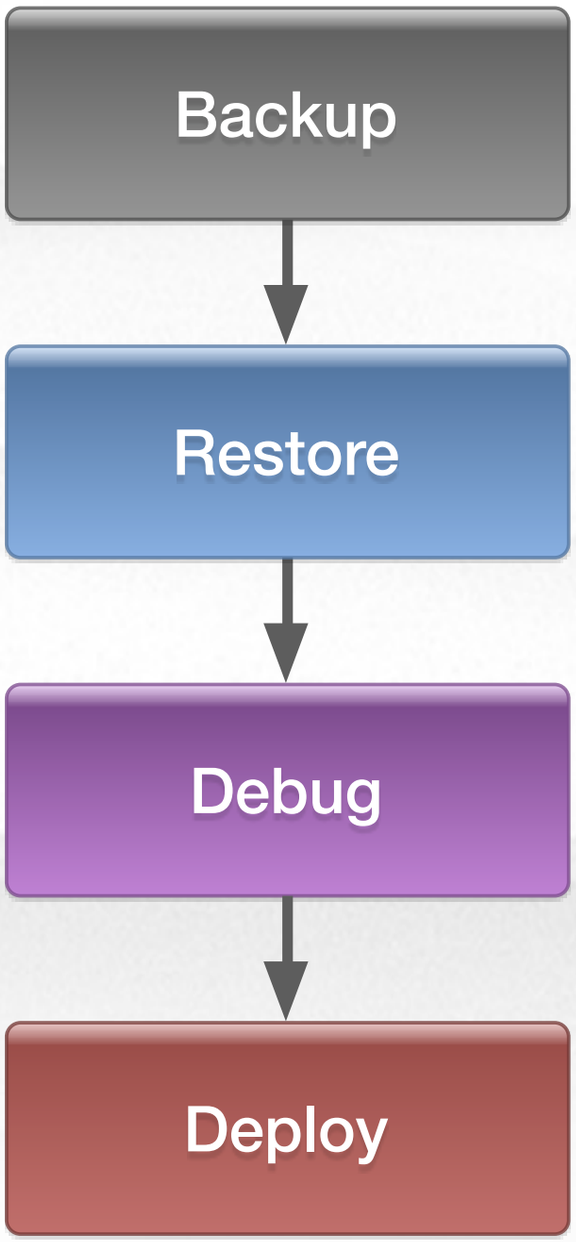
Launchd

- Weakness
  - Other arguments still work
    - “launchctl load paths”
  - Putting your plist files in /Library/LaunchDaemons seems no difference

Outline

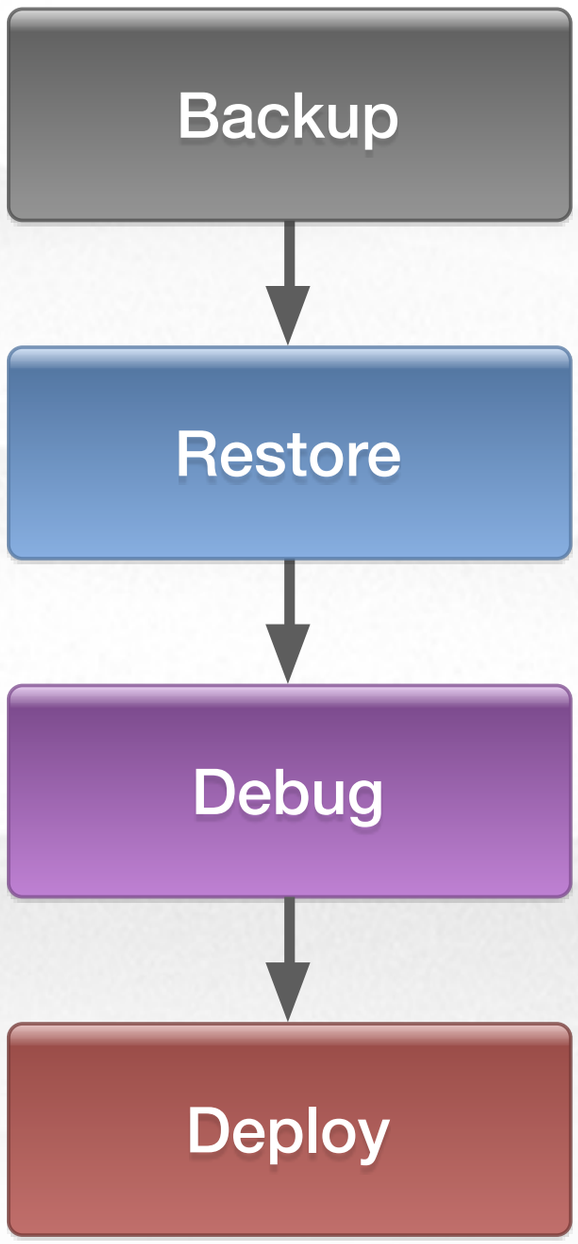
- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Code Signing Validation by Segment Overlapping
- Bypass Team ID Validation by Teasing the Trust-Cache
- Sandbox Escape
- Conclusion

Tethered jailbreak



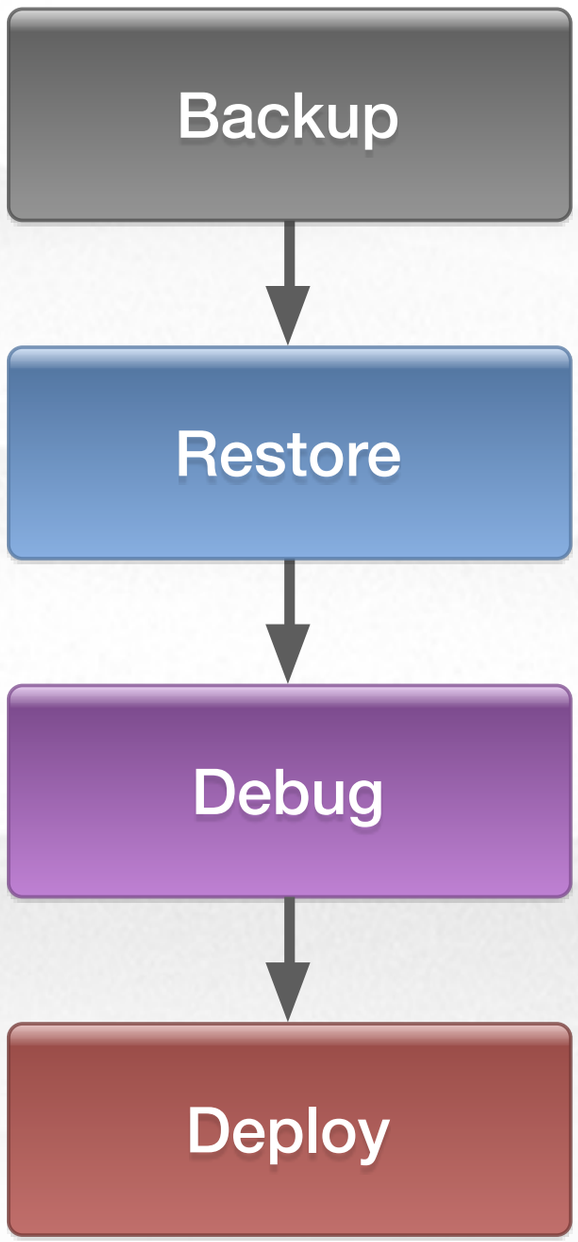
- Get a backup of iOS device

Tethered jailbreak



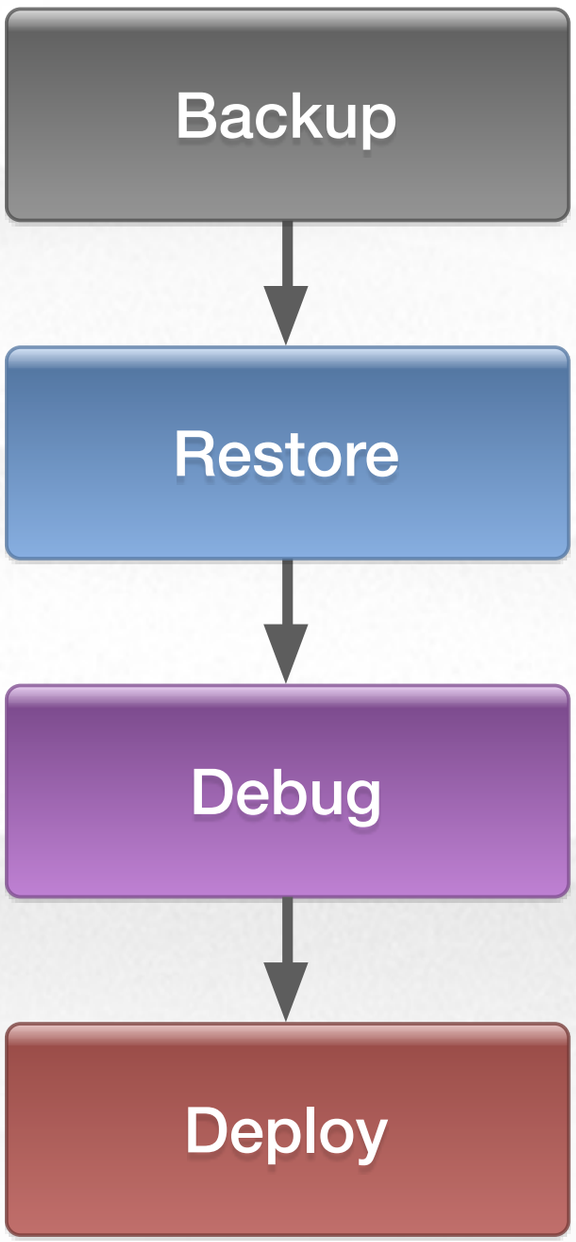
- Inject an expired enterprise license
- Turn off network connection
- Inject an app containing a dylib signed by the enterprise license

Tethered jailbreak



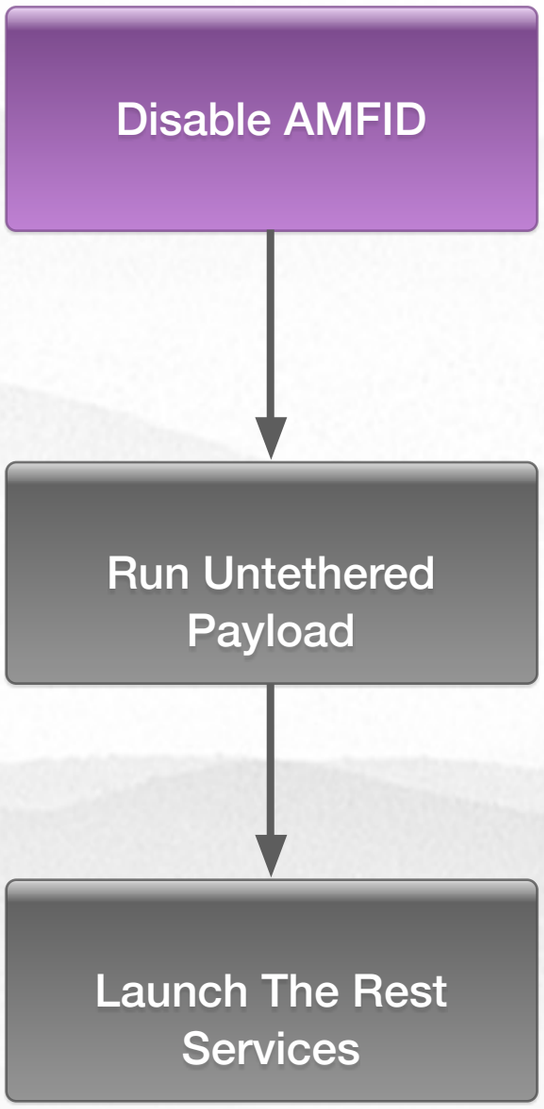
- Mount the developer disk image
- Instruct debugserver to debug neagent
- Force neagent to load the dylib by setting DYLD_INSERT_LIBRARIES

Tethered jailbreak



- Attack kernel through the dylib
- Disable sandbox
- Modify rootfs to place libmis.dylib and enable-dylibs-to-override-cache
- Adjust the boot sequence of launchd daemons

Untethered jailbreak



- Bypass Code Signing
- Bypass Team ID validation

- Exploit and patch the kernel

Outline

- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Team ID Validation by Teasing the Trust-Cache
- Bypass Code Signing Validation by Segment Overlapping
- Sandbox Escape
- Conclusion

Team Identifier Verification

- A new security mechanism introduced in iOS 8
- A team identifier (Team ID) is a 10-character alphanumeric string extracted from an Apple issued certificate.

```
<plist version="1.0">
        <dict>
                <key>keychain-access-groups</key>
                <array>
                        <string>249286WSCD.com.cisco.anyconnect.gui</string>
                </array>

                <key>com.apple.networking.vpn.configuration</key>
                <array>
                        <string>com.cisco.anyconnect.applevpn.plugin</string>
                </array>

                <key>com.apple.developer.team-identifier</key>
                <string>8NH9K37H72</string>

                <key>application-identifier</key>
                <string>249286WSCD.com.cisco.anyconnect.gui</string>

        </dict>
```

Team Identifier Verification

- A program may link against any platform library that ships with the system or any library with the same team identifier in its code signature as the main executable.
- System executables can only link against libraries that ship with the system itself.

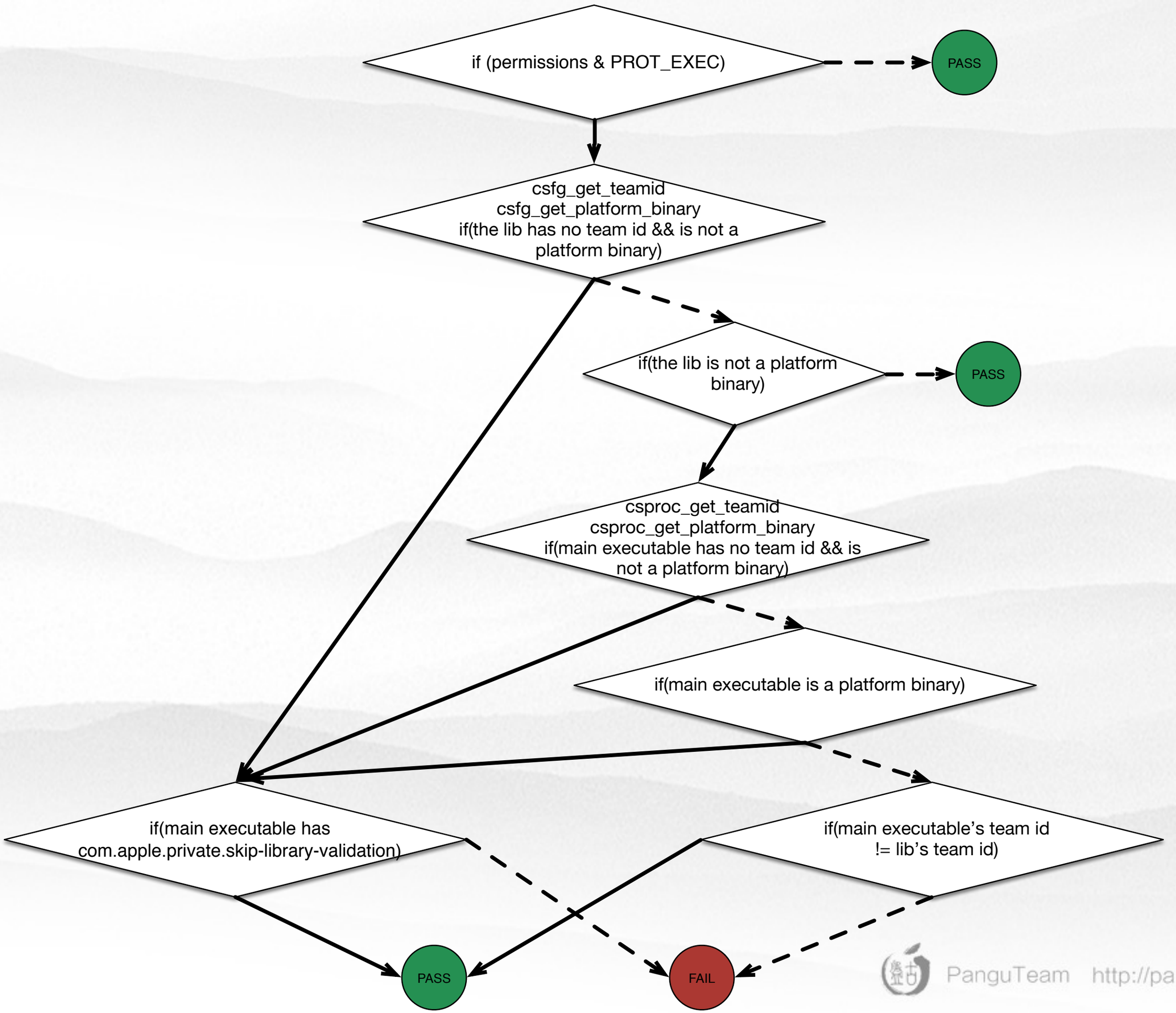
com.apple.driver.AppleMobileFileInteg…	00000034	C	AMFI: in mmap but not enforcing library validation\n
com.apple.driver.AppleMobileFileInteg…	0000004C	C	[deny-mmap] mapped file has no team identifier and is not a platform binary
com.apple.driver.AppleMobileFileInteg…	00000048	C	[deny-mmap] process has no team identifier and is not a platform binary
com.apple.driver.AppleMobileFileInteg…	00000041	C	[deny-mmap] process is a platform binary, but mapped file is not
com.apple.driver.AppleMobileFileInteg…	00000041	C	[deny-mmap] mapped file does not have a matching team identifier
com.apple.driver.AppleMobileFileInteg…	0000001F	C	AMFI: failed to get file path\n
com.apple.driver.AppleMobileFileInteg…	0000002B	C	[deny-mmap] process has team identifier %s
com.apple.driver.AppleMobileFileInteg…	0000002F	C	[deny-mmap] mapped file has team identifier %s

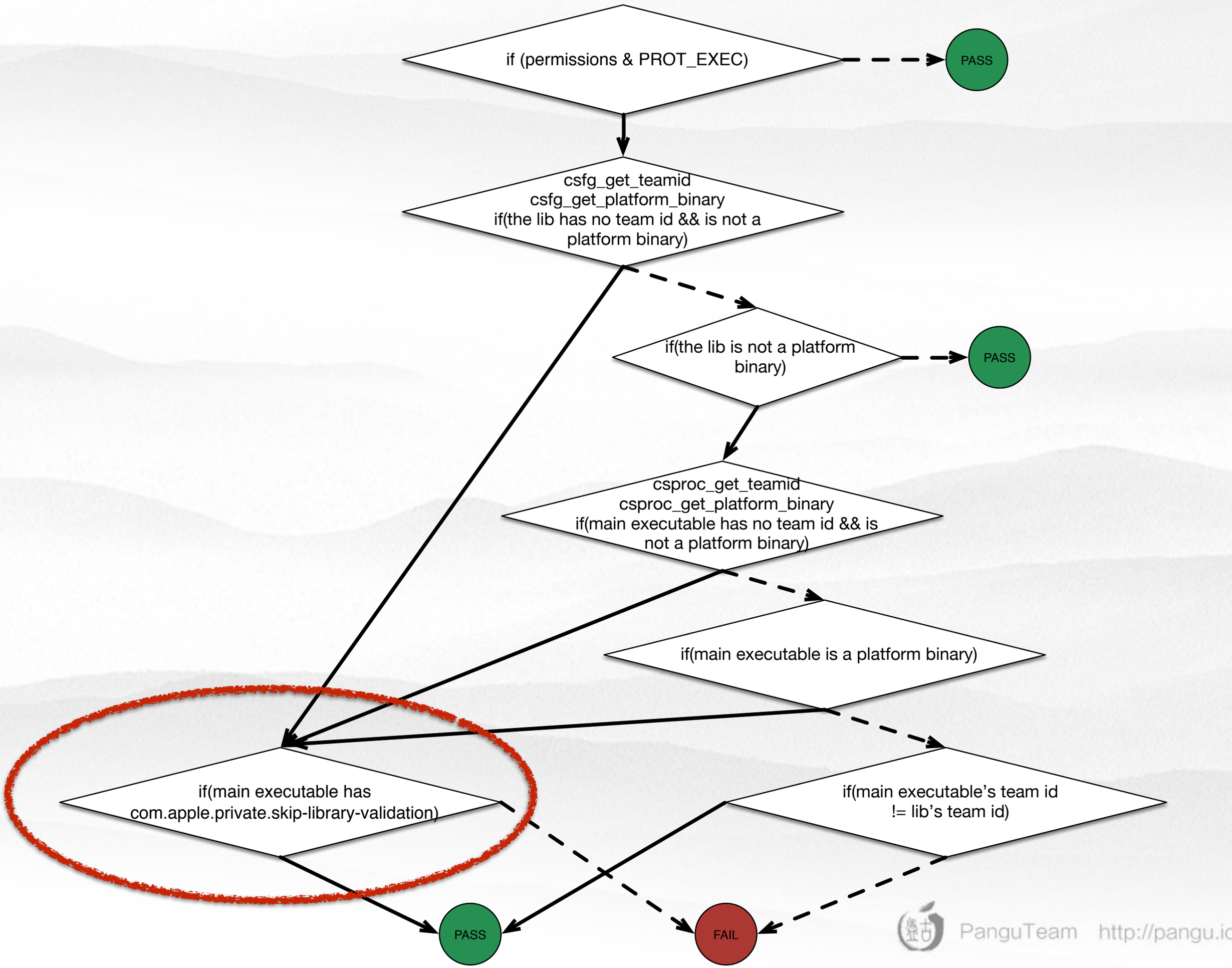
Troubles for jailbreak

- Code signing bypass
  - Method: force dyld to load a fake libmis.dylib
    - evasi0n, evasi0n 7, pangu 7
    - Challenge: the fake libmis.dylib must also pass the TeamID validation
- Sandbox escape
  - Method: Inject a dynamic library signed by a developer license into system processes, e.g., setting DYLD_INSERT_LIBRARIES
  - Challenge: the injected library has to pass the TeamID validation

Team ID verification Implementation

- AppleMobileFileIntegrity hooks the mmap function
- When a file is mapped into memory:
  - csfg_get_platform_binary
  - csfg_get_teamid
  - csproc_get_platform_binary
  - csproc_get_teamid





Who has the com.apple.private.skip-library-validation

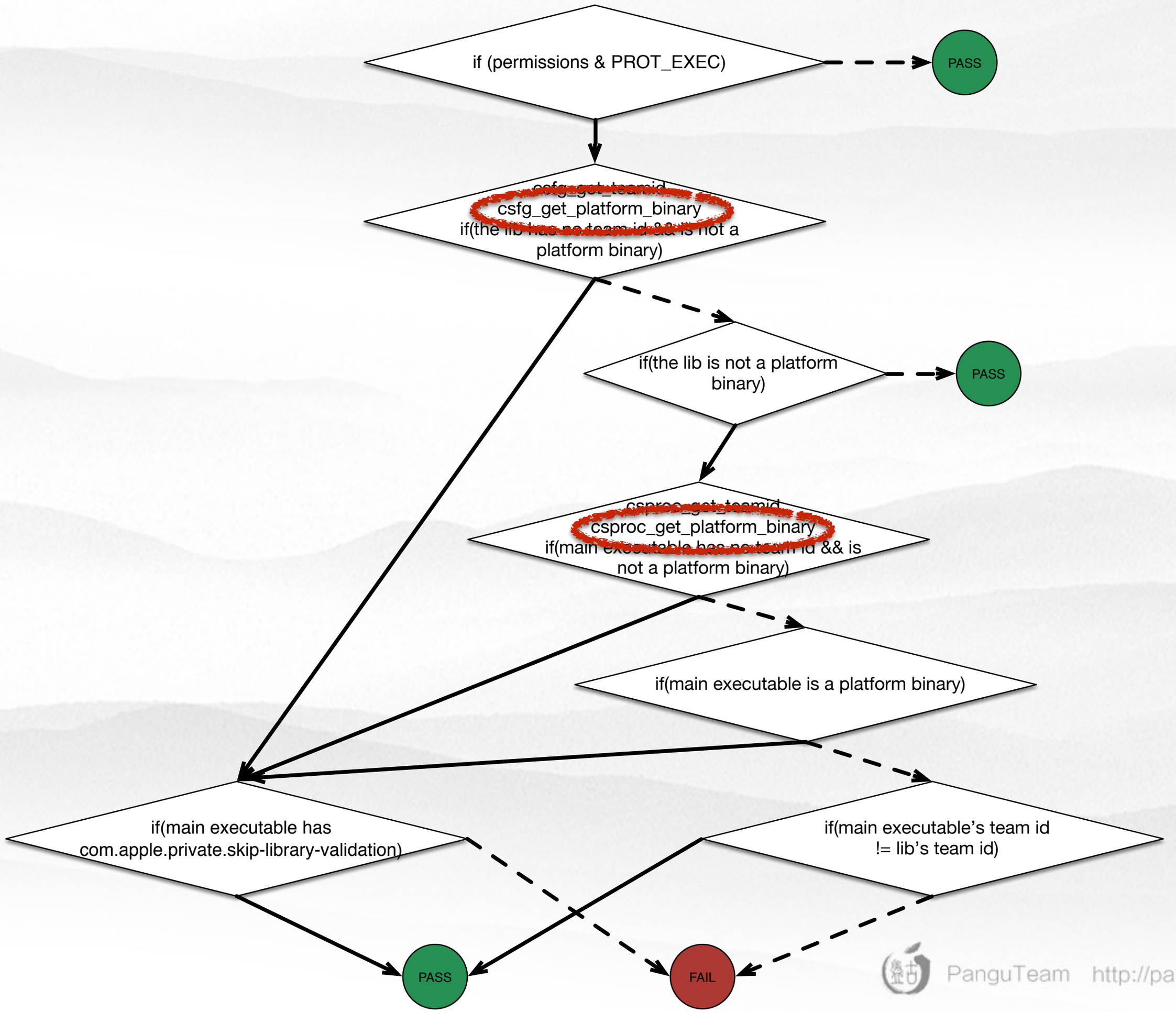
Good News: neagent has the entitlement

```
<plist version="1.0">
<dict>
        <key>com.apple.private.MobileGestalt.AllowedProtectedKeys</key>
        <array>
                <string>UniqueDeviceID</string>
        </array>
        <key>com.apple.private.neagent</key>
        <true/>
        <key>com.apple.private.necp.match</key>
        <true/>
        <key>com.apple.private.skip-library-validation</key>
        <true/>
        <key>keychain-access-groups</key>
        <array>
                <string>com.apple.identities</string>
                <string>apple</string>
                <string>com.apple.certificates</string>
        </array>
</dict>
</plist>
```

Bad News: neagent is the only one with the entitlement

Recall: Troubles for jailbreak

- Code signing bypass
  - Method: force dyld to load a fake libmis.dylib
  - Challenge: the fake libmis.dylib must also pass the TeamID validation
    - **Unsolved**
- Sandbox escape
  - Method: Inject a dynamic library signed by a developer license into system processes, e.g., setting DYLD_INSERT_LIBRARIES
  - Challenge: the injected library has to pass the TeamID validation
  - **Solved: inject the library to neagent**



How does iOS confirm a platform binary?

```
EXPORT _csproc_get_platform_binary
_csproc_get_platform_binary
PUSH            {R7,LR}
CMP             R0, #0
MOV             R7, SP
ITT NE
LDRNE.W         R1, [R0,#0x158]
CMPNE           R1, #0
BEQ             returnBranch
```

```
LDRD.W          R2, R3, [R0,#0x15C]
MOV             R0, R1
MOV.W           R1, #0xFFFFFFFF
BL              _ubc_cs_blob_get
CMP             R0, #0
ITT NE
LDRNE           R0, [R0,#0x50]
POPNE           {R7,PC}
```

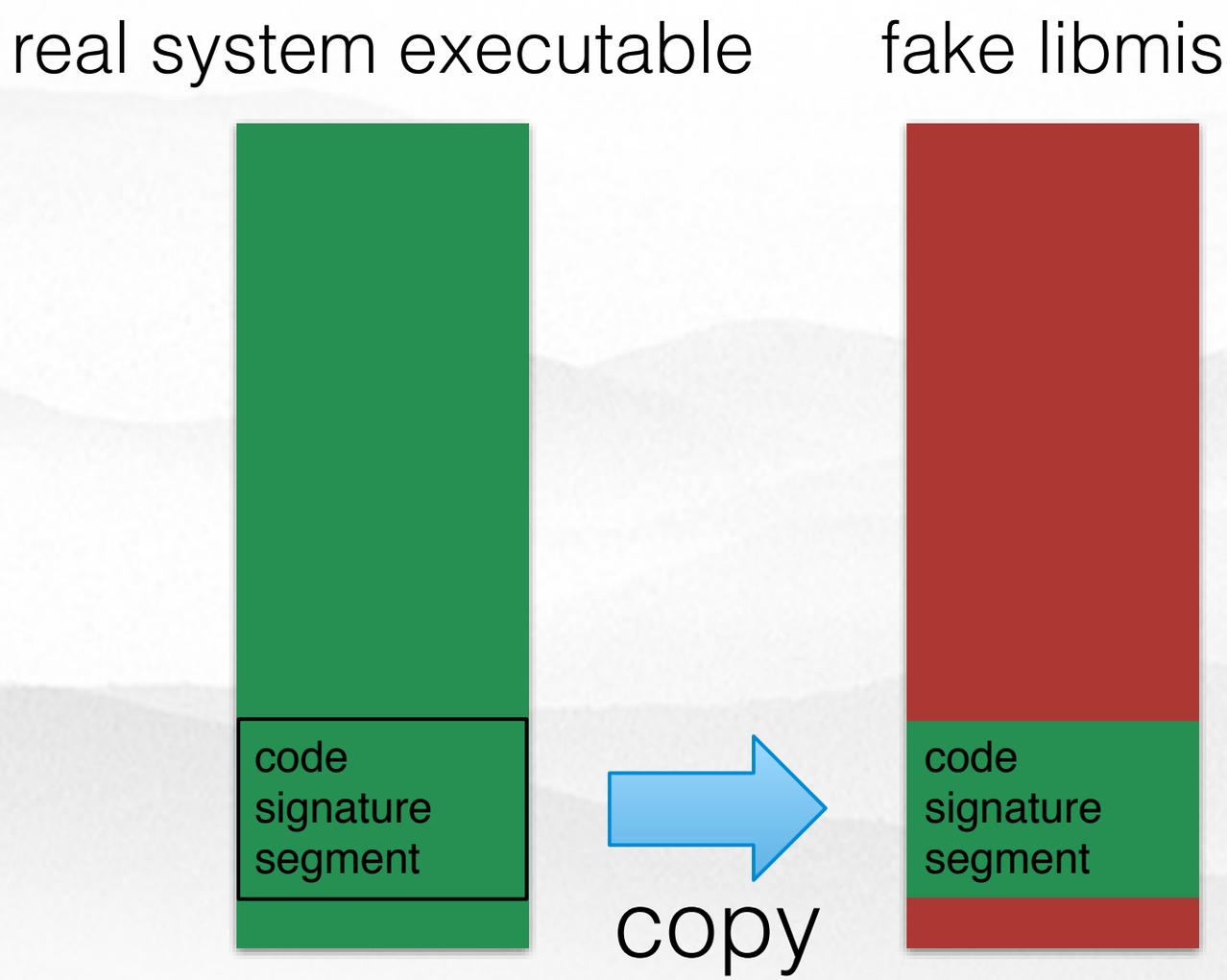
```
returnBranch
MOVS            R0, #0
POP             {R7,PC}
; End of function _csproc_get_platform_binary
```

```
struct cs_blob {
        struct cs_blob  *csb_next;
        cpu_type_t      csb_cpu_type;
        unsigned int    csb_flags;
        off_t           csb_base_offset;        /*
        off_t           csb_start_offset;       /*
        off_t           csb_end_offset;         /*
        ipc_port_t      csb_mem_handle;
        vm_size_t       csb_mem_size;
        vm_offset_t     csb_mem_offset;
        vm_address_t    csb_mem_kaddr;
        unsigned char   csb_sha1[SHA1_RESULTLEN];
        unsigned int    csb_sigpup;
        const char      *csb_teamid;
        unsigned int    csb_platform_binary;
};
```

How does iOS confirm a platform binary?

- Trust Cache
  - The kernel records the hash values of system executables
  - Rather than storing the hash value of the whole file, the trust cache only stores the sha1 value of the CS_CodeDirectory structure of the code signature segment in a system executable

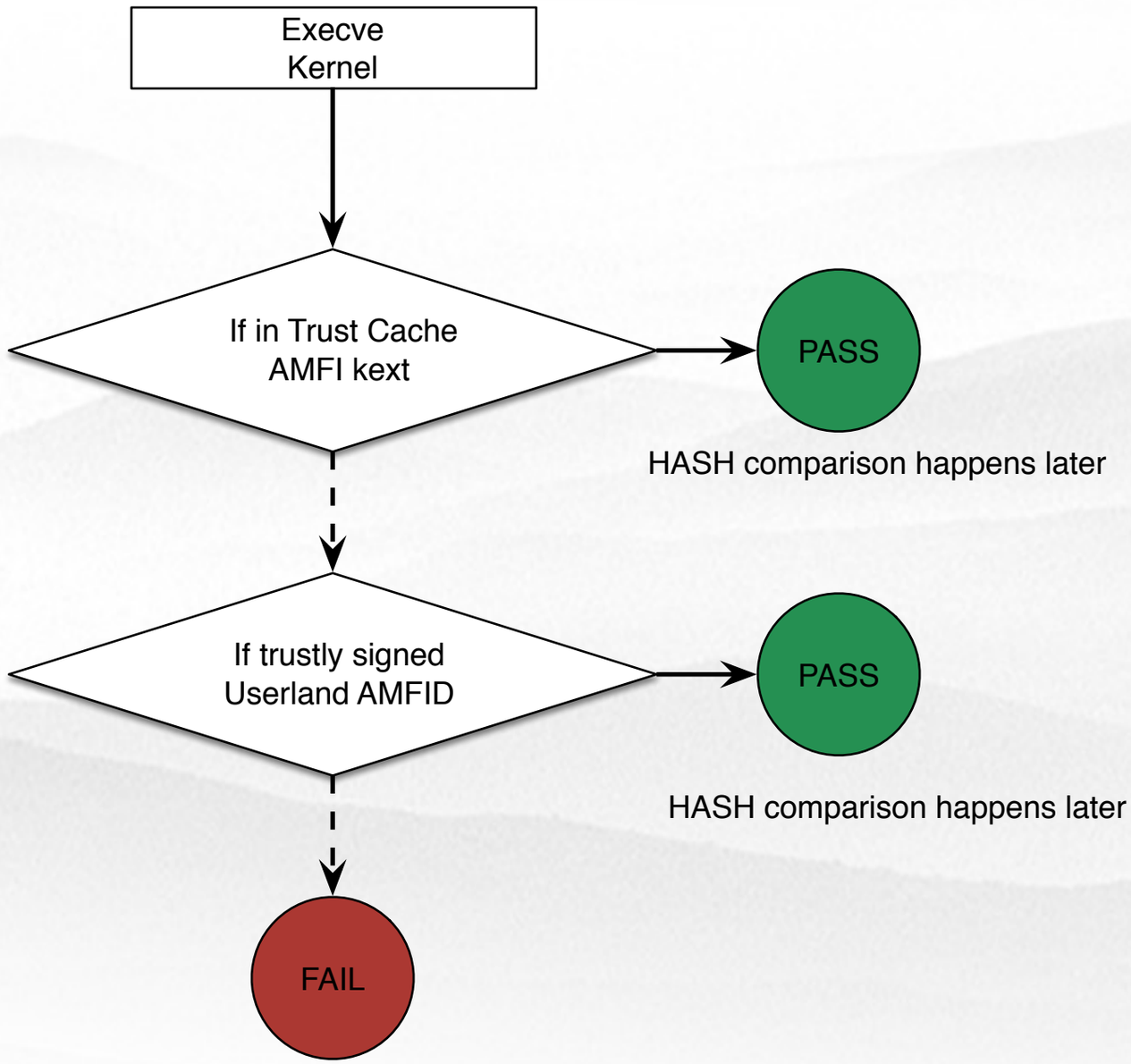
Fake libmis with a “correct” code signature segment



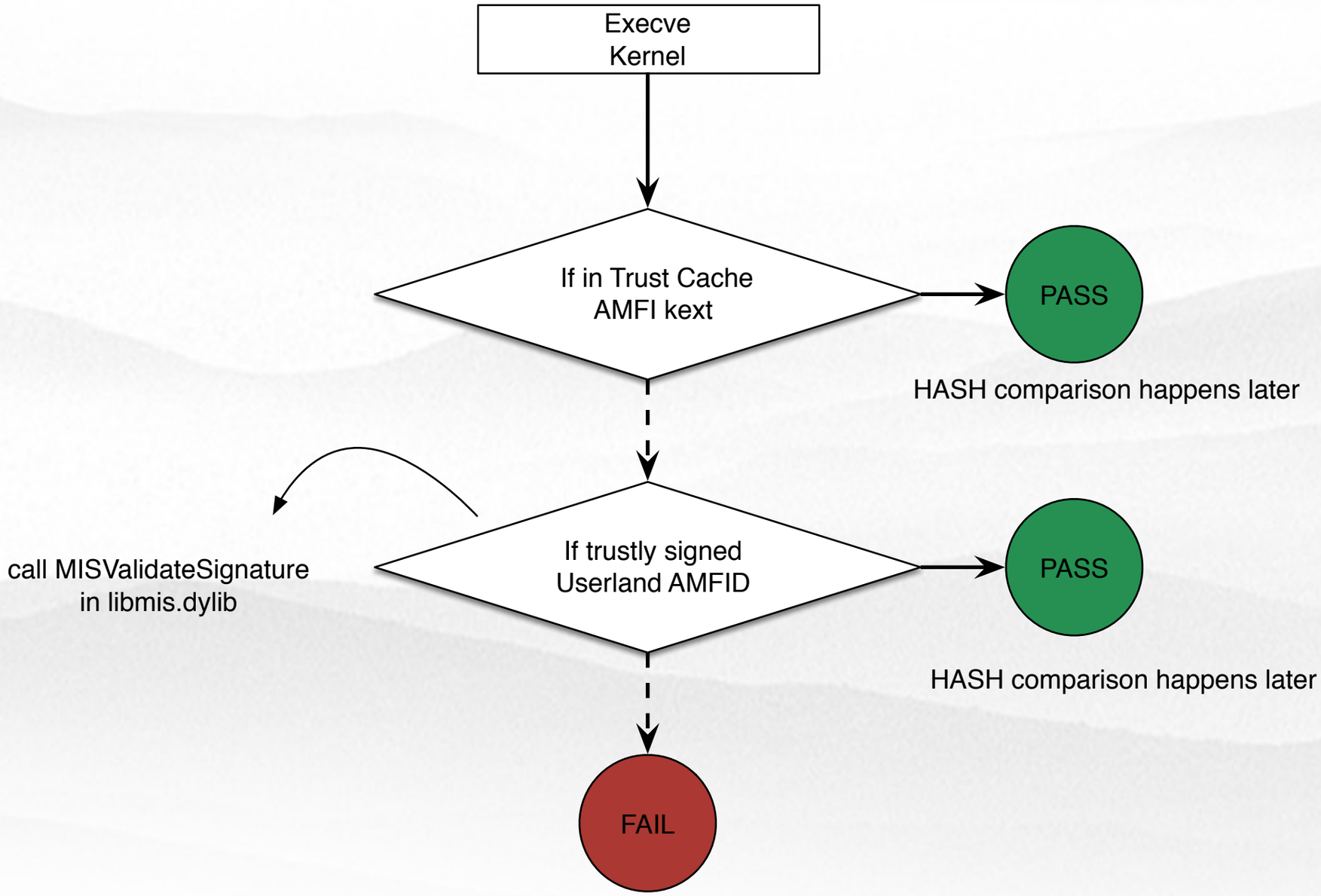
Outline

- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Team ID Validation by Teasing the Trust-Cache
- Bypass Code Signing Validation by Segment Overlapping
- Sandbox Escape
- Conclusion

Code Signing Workflow



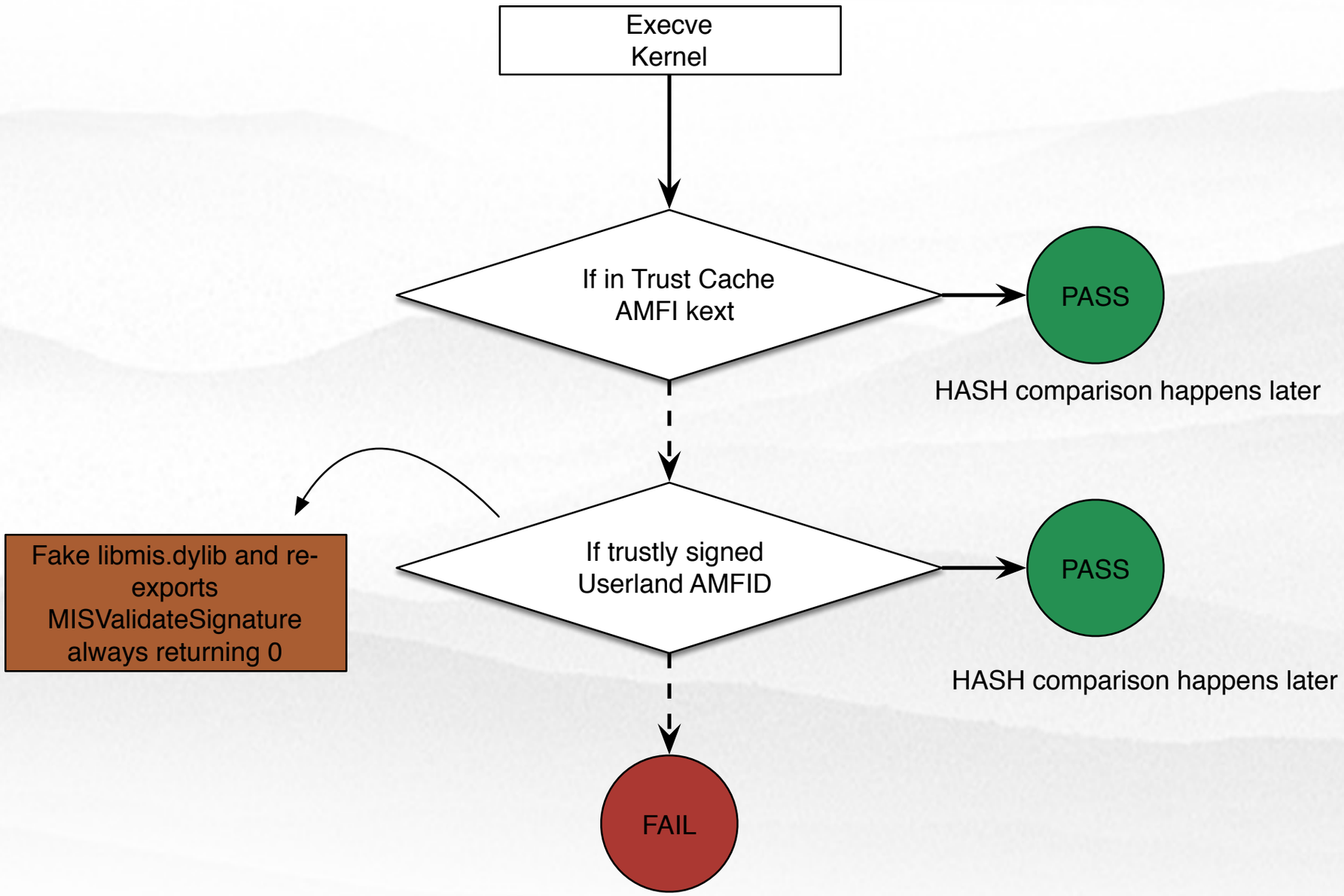
Code Signing Workflow



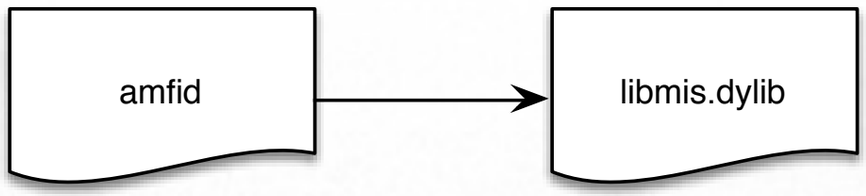
High Level Idea

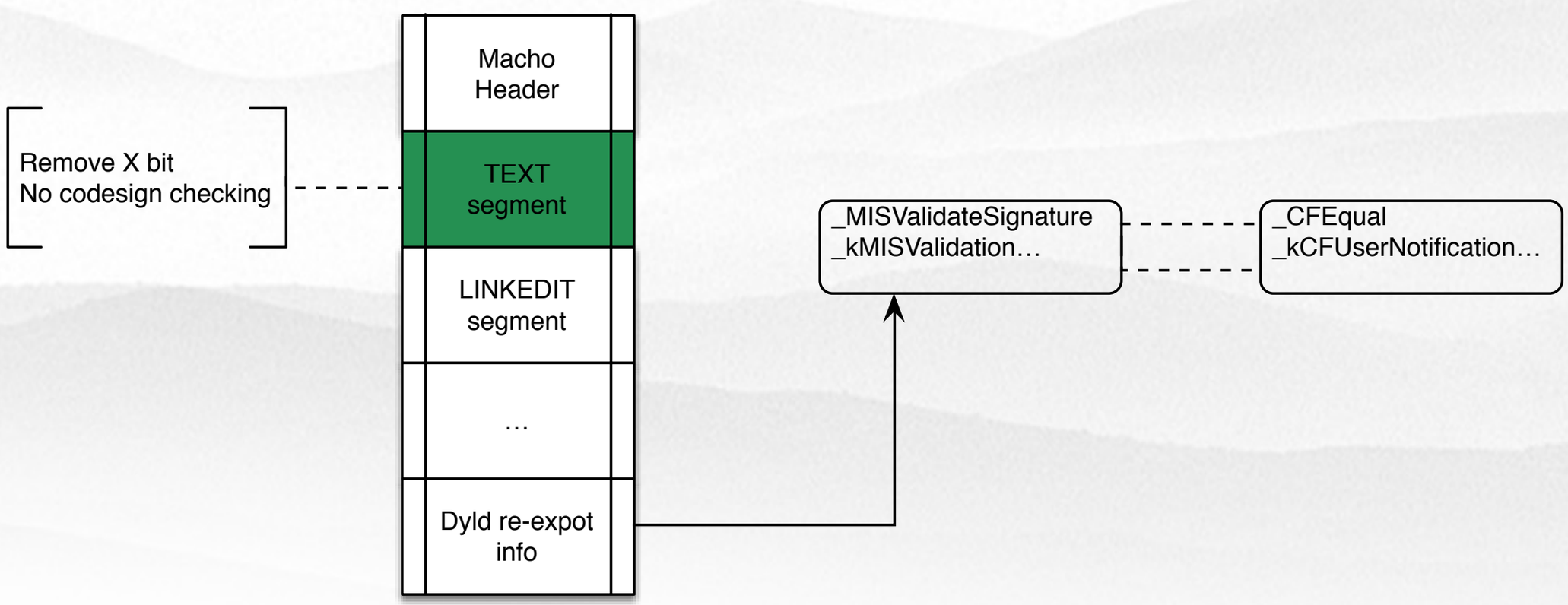
- First proposed by evad3rs since evasi0n 6
  - Use a simple dylib with no executable pages to replace libmis.dylib
  - The simple dylib itself does not trigger code signing checks at all, but it can interpose critical APIs responsible for the code signing enforcement

Code Signing Bypass



How to construct the dylib





Segment Overlapping Attack in evasi0n 6

Mach O File in Disk

Loading into Memory

Memory

TEXT Segment A

R.-.X
VMAddr: 0
VMSize: 4KB

TEXT Segment A

R.-.X

TEXT Segment B

R.-.-
VMAddr: 0
VMSize: 4KB

Segment Overlapping Attack in evasi0n 6

Mach O File in Disk

TEXT Segment A

R.-.X
VMAddr: 0
VMSize: 4KB

TEXT Segment B

R.-.-
VMAddr: 0
VMSize: 4KB

Loading into Memory

Memory

TEXT Segment A

TEXT Segment B

R.-.-

Review the fix

- It is really a challenge for us to find a new code sign exploit
- We reviewed the latest dyld source code carefully
  - How did Apple fix the segment overlapping problem?

```
// <rdar://problem/13145644> verify another segment does not over-map load commands
cmd = startCmds;
if ( context.codeSigningEnforced ) {
        for (uint32_t i = 0; i < cmd_count; ++i) {
                switch (cmd->cmd) {
                        case LC_SEGMENT_COMMAND:
                                if ( i != loadCommandSegmentIndex ) {
                                        segCmd = (struct macho_segment_command*)cmd;
                                        uintptr_t start = segCmd->vmaddr;
                                        uintptr_t end = segCmd->vmaddr + segCmd->vmsize;
                                        if ( ((start <= loadCommandSegmentVMStart) && (end > loadCommandSegmentVMStart))
                                           || ((start >= loadCommandSegmentVMStart) && (start < loadCommandSegmentVMEnd)) )
                                                dyld::throwf("malformed mach-o image: segment %s overlaps load commands", segCmd->segname);
                                }
                                break;
                }
                cmd = (const struct load_command*)(((char*)cmd)+cmd->cmdsize);
        }
}
```

Segment Overlapping's Revenge in Pangu 7

```
uintptr_t end = segCmd->vmaddr + segCmd->vmsize;
```

```
loadCommandSegmentVMEnd  = segCmd->vmaddr + segCmd->vmsize;
```

- **Integer overflow will cause the overlapping check to be bypassed**

- Finally we can still force two segments to overlap

Segment Overlapping's Revenge in Pangu 7

Mach O File in Disk

Loading into Memory

Memory

TEXT Segment A

R.-.X
VMAddr: 4KB
VMSize: **-4KB**

TEXT Segment A

R.-.X

TEXT Segment B

R.-.-
VMAddr: 4KB
VMSize: **-4KB**

Segment Overlapping's Revenge in Pangu 7

Mach O File in Disk

TEXT Segment A

R.-.X
VMAddr: 4KB
VMSize: **-4KB**

TEXT Segment B

R.-.-
VMAddr: 4KB
VMSize: **-4KB**

Loading into Memory

Memory

TEXT Segment A
TEXT Segment B

R.-.-

Apple’s fix in iOS 8

- To fix Pangu7’s codesign exploit, Apple adds more checks to the 1st R-X segment
  - vmsize can’t be negative
  - vmaddr + vmsize cannot overflow any more

```
loadCommandSegmentVMStart = segCmd->vmaddr;
loadCommandSegmentVMEnd   = segCmd->vmaddr + segCmd->vmsize;
if ( (intptr_t)(segCmd->vmsize) < 0)
    dyld::throwf("malformed mach-o image: segment load command %s size too large", segCmd->segname);
if ( loadCommandSegmentVMEnd < loadCommandSegmentVMStart )
    dyld::throwf("malformed mach-o image: segment load command %s wraps around address space", segCmd->segname);
```

The new problem in iOS 8

- The added checks do not apply to other segments!

```
for(unsigned int i=0, e=segmentCount(); i < e; ++i) {
    const uintptr_t segLow = segPreferredLoadAddress(i);
    const uintptr_t segHigh = dyld_page_round(segLow + segSize(i));
    if ( segLow < highAddr ) {
        if ( dyld_page_size > 4096 )
            dyld::throwf("can't map segments into 16KB pages");
        else
            dyld::throwf("overlapping segments");
    }
    if ( segLow < lowAddr )
        lowAddr = segLow;
    if ( segHigh > highAddr )
        highAddr = segHigh;
```

- No negative or overflow checking for other segments!

http://opensource.apple.com/source/dyld/dyld-353.2.1/src/ImageLoaderMachO.cpp

Segment Overlapping's Revenge in Pangu 8

- What did Pangu8 do
  - dyld will first allocate a memory range for the first segment base on its vmaddr
  - We can make the second segment to overlap the first one again by setting the second segment's vmaddr and vmsize

```
Load command 0
      cmd LC_SEGMENT
  cmdsize 56
  segname __FAKE_TEXT
   vmaddr 0x00000000
   vmsize 0x00040000
  fileoff 0
 filesize 262144
  maxprot 0x00000001
 initprot 0x00000005
   nsects 0
    flags 0x0
Load command 1
      cmd LC_SEGMENT
  cmdsize 56
  segname __TEXT
   vmaddr 0xffff000
   vmsize 0x00001000
 filesize 266240
  maxprot 0x00000001
 initprot 0x00000001
   nsects 0
    flags 0x0
```

Segment Overlapping's Revenge in Pangu 8

Mach O File in Disk

Loading into Memory

Memory

TEXT Segment A

R.-.X
VMAddr: 0KB
VMSize: **4KB**

TEXT Segment A

R.-.X

TEXT Segment B

R.-.-
VMAddr: **-4KB**
VMSize: **4KB**

Segment Overlapping's Revenge in Pangu 8

Mach O File in Disk

TEXT Segment A

R.-.X
VMAddr: 0KB
VMSize: **4KB**

TEXT Segment B

R.-.-
VMAddr: **-4KB**
VMSize: **4KB**

Loading into Memory

Memory

TEXT Segment B
TEXT Segment A

R.-.-

Segment Overlapping's Revenge in Pangu 8

- What did Pangu8 do
  - The dyld's debugging output while loading Pangu8's limbs.dylib

```
dyld: Mapping ./libmis.dylib (slice offset=16384)
        __FAKE_TEXT at 0x00129000->0x00168FFF with permissions r.x
            __TEXT at 0x00128000->0x00168FFF with permissions r..
        __LINKEDIT at 0x0016B000->0x0016B0BA with permissions r..
dyld: loaded: ./libmis.dylib
```

- We can still do the overlap segment attack!

Apple’s fix in iOS 8.1.1

- Apple added vmsize and filesize checks in ImageLoaderMachO::sniffLoadCommands

```
else if ( (_DWORD)a1 == 1 )
{
  LODWORD(a1) = *(_DWORD *)(v12 + 28);
  HIDWORD(a1) = *(_DWORD *)(v12 + 36);
  if ( HIDWORD(a1) > (unsigned int)a1 )
    dyld::throwf(
      (dyld *)"malformed mach-o image: segment load command %s filesize is larger than vmsize",
      (const char *)(v12 + 8),
      a5);
```

Hey Apple, do you really understand the issue?

Apple's ~~fix~~ in iOS 8.1.1

- The issue is about overlap in vmaddr

- Checks on vmsize/file size do not help at all

- We can still adjust vmsize in our codesign exploit and it is still working on iOS 8.1.1 - 8.1.2

Apple’s final fix in iOS 8.1.3

- Apple adds more checks for vm/file content overlapping

```
if ( v28 >= v23 && v28 < v26 && v30 > v28 )
{

  v37 = (dyld *)"malformed mach-o image: segment %s vm overlaps segment %s";
  goto LABEL_81;
}
v33 = *(_DWORD *)(v25 + 36) + v29;
if ( v29 <= v21 )
{
  v34 = v33 >= v21;
  v35 = v33 == v21;
  if ( v33 > v21 )
  {
    v34 = v24 >= v21;
    v35 = v24 == v21;
  }
  if ( !v35 & v34 )
    goto LABEL_100;
}
if ( v29 >= v21 && v29 < v24 && v33 > v29 )
{

  v37 = (dyld *)"malformed mach-o image: segment %s file content overlaps segment %s";

  dyld::throwf(v37, (const char *)(v19 + 8), v25 + 8);
}
```

- Bypassable?

Outline

- Introduction
- New Security Enhancements in iOS 8
- Pangu 8 Overview
- Bypass Team ID Validation by Teasing the Trust-Cache
- Bypass Code Signing Validation by Segment Overlapping
- Sandbox Escape
- Conclusion

Why we chose neagent

- Kernel exploits against IOHIDEventService require a loose sandboxed environment
- We have to bypass the Team ID verification at the first step
- debugserver + neagent is the perfect target

Forcing neagent to load our library

- Solution: leverage idevicedebug in the libimobiledevice package to communicate with debugserver in the iOS device

```
Usage: idevicedebug [OPTIONS] COMMAND
Interact with the debugserver service of a device.

 Where COMMAND is one of:
  run BUNDLEID [ARGS...]        run app with BUNDLEID and optional ARGS on device.

 The following OPTIONS are accepted:
  -e, --env NAME=VALUE  set environment variable NAME to VALUE
  -u, --udid UDID       target specific device by its 40-digit device UDID
  -d, --debug           enable communication debugging
  -h, --help            prints usage information
```

Apple’s fix in iOS 8.1.2

- Apple only allows debugserver to launch executables with debug-mode

```
allow process-fork
    (0)[25](debug-mode)

allow process-exec-interpreter
    (0)[29](debug-mode)
```

Conclusion

- Developing an untethered jailbreak requires a lot of effort
- Apple made similar mistakes again and again
- Next jailbreak?

Thanks

- Thank all of you
- Thanks Apple for bringing us such great devices
- Thanks the jailbreak community
    - special thanks goes to evad3rs, saurik and iH8sn0w
- Thanks for open source project libimobiledevice and Duilib

Q & A