GEORGIA TECH INFORMATION SECURITY CENTER
Safeguarding Digital Infomation Through Innovative Research and Education

Exploiting Unpatched iOS Vulnerabilities for Fun and Profit

Yeongjin Jang, Tielei Wang,
Byoungyoung Lee, and Billy Lau
Georgia Tech Information Security Center (GTISC)

Georgia Institute of Technology

Agenda

- iOS security overview
  - Why is rooting an iOS device hard?
- How were previous jailbreaks performed?
  - General steps
  - Steps in evasi0n7
- How was evasi0n7 patched?
  - Patch logs in iOS 7.1
  - Which steps were fixed?

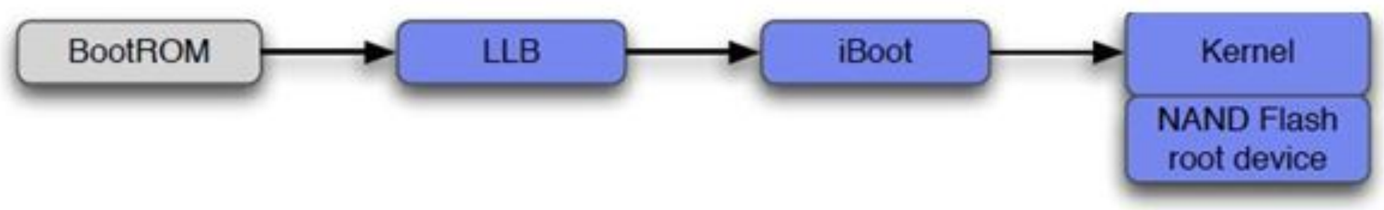
Agenda

- Analysis of patched/unpatched vulnerabilities
  – What steps need to be re-exploited?
- Discovery of new vulnerabilities to replace patched vulnerabilities
- Steps for Jailbreaking iOS 7.1.2

iOS Security Overview

- Why is rooting an iOS device hard?
  - Secure Boot Chain
  - Mandatory Code Signing
  - App Sandbox
  - Privilege Isolation

iOS Security – Secure Boot Chain

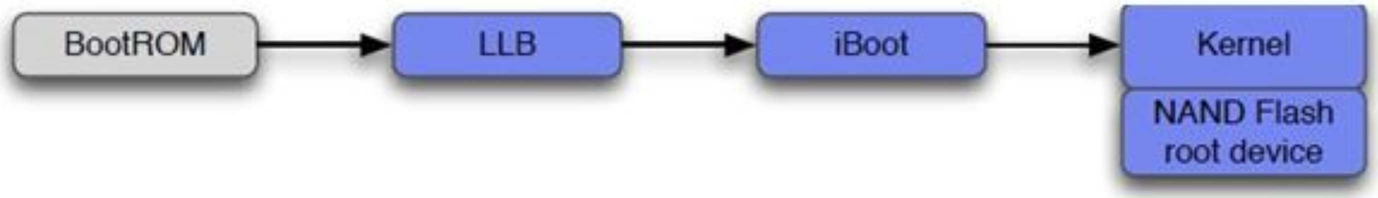


(Figure 1) (copied from Sogeti presentation)

- Chained code signing check
  - Verify RSA signatures on loading of each code

iOS Security – Secure Boot Chain

- Encrypted firmware
  - Encrypted with GID key of the device.
    - GID key is same among the same device
      - GID of iPhone 5 != GID of iPhone 4
  - Image is only decrypted on the device.
    - GID key is not designed to be leaked.
      - Before getting into kernel boot, GID key is disabled by iBoot.



(Figure 1) (copied from Sogeti presentation)

iOS Security – Mandatory Code Signing

- Code signing check
  - Enforced by kernel (AMFI), handled by a user-space daemon (amfid)
    - Kernel code is signed (cannot be modified).
    - Signature of all user-level code is checked by kernel.
  - Signing Entity
    - Apple App Store / Developer License / Enterprise License
    - For system binaries such as /bin/launchctl,
      - Their hash is whitelisted in kernel.

iOS Security – Mandatory Code Signing

- W+X protection
  - What will be happen if a program in iOS can write a code and jump into it?
    - A way of bypassing code signing check!

iOS Security – Mandatory Code Signing

- W+X protection
  - Disallows having both write and execute permissions on any single memory page
    - mmap() with PROT_WRITE & PROT_EXEC will fail
      - mprotect() also disallowed to change permission of a previously mapped page into an executable page.
    - Only allowed apps with dynamic-codesign Entitlement, e.g. MobileSafari, for utilizing Just-in-Time compilation
      - Google Chrome in iOS cannot use JIT, upto iOS 7.
        - » iOS 8 has new JIT-enabled WebView, as separated process.

iOS Security – App Sandbox

- All third party apps residing at /var/mobile/Applications/* will be contained by a built-in sandbox profile named *container*
  - Enforced by kernel.
- For some built-in binaries, the sandbox is initiated by invoking APIs in libsandbox.dylib.
  - /usr/libexec/afcd, etc.
- Running a third party app outside of the container will trigger the "`outside_of_container && !i_can_has_debugger`" exception
  - Non-whitelisted binary (all signed binaries) must be executed under the container.
- Refer to "The Apple Sandbox" talk in BH DC 2011

iOS Security – Privilege Isolation

- UID of Apps
  - – mobile (501) is used for regular apps
    - For all Developer, Enterprise, and App Store apps.
- A few daemons run as root
  - – syslogd, lockdownd.

Why is Rooting an iOS Device Hard?

- Extremely restricted environment in sandbox
  - Mandatory for user-written or App Store apps
- Unable to run unsigned code
  - One must bypass code signing checks to run attack code
- Privilege escalation is required
  - All apps are running as mobile (uid=501) user
- Cannot permanently modify kernel image
  - Integrity checking is enforced

General Methods for Jailbreaking

- Bypass code signing
- Escape the sandbox
- Privilege escalation to root
- Patch kernel to nullify security checks

General Methods for Jailbreaking

- Bypass code signing
  - Exploit vulnerabilities in dyld during loading of code.
    - evasi0n7, Pangu
  - Use R.O.P. or exploit the process with dynamic code signing.
    - MobileSafari

General Methods for Jailbreaking

- Escape the sandbox
  - Exploit an un-sandboxed process.
  - Exploit design flaw in sandbox implementation.
  - Override sandbox functions in libsandbox.dylib.
    - Run the sandboxed process without really invoking the sandbox functions.
  - For apps in the container, kernel patching is required.

General Methods for Jailbreaking

- Root Privilege Escalation
  - Exploit vulnerabilities in a root daemon.
    - CrashHouseKeeping, etc.

General Methods for Jailbreaking

- Patch the kernel
  - Disable code signing.
  - Disable kernel-enforced sandbox.
  - Enable RWX mapping.
  - Enable kernel debugging (task_for_pid 0).

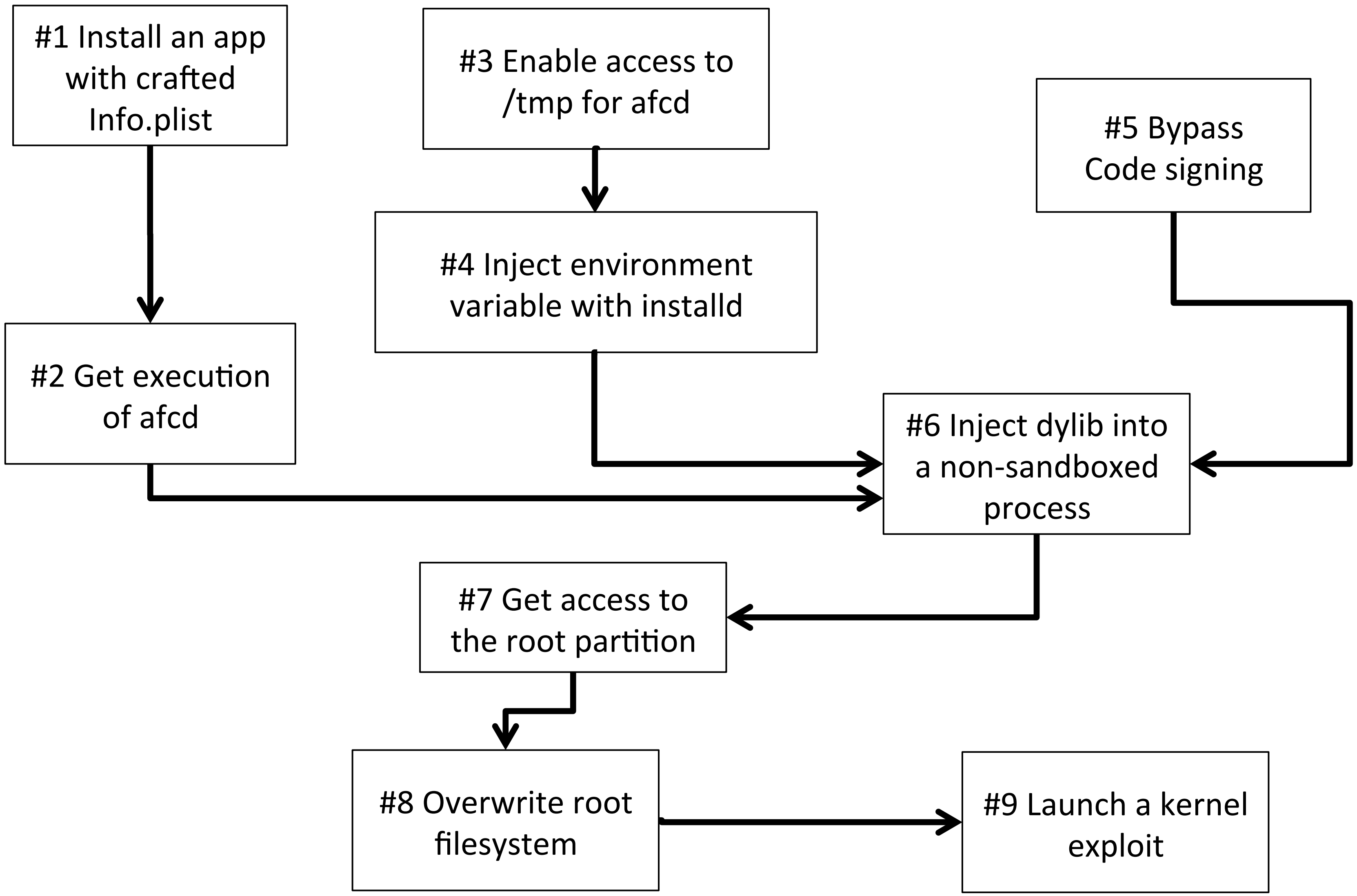
General Methods for Jailbreaking

- Apply Permanent Changes
  - Overwrite the root partition
    - Remount with read/write permission (ver < iOS 7), or use afcd (iOS 7.0.x).
  - Do not modify critical parts that are involved in the boot sequence
    - Chained integrity check could block boot process.

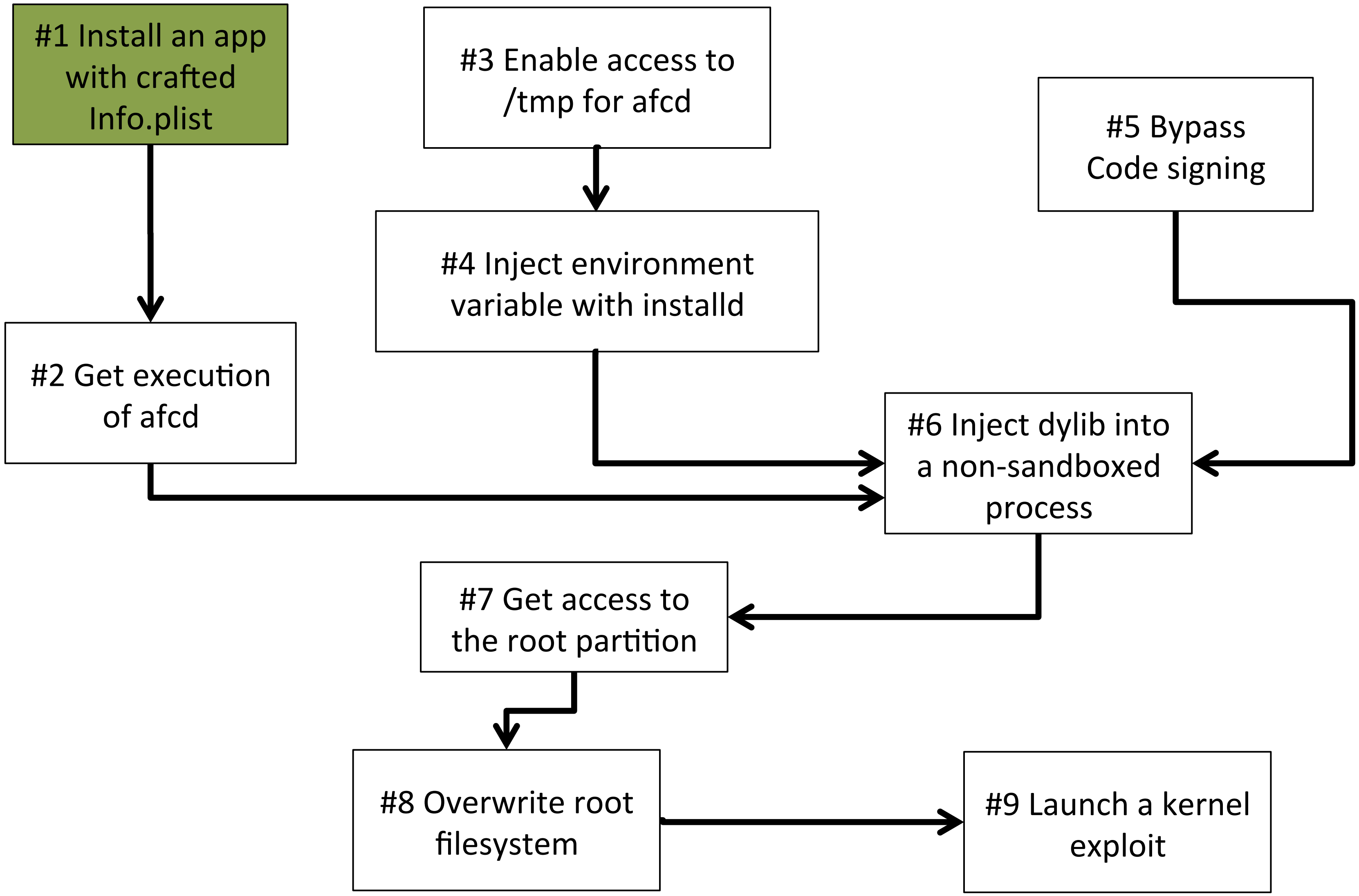
evasi0n7

- Exploited multiple vulnerabilities to bypass code signing checks, escape the sandbox, and overwrite the root partition.
- Exploited a kernel vulnerability to patch the kernel.
- Thanks to evad3rs for their jailbreak tool.
- Thanks to geohot for his detailed write-up.

evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n7 Workflow

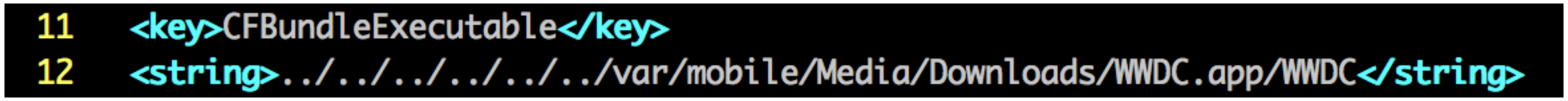

#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

App Installation

- The container
  - All third-party apps are installed into the container
    - /var/mobile/Applications/UUID-of-an-App/WWDC.app
  - If app executable resides inside of the container,
    - The App Sandbox will be applied from the kernel
      - Just compare the prefix for the directory
    - As a user-level process, there is no way of bypassing it without patching the kernel

evasi0n7 – Vulnerability #1

- Install an app with crafted Info.plist
  - Crafted Info.plist forces installd to install the app outside of the container
    - Using ../../../../../../ in CFBundleExecutable field



```
<key>CFBundleExecutable</key>
<string>../../../../../../var/mobile/Media/Downloads/WWDC.app/WWDC</string>
```

  - Prepare the original executable in that folder
    - In /var/mobile/Media/Downloads/* directory
    - Accessible through USB cable
  - Installation will succeed
    - Code signing check will use Downloads directory
    - Installd will do chmod +x on that executable file.

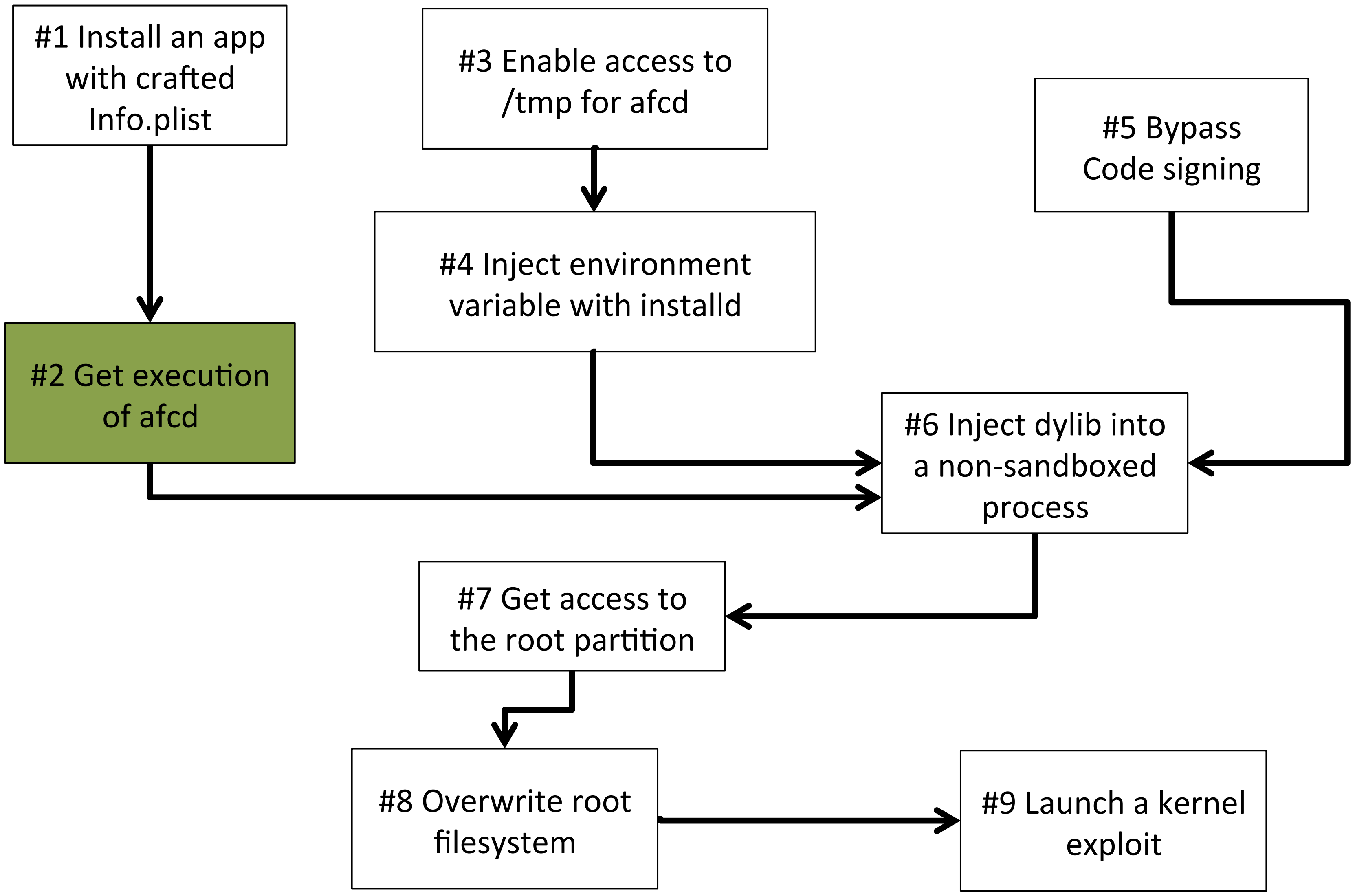
evasi0n7 – Vulnerability #1

- Install an app with crafted Info.plist
  - On launching the application,
    - /var/mobile/Media/Downloads/WWDC.app/WWDC will be executed.
    - The container sandbox is not applied…
      - Prefix is not matched with /var/mobile/Applications
    - However, there will be an exception

iOS Security – App Sandbox

- All third party apps residing at /var/mobile/Applications/* will be contained by a built-in sandbox profile named *container*
  - Enforced by kernel.
- For some built-in binaries, the sandbox is initiated by invoking APIs in libsandbox.dylib.
  - /usr/libexec/afcd, etc.
- **Running a third party app outside of the container will trigger the “`outside_of_container && !i_can_has_debugger`” exception**
  - Non-whitelisted binary (all signed binaries) must be executed under the container.
- Refer to “The Apple Sandbox” talk in BH DC 2011

evasi0n7 Workflow

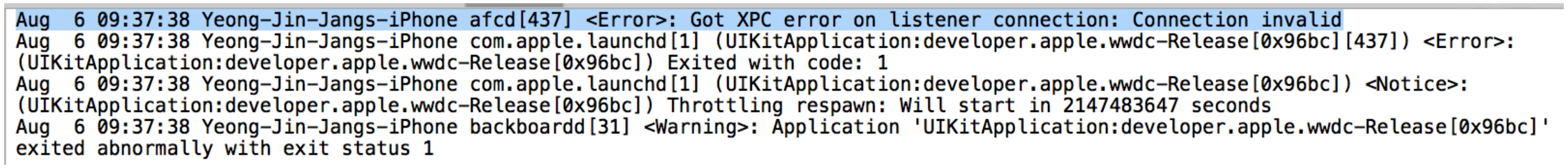

#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

Bypass Kernel-enforced Sandbox

- Now we can execute a file outside of the container
  - But we cannot run 3rd party apps
  - Then, launch system binary whose hash is whitelisted in the kernel!
    - /usr/libexec/afcd
    - /bin/launchctl
    - etc.

evasi0n7 – Vulnerability #2

- Gain execution of afcd (Apple File Conduit)
  - Since afcd has access to the /var/mobile/Media/ Downloads/ directory, a PC can ask afcd to change the content of an app executable to a hashbang
    - #!/usr/libexec/afcd –S –d / -p 8888
- Clicking the app icon will trigger the execution of "afcd" with forged arguments
  - New instance of afcd, different with original one.

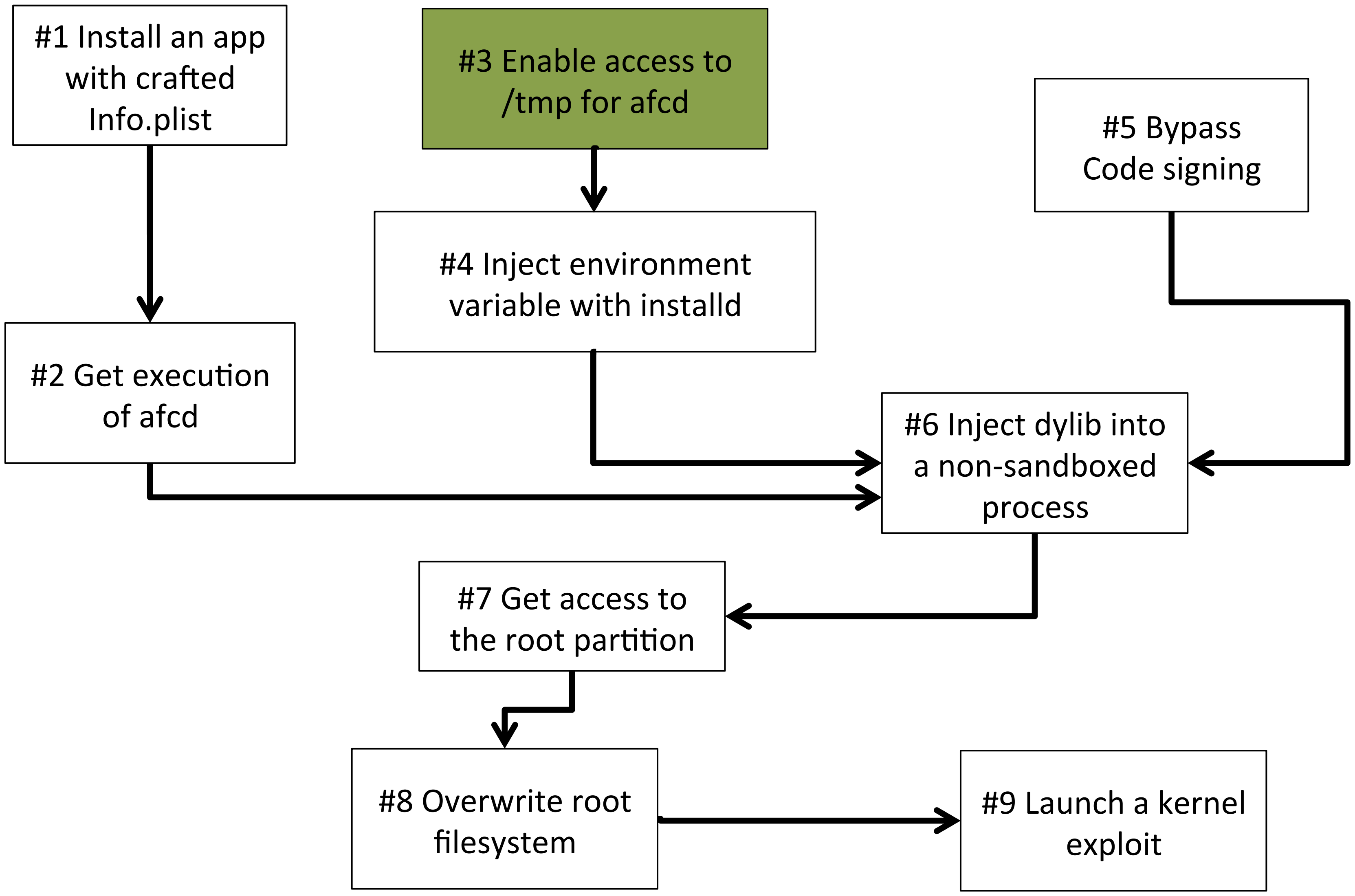


```
Aug  6 09:37:38 Yeong-Jin-Jangs-iPhone afcd[437] <Error>: Got XPC error on listener connection: Connection invalid
Aug  6 09:37:38 Yeong-Jin-Jangs-iPhone com.apple.launchd[1] (UIKitApplication:developer.apple.wwdc-Release[0x96bc][437]) <Error>:
(UIKitApplication:developer.apple.wwdc-Release[0x96bc]) Exited with code: 1
Aug  6 09:37:38 Yeong-Jin-Jangs-iPhone com.apple.launchd[1] (UIKitApplication:developer.apple.wwdc-Release[0x96bc]) <Notice>:
(UIKitApplication:developer.apple.wwdc-Release[0x96bc]) Throttling respawn: Will start in 2147483647 seconds
Aug  6 09:37:38 Yeong-Jin-Jangs-iPhone backboardd[31] <Warning>: Application 'UIKitApplication:developer.apple.wwdc-Release[0x96bc]'
exited abnormally with exit status 1
```

The App Sandbox

- Sandbox is not bypassed yet
  - On execution of afcd, the binary itself initiates its own sandbox
    - Sandbox functions in *libsystem_sandbox.dylib*
      - sandbox_init(), sandbox_init_with_parameters()
      - sandbox_check(), etc.
    - Only allows filesystem access to /var/mobile/Media/*

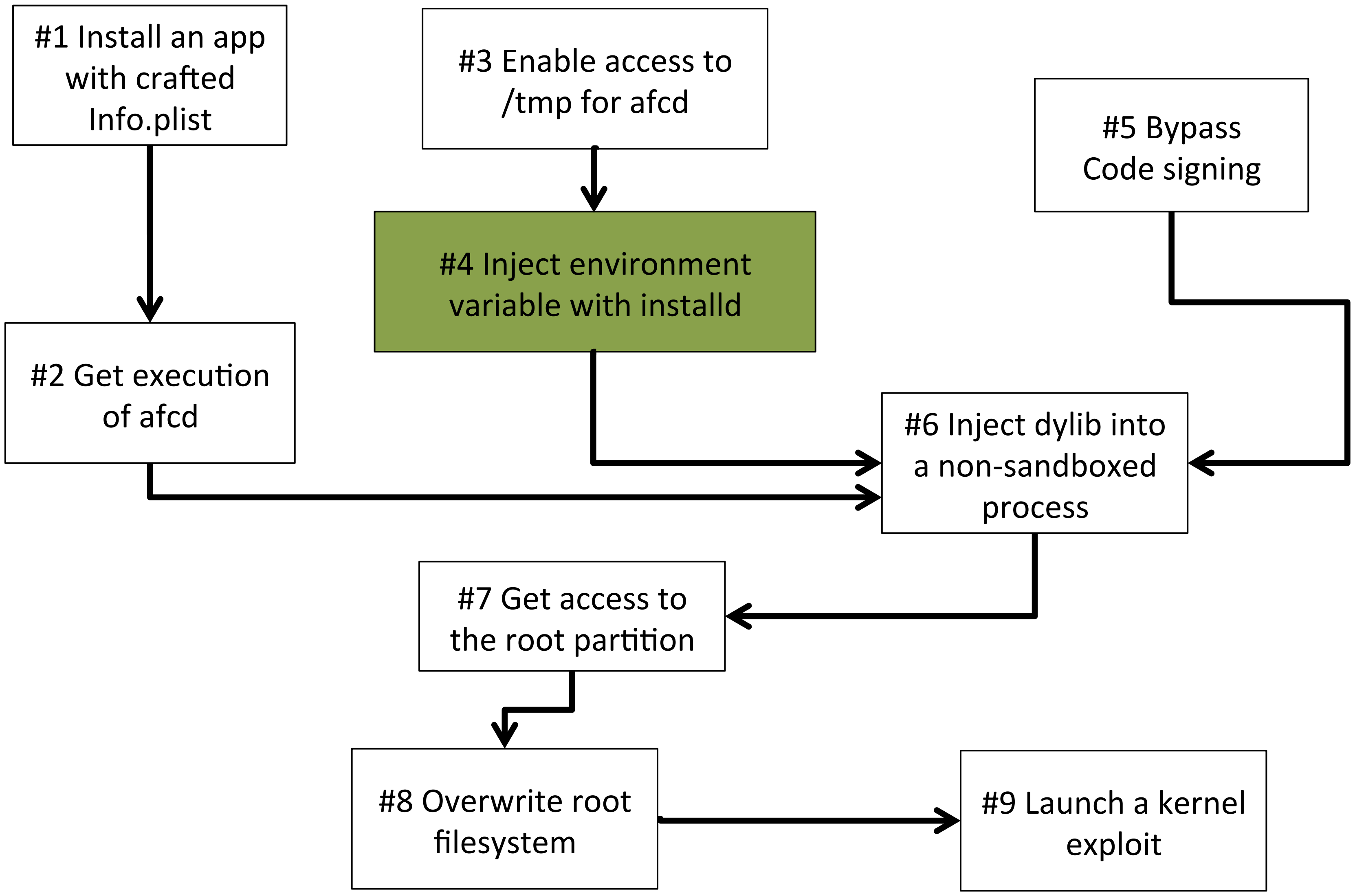
evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n7 – Vulnerability #3

- Enable access to /tmp for afcd
  - The original afcd has no access to /tmp
  - A symlink bug in sandbox policy.
    - Creates a symlink to "`../../../../../tmp`" at /var/mobile/Media/Downloads/a/a/a/a/a/a
      - Link to /var/mobile/Media/Downloads/tmp
      - It is inside of the sandbox!
    - Move the symlink to the upper directory
      - Move to /var/mobile/Media/Downloads…
  - Then afcd gains access to /tmp
    - /var/mobile/Media/Downloads/../../../../../tmp => /tmp!

evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n7 – Vulnerability #4

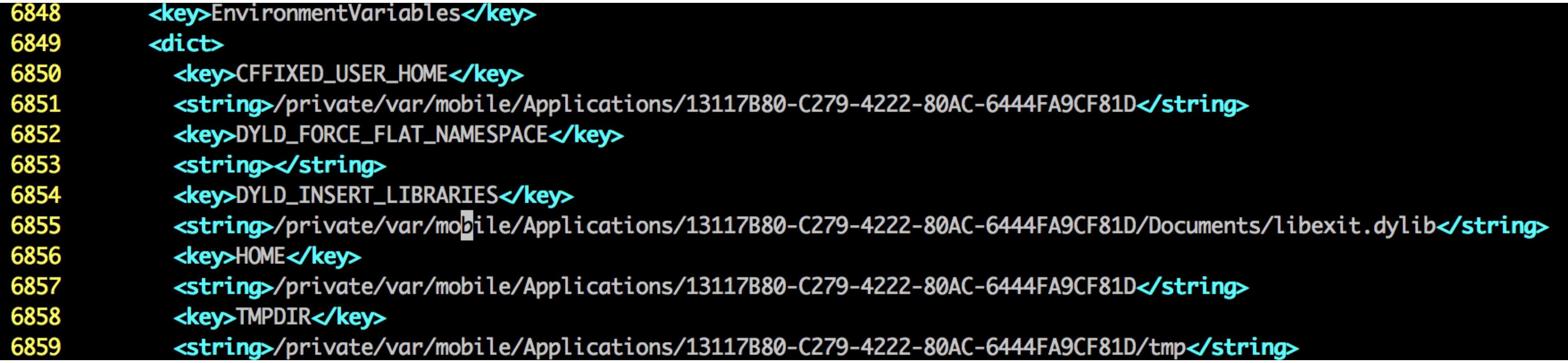
- Inject an environment variable using *installd*
  - During the installation of an app, *installd* will create a temporary directory at /tmp/install_staging.XXXXXX/foo_extracted, and then unzip the ipa file into that directory.

evasi0n7 – Vulnerability #4

- Inject an environment variable using *installd*
  - Exploit: Ask afcd to create a symlink at foo_extracted
    - The symlink links to /var/mobile/Library/Caches/
    - Installd will drop files into /var/mobile/Library/Caches/
    - Does not matter the success of installation…
      - rm –rf /tmp/install_staging.XXXXXX/ would just remove the symlink `foo_extracted`

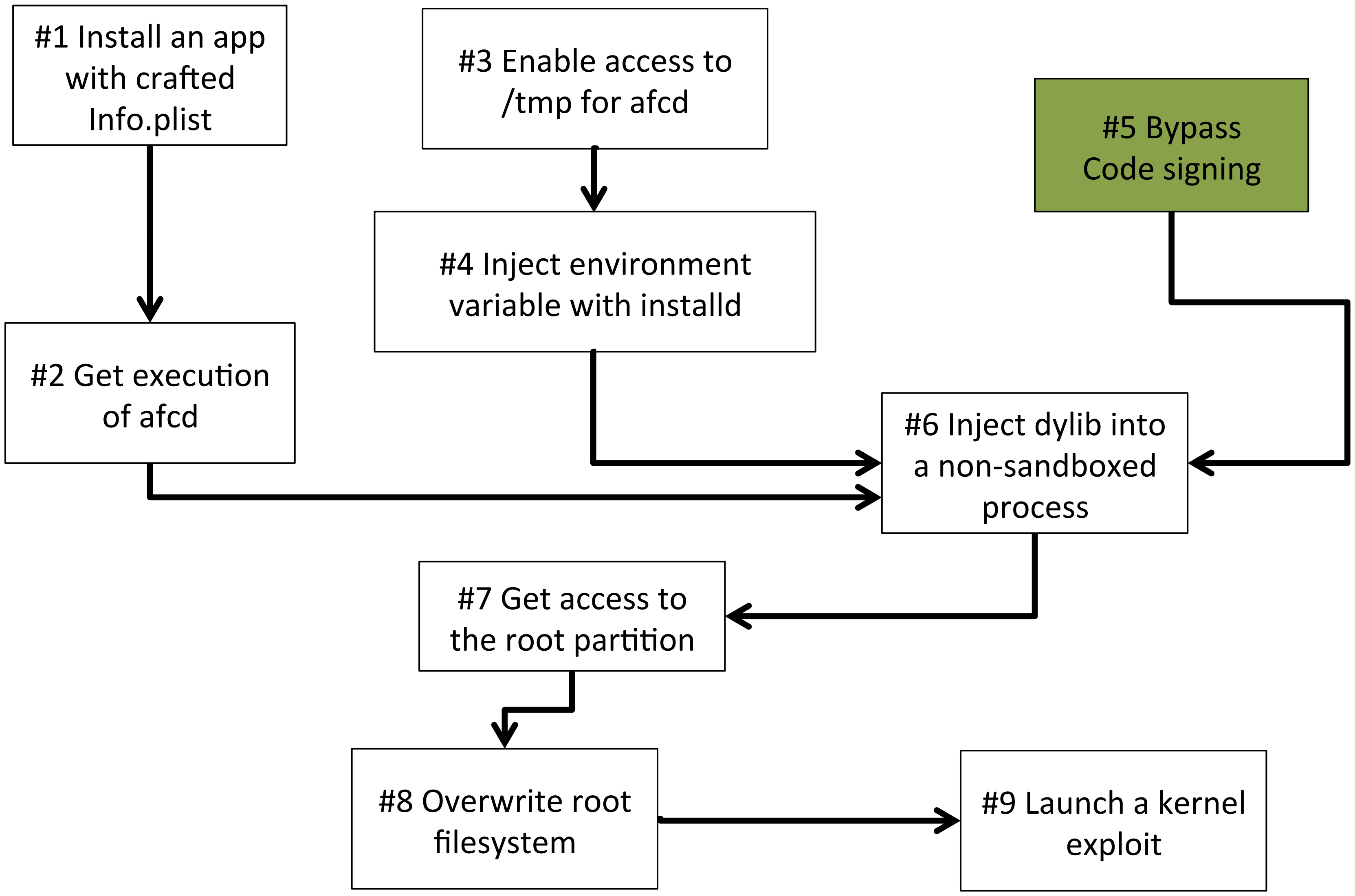
evasi0n7 – Vulnerability #4

- By overwriting com.apple.mobile_installation.plist (in /var/mobile/Library/Caches/), evasi0n7 can specify the DYLD_INSERT_LIBRARIES environment variable for a target app.
  - Injection of dylib is possible!



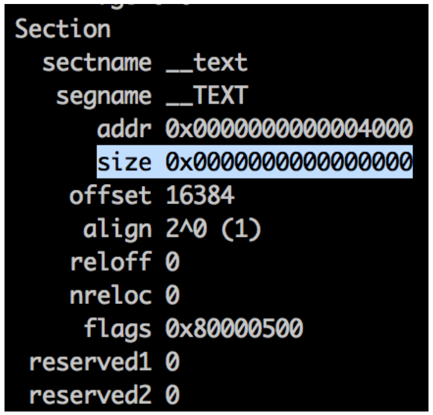
```
      <key>EnvironmentVariables</key>
      <dict>
        <key>CFFIXED_USER_HOME</key>
        <string>/private/var/mobile/Applications/13117B80-C279-4222-80AC-6444FA9CF81D</string>
        <key>DYLD_FORCE_FLAT_NAMESPACE</key>
        <string></string>
        <key>DYLD_INSERT_LIBRARIES</key>
        <string>/private/var/mobile/Applications/13117B80-C279-4222-80AC-6444FA9CF81D/Documents/libexit.dylib</string>
        <key>HOME</key>
        <string>/private/var/mobile/Applications/13117B80-C279-4222-80AC-6444FA9CF81D</string>
        <key>TMPDIR</key>
        <string>/private/var/mobile/Applications/13117B80-C279-4222-80AC-6444FA9CF81D/tmp</string>
```

evasi0n7 Workflow

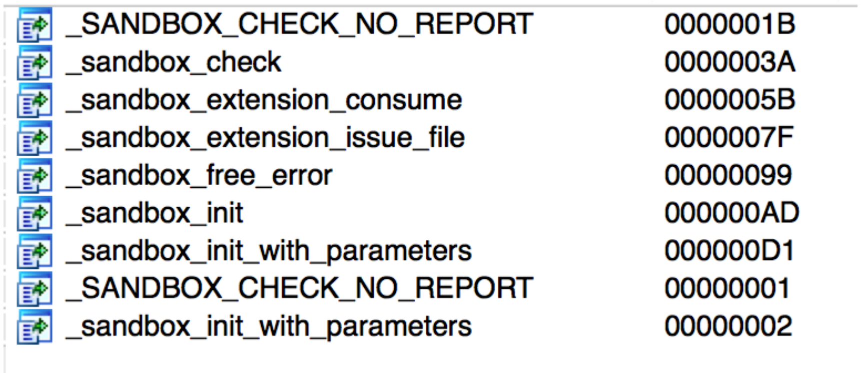

#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n7 – Vulnerability #5

- Inject an unsigned dylib and bypass code signing (gameover.dylib)
  - Size of the code section is 0
    - dyld will ignore this section and will not valid its signature
  - But some executable parts exist
  - And can override some functions

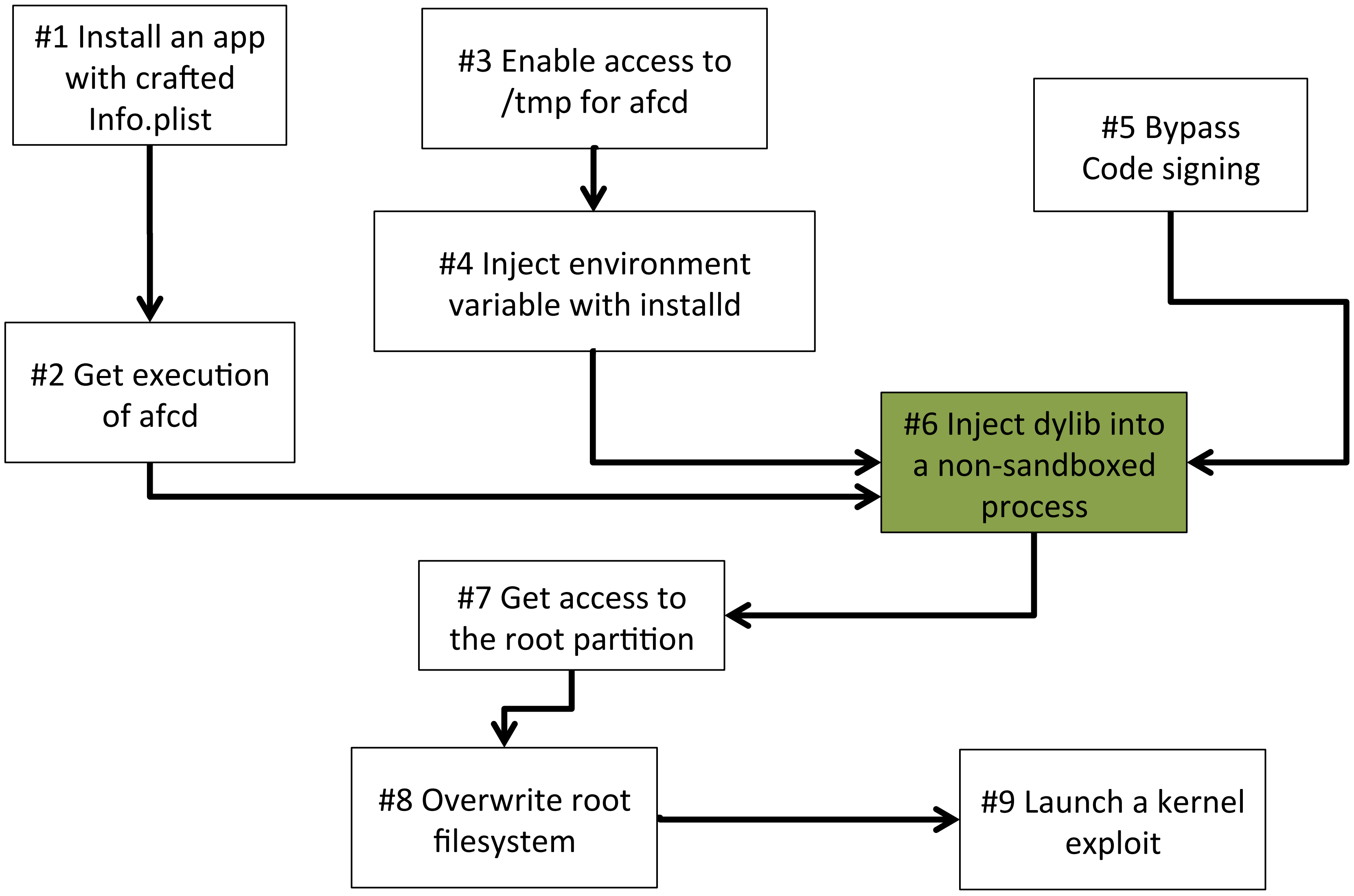


```
Section
  sectname __text
   segname __TEXT
      addr 0x0000000000004000
      size 0x0000000000000000
    offset 16384
     align 2^0 (1)
    reloff 0
    nreloc 0
     flags 0x80000500
 reserved1 0
 reserved2 0
```



```
_SANDBOX_CHECK_NO_REPORT          0000001B
_sandbox_check                    0000003A
_sandbox_extension_consume        0000005B
_sandbox_extension_issue_file     0000007F
_sandbox_free_error               00000099
_sandbox_init                     000000AD
_sandbox_init_with_parameters     000000D1
_SANDBOX_CHECK_NO_REPORT          00000001
_sandbox_init_with_parameters     00000002
```

evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n 7 – Vulnerability #6

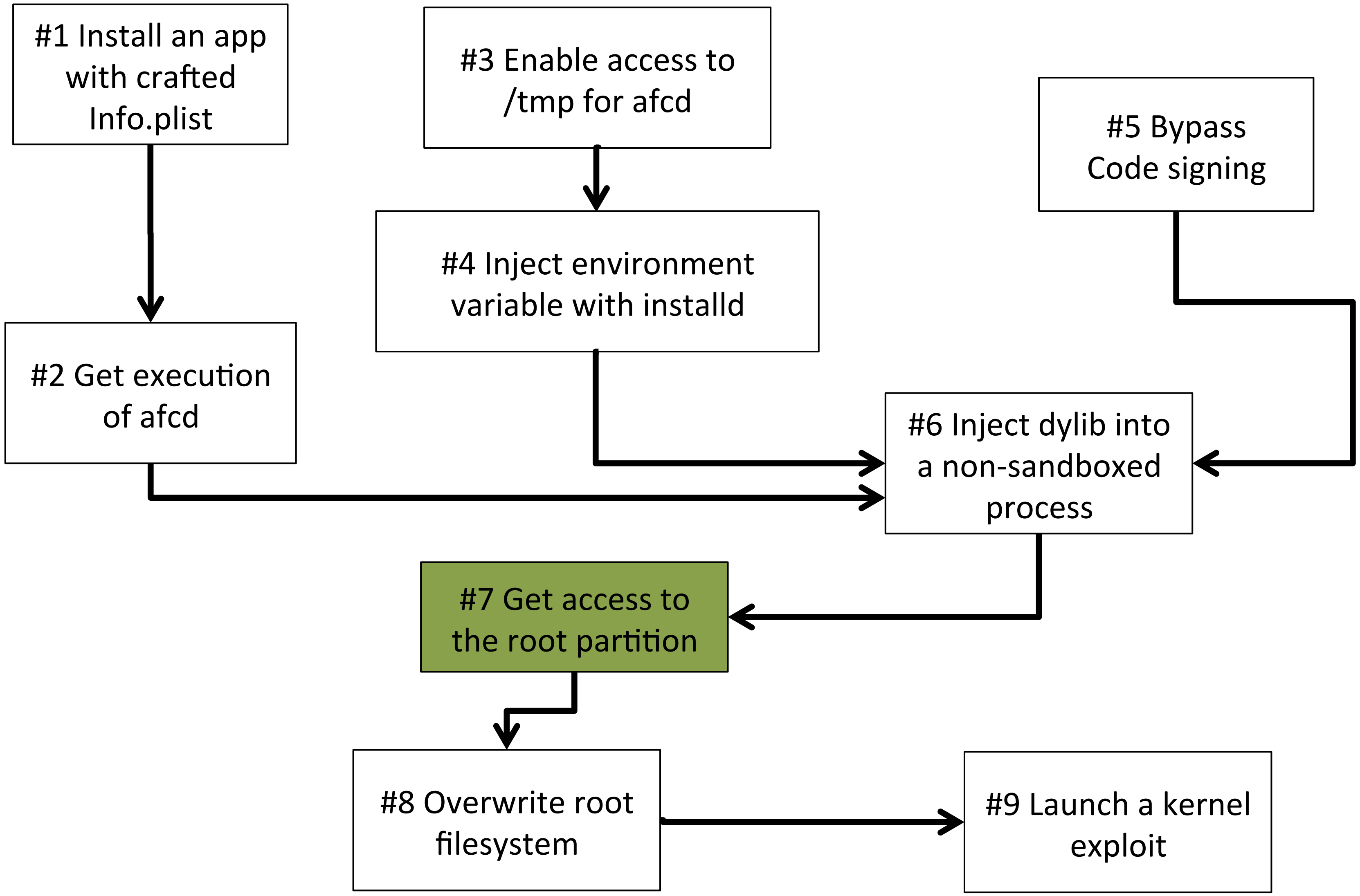
- Combination of #1, #2, #3, #4, #5
  - #1 & #2
    - Can run /usr/libexec/afcd, without kernel-enforced sandbox
  - #3 & #4
    - Inject dylib into execution environment of afcd
  - #5
    - Bypass code signing check, override sandbox related functions

evasi0n 7 – Vulnerability #6

- Clicking the app icon will trigger the execution of “afcd” and load gameover.dylib.

- Since gameover.dylib nullifies the sandbox functions, afcd now runs outside of the sandbox.
    - But still runs as mobile UID (501).

evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp for afcd
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

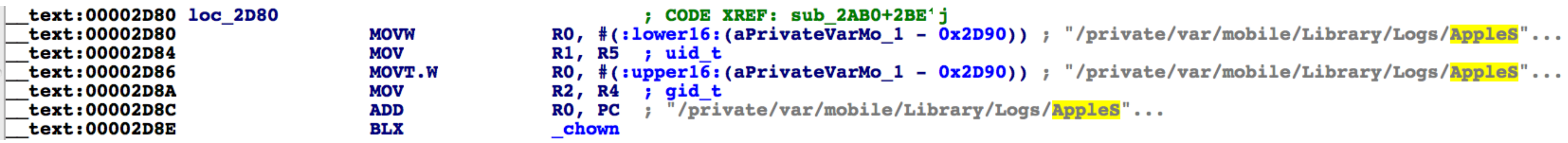
Get Access to Root Partition

- Root partition is mounted as read-only
  - Starting from iOS 7, it is prohibited to re-mount root partition as read/write.
  - Even for processes running with root privilege, they cannot overwrite root partition.
- Then how?
  - Make block device accessible to mobile UID!
    - /dev/rdisk0s1s1 is block device for root partition
    - Can only read/writable by root

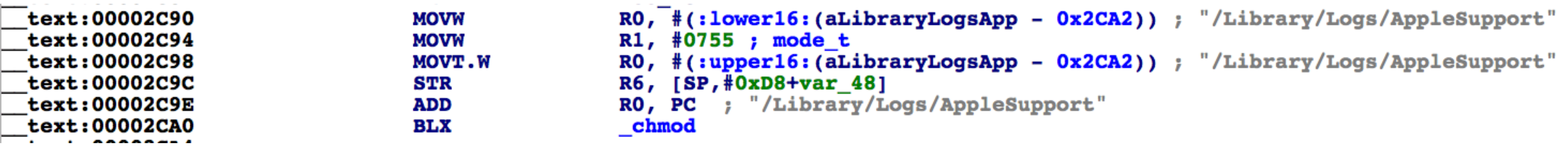
evasi0n7 – Vulnerability #7

- afcd running outside the sandbox now can create a symlink anywhere.

- CrashHouseKeeping, running as root, will do the following:
  - chmod (“/var/mobile/Library/Logs/AppleSupport”, 775)
    - `rwxrwxr-x root root`
  - chown (“/var/mobile/Library/Logs/AppleSupport”, 501, 501)
    - `rwxrwxr-x mobile mobile` …



```
__text:00002D80 loc_2D80                                ; CODE XREF: sub_2AB0+2BE↑j
__text:00002D80                 MOVW            R0, #(:lower16:(aPrivateVarMo_1 - 0x2D90)) ; "/private/var/mobile/Library/Logs/AppleS"...
__text:00002D84                 MOV             R1, R5  ; uid_t
__text:00002D86                 MOVT.W          R0, #(:upper16:(aPrivateVarMo_1 - 0x2D90)) ; "/private/var/mobile/Library/Logs/AppleS"...
__text:00002D8A                 MOV             R2, R4  ; gid_t
__text:00002D8C                 ADD             R0, PC  ; "/private/var/mobile/Library/Logs/AppleS"...
__text:00002D8E                 BLX             _chown
```

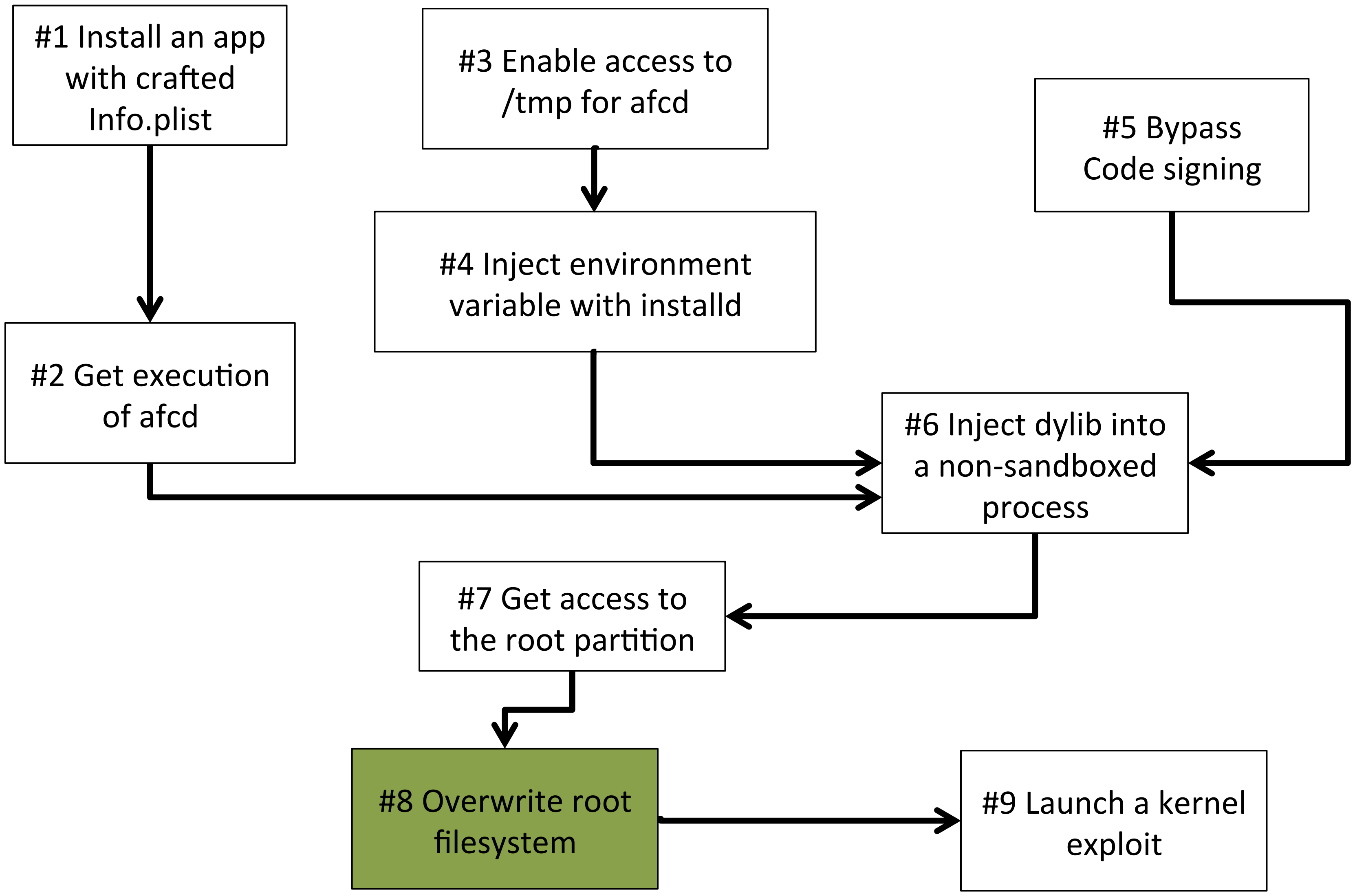


```
__text:00002C90                 MOVW            R0, #(:lower16:(aLibraryLogsApp - 0x2CA2)) ; "/Library/Logs/AppleSupport"
__text:00002C94                 MOVW            R1, #0755 ; mode_t
__text:00002C98                 MOVT.W          R0, #(:upper16:(aLibraryLogsApp - 0x2CA2)) ; "/Library/Logs/AppleSupport"
__text:00002C9C                 STR             R6, [SP,#0xD8+var_48]
__text:00002C9E                 ADD             R0, PC  ; "/Library/Logs/AppleSupport"
__text:00002CA0                 BLX             _chmod
```

evasi0n7 – Vulnerability #7

- Use afcd to create a symlink that points to "../../../../../../../dev/rdisk0s1s1" at "/var/mobile/Library/Logs/AppleSupport"
- With this symlink, CrashHouseKeeping will change /dev/rdisk0s1s1 to be readable/writable by the mobile user
  - `rwxrwxr-x mobile mobile /dev/rdisk0s1s1`

evasi0n7 Workflow


#1 Install an app with crafted Info.plist
#3 Enable access to /tmp for afcd
#5 Bypass Code signing
#4 Inject environment variable with installd
#2 Get execution of afcd
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

evasi0n7 – Vulnerability #8

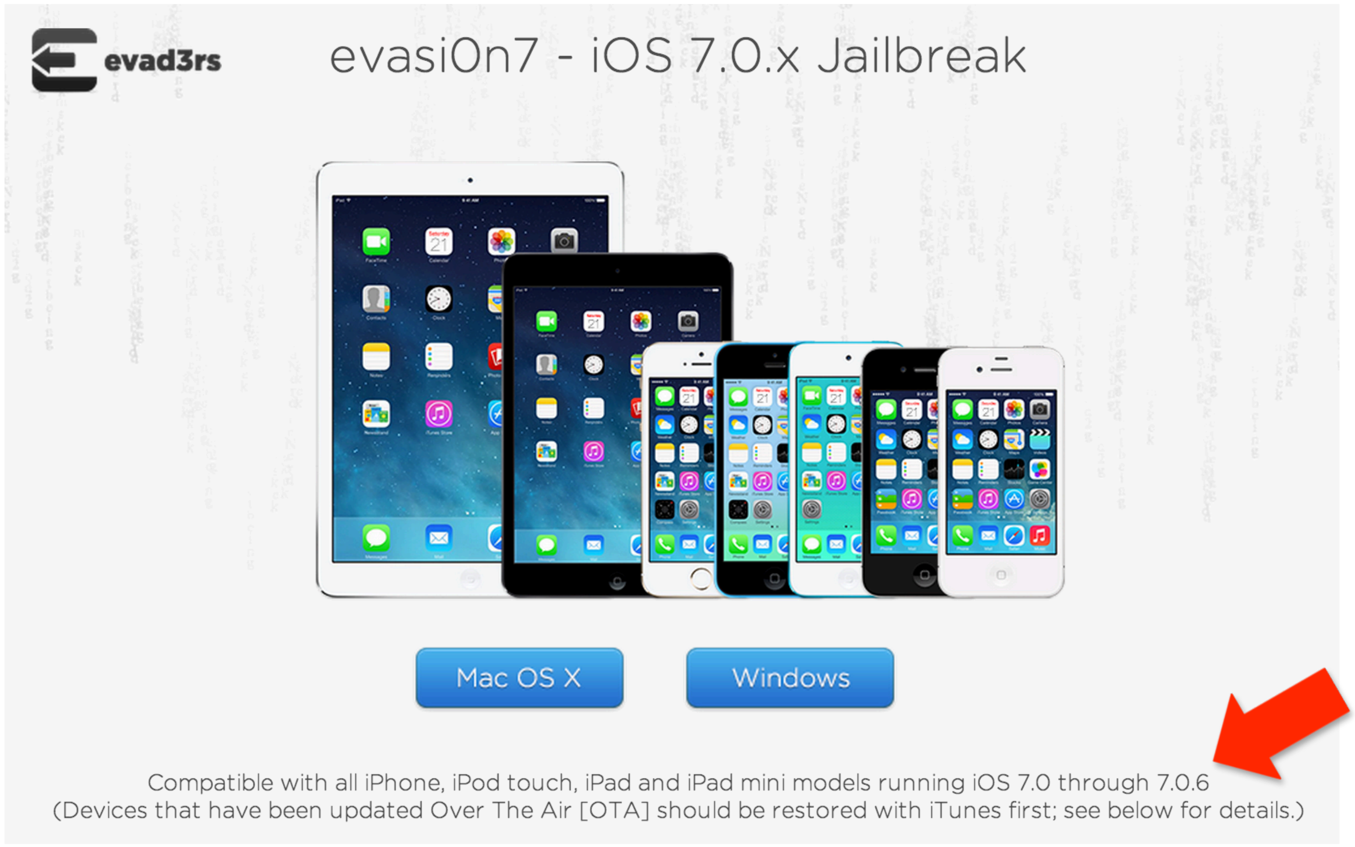
- afcd, running outside of the sandbox, further gains access to the block device
  - With –S option in afcd, it can access special files such as block device.
    - #!/usr/libexec/afcd –S –d / -p 8888
  - Using the AFC protocol, a PC can overwrite the root partition
    - Open /dev/rdisk0s1s1
    - Traverse sub-directories
    - Write files
      - Drop executable for launching kernel exploit on every reboot

evasi0n7 – Vulnerability #9

- A kernel vulnerability is used to patch the kernel
  - Disable code signing check
  - Enable RWX page
  - Enable task_for_pid 0 (debugging kernel process)
  - Enable PE_i_can_has_debugger flag
    - Allow execve of unsigned binary outside of container
      - e.g. executing unsigned /bin/sh

iOS 7.1

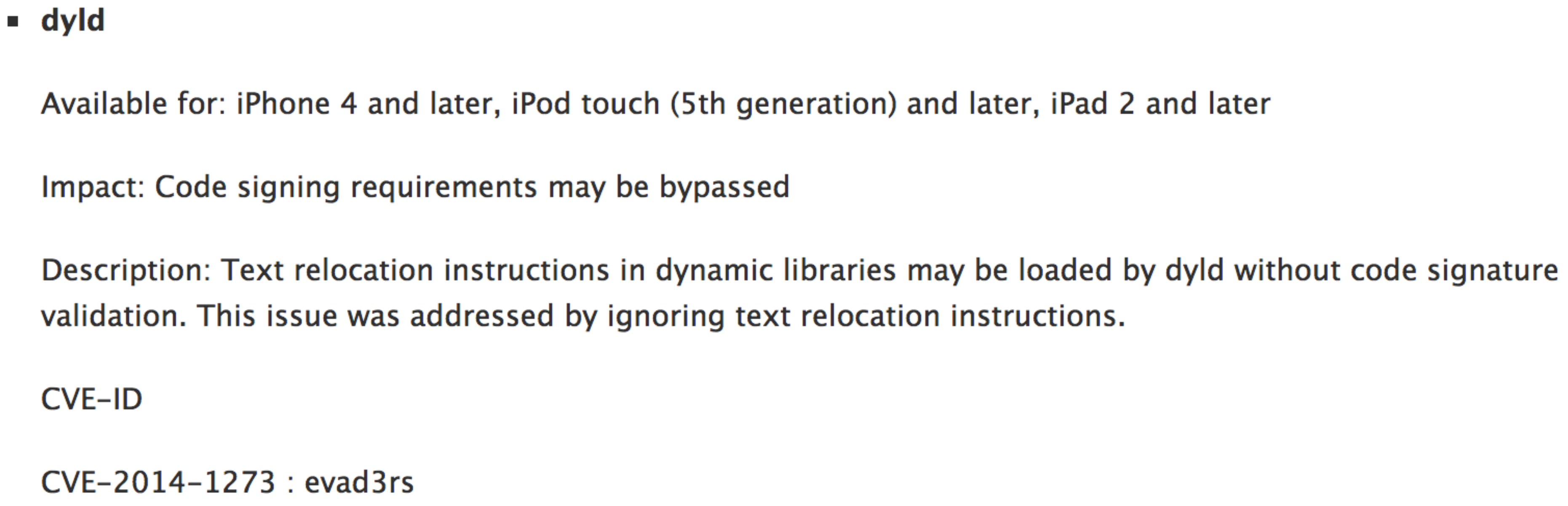
- Apple fixed bugs used in evasi0n7
  - It does not work on 7.1, 7.1.1, and 7.1.2



Compatible with all iPhone, iPod touch, iPad and iPad mini models running iOS 7.0 through 7.0.6
(Devices that have been updated Over The Air [OTA] should be restored with iTunes first; see below for details.)

How was evasi0n7 Patched?

- Patch log from iOS 7.1
  - Patch for bypassing code signing (#5)



- **dyld**

  Available for: iPhone 4 and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: Code signing requirements may be bypassed

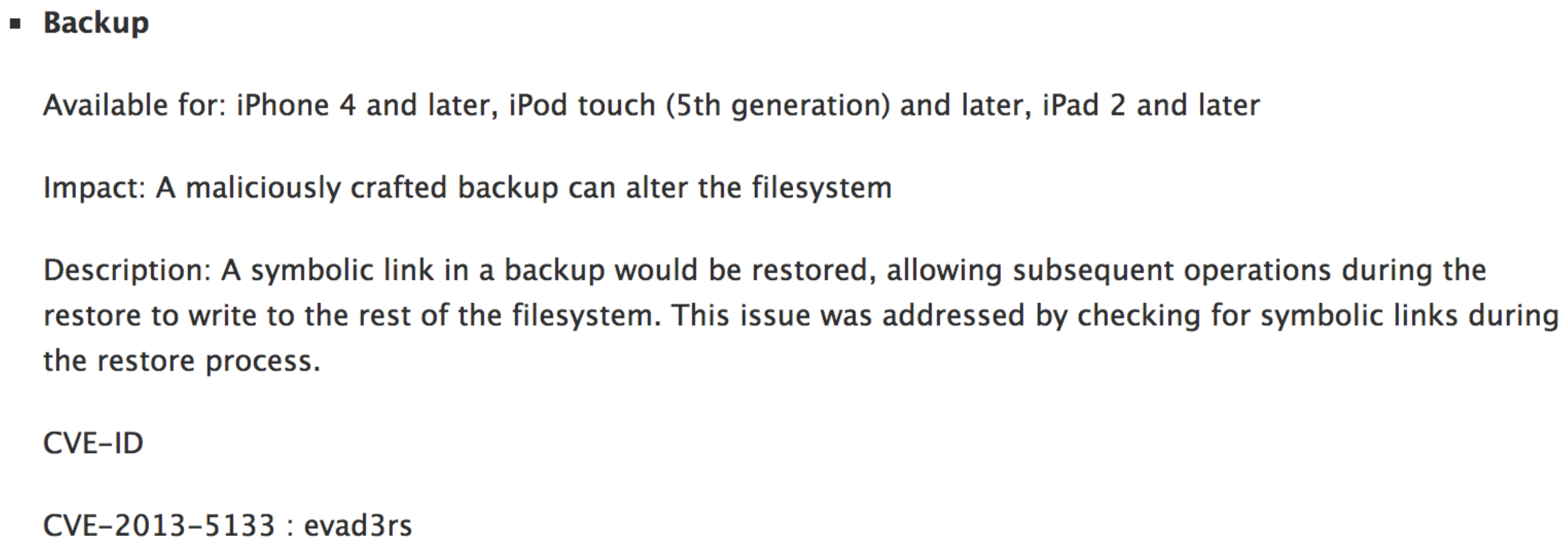
  Description: Text relocation instructions in dynamic libraries may be loaded by dyld without code signature validation. This issue was addressed by ignoring text relocation instructions.

  CVE-ID

  CVE-2014-1273 : evad3rs

How was evasi0n7 Patched?

- Patch log from iOS 7.1
  - Patch for escaping the file system sandbox (#3)



- **Backup**

  Available for: iPhone 4 and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: A maliciously crafted backup can alter the filesystem

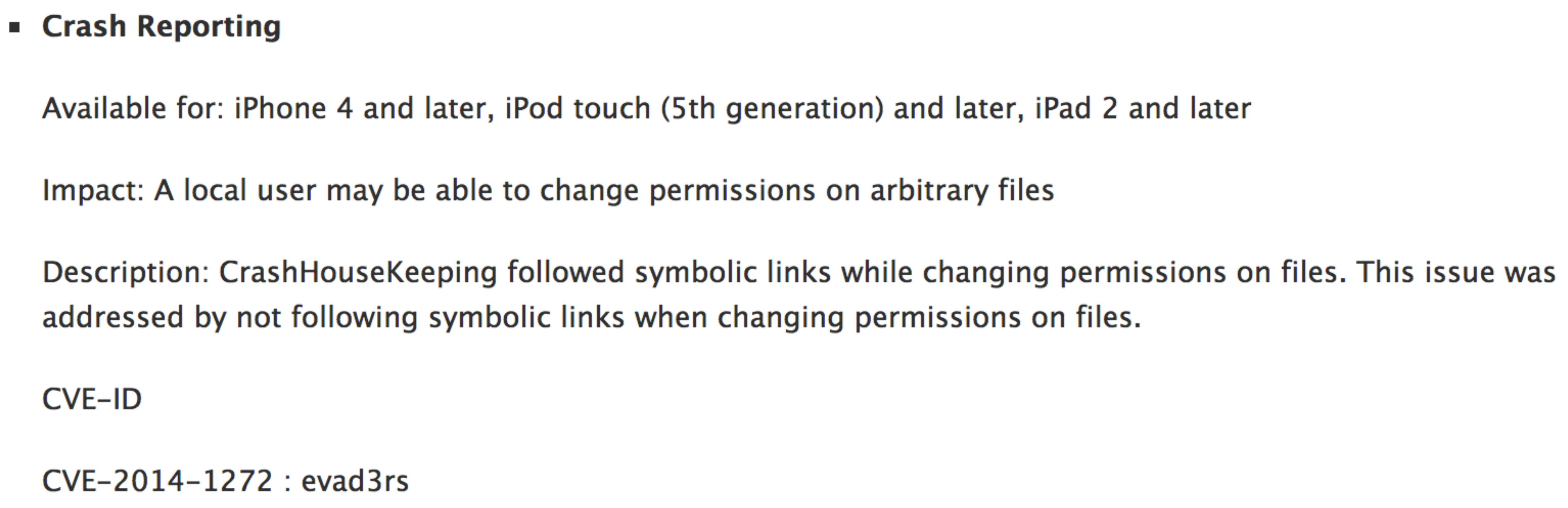
  Description: A symbolic link in a backup would be restored, allowing subsequent operations during the restore to write to the rest of the filesystem. This issue was addressed by checking for symbolic links during the restore process.

  CVE-ID

  CVE-2013-5133 : evad3rs

How was evasi0n7 Patched?

- Patch log from iOS 7.1
  - Patch for the symlink bug in CrashHouseKeeping (#7)



- **Crash Reporting**

  Available for: iPhone 4 and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: A local user may be able to change permissions on arbitrary files

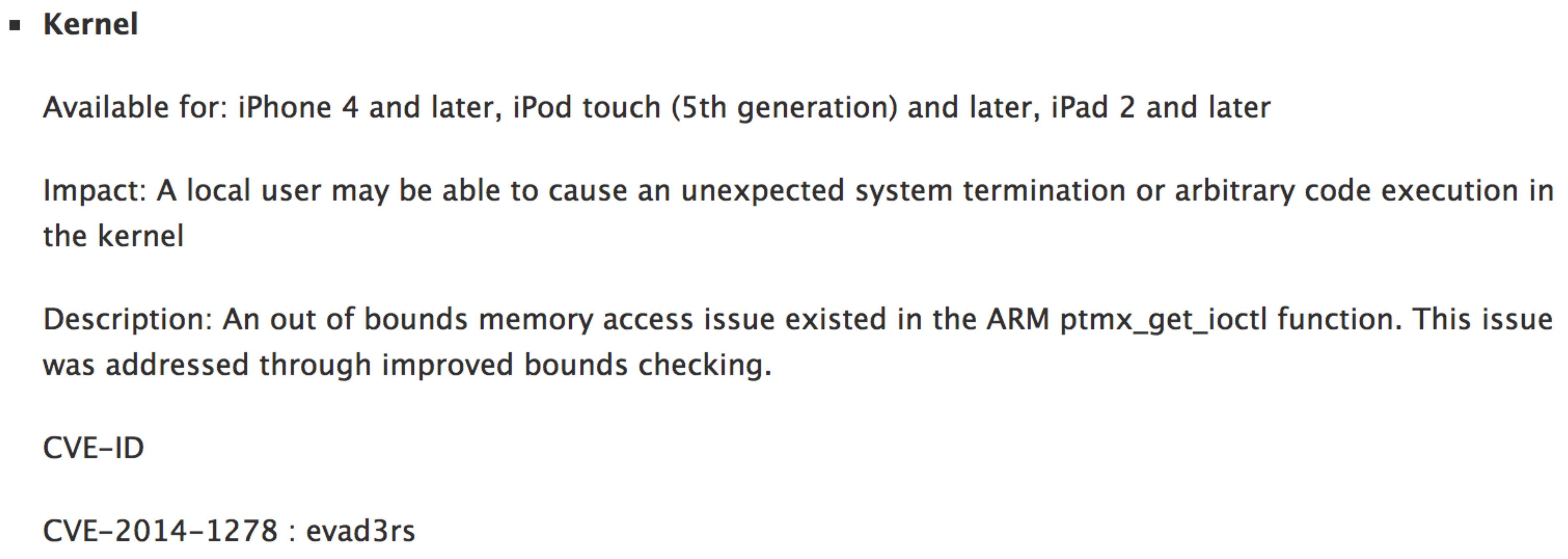
  Description: CrashHouseKeeping followed symbolic links while changing permissions on files. This issue was addressed by not following symbolic links when changing permissions on files.

  CVE-ID

  CVE-2014-1272 : evad3rs

How was evasi0n7 Patched?

- Patch log from iOS 7.1
  - Patch for the kernel vulnerability (#9)



- **Kernel**

  Available for: iPhone 4 and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: A local user may be able to cause an unexpected system termination or arbitrary code execution in the kernel

  Description: An out of bounds memory access issue existed in the ARM ptmx_get_ioctl function. This issue was addressed through improved bounds checking.

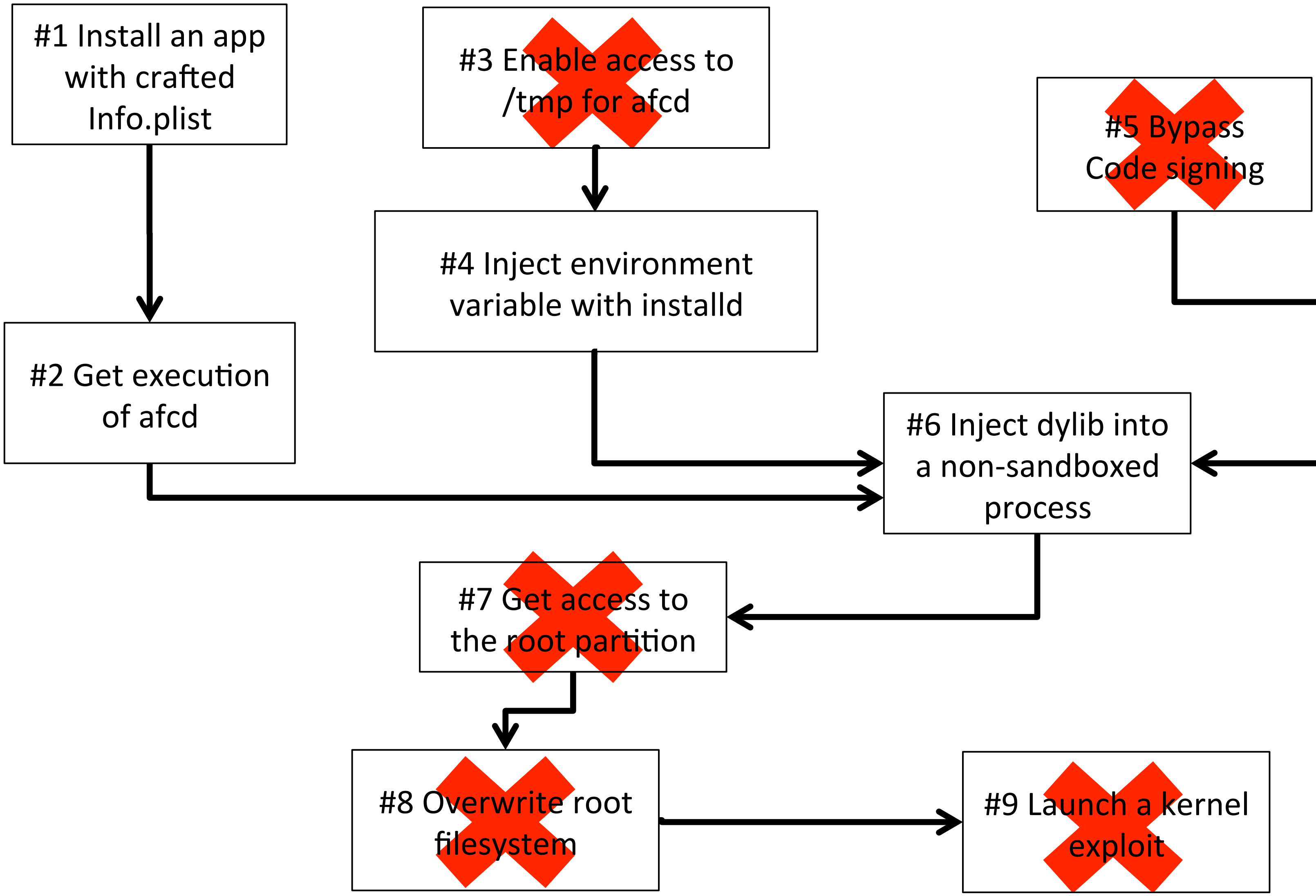
  CVE-ID

  CVE-2014-1278 : evad3rs

How was evasi0n7 Patched?

- Via binary analysis, the “–S” option for afcd was removed (#8)

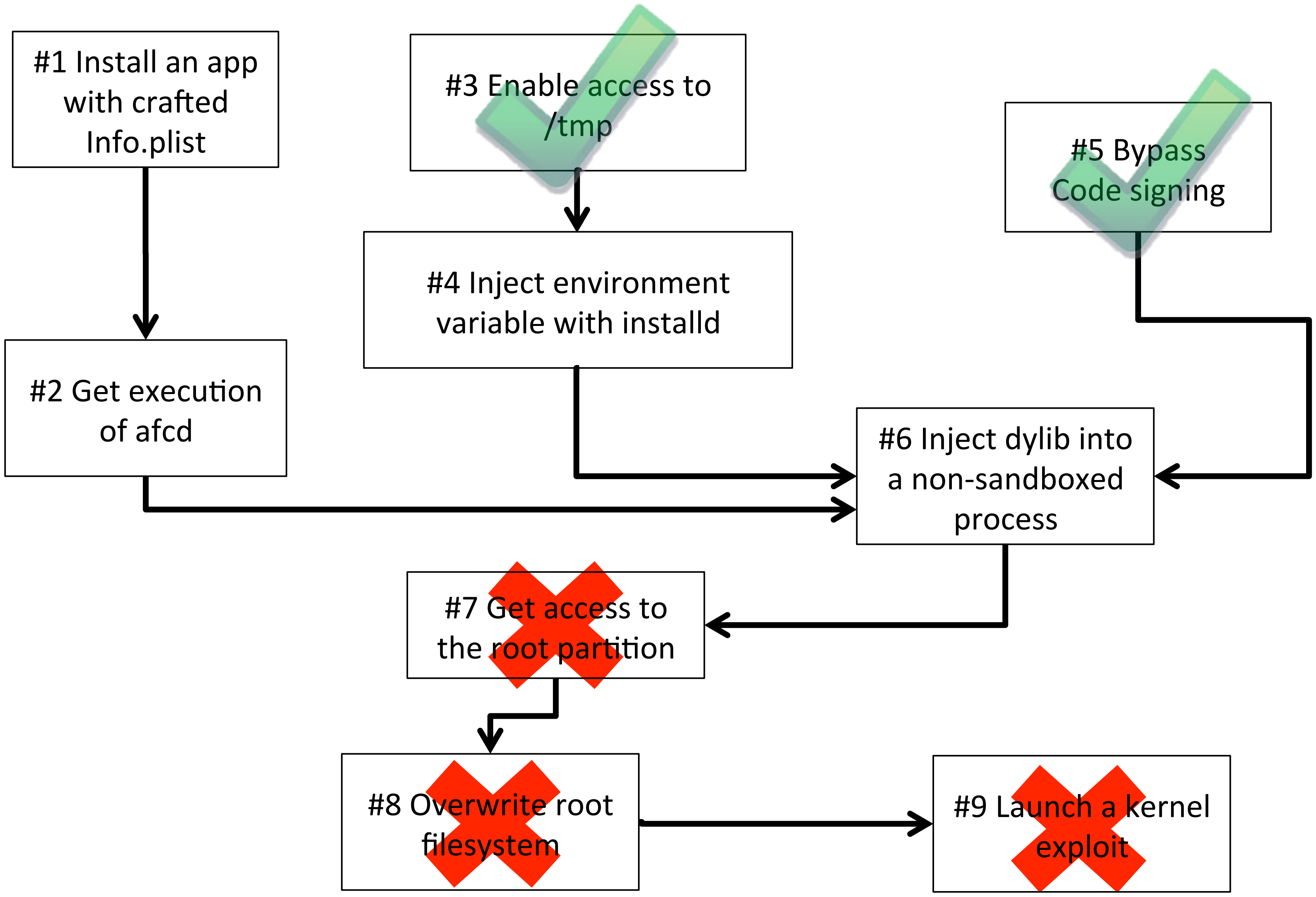
Missing Pieces


#1 Install an app with crafted Info.plist
#3 Enable access to /tmp for afcd
#5 Bypass Code signing
#4 Inject environment variable with installd
#2 Get execution of afcd
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

Our Work

- Attempt to reconstruct the chain of exploits:
  - Find new exploit paths
  - Discover new vulnerabilities

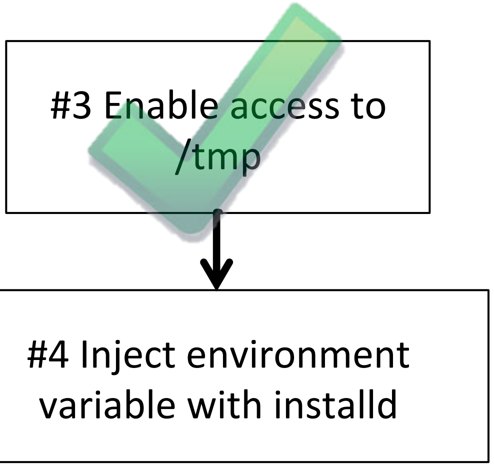
Using Developer Licenses to Enable #3 and #5


#1 Install an app with crafted Info.plist
#3 Enable access to /tmp
#5 Bypass Code signing
#4 Inject environment variable with installd
#2 Get execution of afcd
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

Use Developer Licenses to Enable #3 and #5

- #3: Third party apps have access to /tmp for free
  - Use app to access /tmp to create symlink on exploit #6
- #5: Sign the code with Developer/Enterprise License
  - Load developer-signed dylib in exploit #6

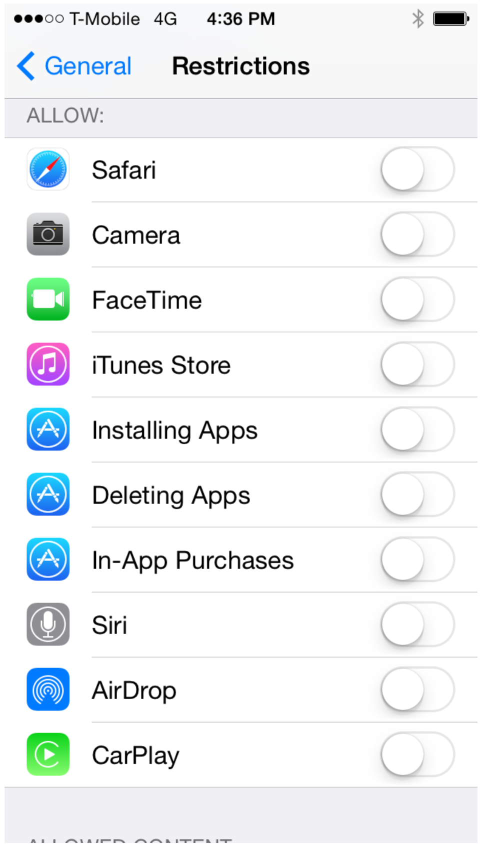
Take a Short Break



- What can we do with just these two vulnerabilities?
  - A malicious app can trick the user to install another app. During this process, it can overwrite many system configurations.

Modifying Configurations

- Restriction Settings
  - In iOS, there exists an option to disable certain features from the device.
  - Using the vulnerability in installd, we could overwrite those settings.


●●●○○ T-Mobile 4G 4:36 PM

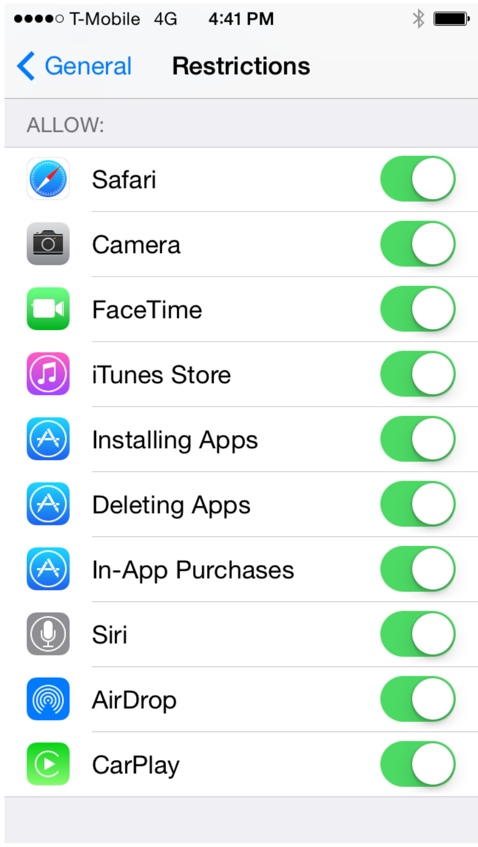
General Restrictions

ALLOW:

Safari	Off
Camera	Off
FaceTime	Off
iTunes Store	Off
Installing Apps	Off
Deleting Apps	Off
In-App Purchases	Off
Siri	Off
AirDrop	Off
CarPlay	Off

ALLOWED CONTENT

Before the attack


●●●●○ T-Mobile 4G 4:41 PM

General Restrictions

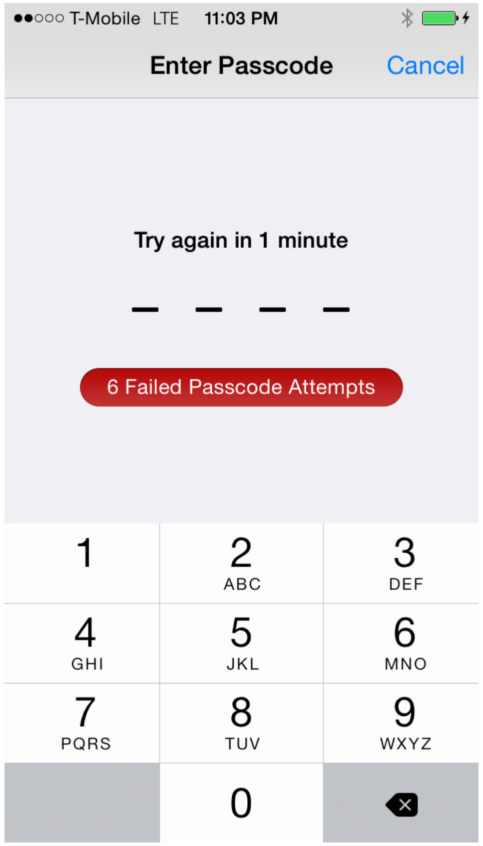
ALLOW:

Safari	On
Camera	On
FaceTime	On
iTunes Store	On
Installing Apps	On
Deleting Apps	On
In-App Purchases	On
Siri	On
AirDrop	On
CarPlay	On

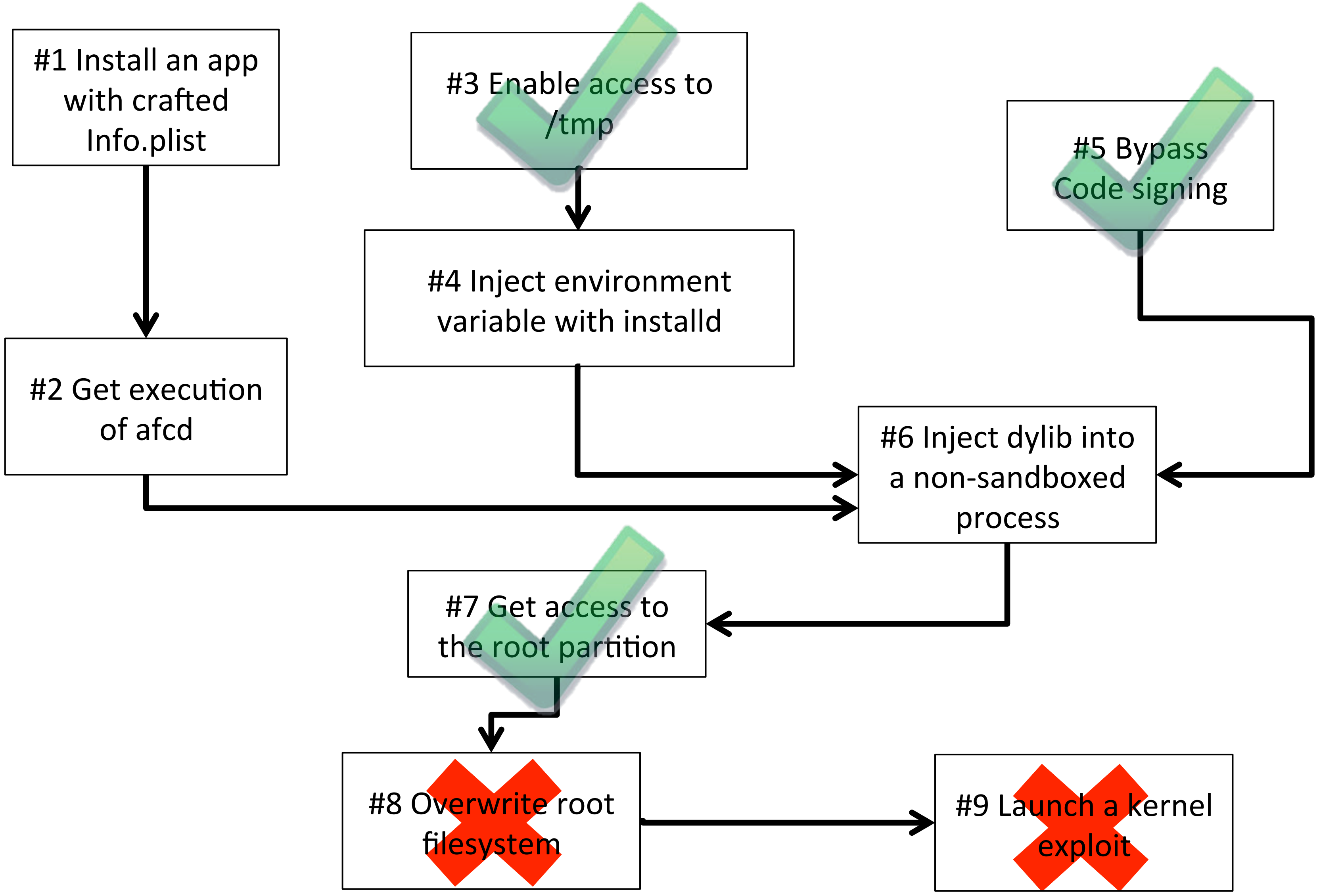
After the attack

Modifying Configurations

- Restriction Settings
  - We can overwrite the passcode for this restriction settings.
  - Since the passcode is not known to the user, the user cannot disable it.



New Vulnerability for Permission Downgrading



#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp
✓
#4 Inject environment variable with installd
#5 Bypass Code signing
✓
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
✓
#8 Overwrite root filesystem
✗
#9 Launch a kernel exploit
✗

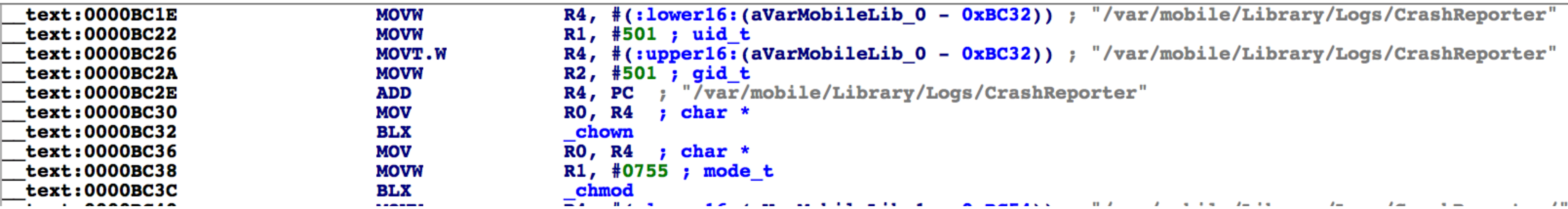
Syslogd Chown Symlink Bug

- grep –E ‘chmod|chown’ -r ./
  - Find all programs that invoke chmod/chown in /usr/libexec
- ps -aux
  - List all daemons running as root in iOS 7.0.6

We are lucky. Find a new one in syslogd in 5 mins

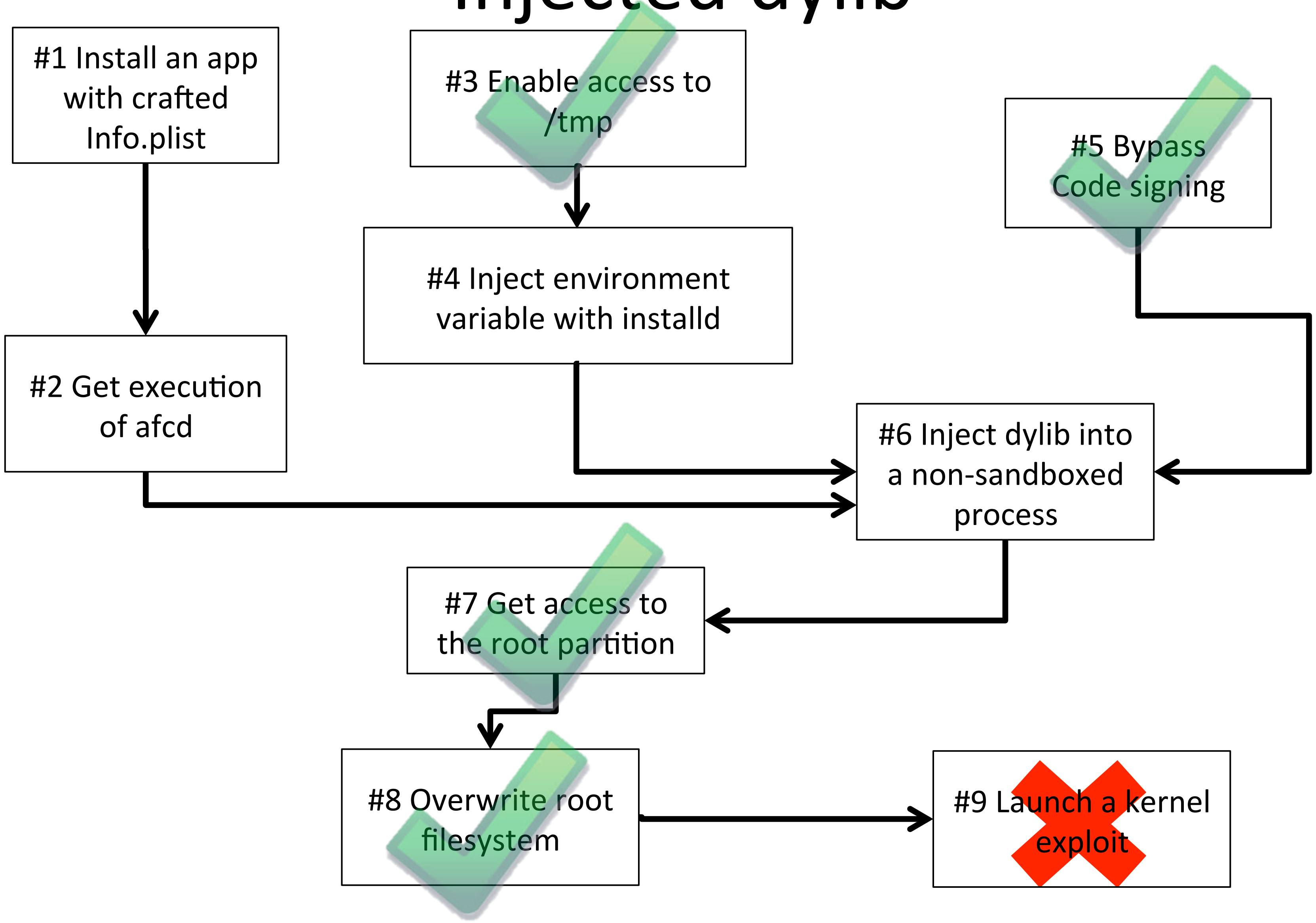
Syslogd Chown Symlink Bug

- chown("/var/mobile/Library/Logs/CrashReporter", 501, 501)
  - UID 501 is mobile
- chmod("/var/mobile/Library/Logs/CrashReporter", 755)
  - `rwxr-xr-x mobile mobile /dev/rdisk0s1s1`



```
__text:0000BC1E                 MOVW            R4, #(:lower16:(aVarMobileLib_0 - 0xBC32)) ; "/var/mobile/Library/Logs/CrashReporter"
__text:0000BC22                 MOVW            R1, #501 ; uid_t
__text:0000BC26                 MOVT.W          R4, #(:upper16:(aVarMobileLib_0 - 0xBC32)) ; "/var/mobile/Library/Logs/CrashReporter"
__text:0000BC2A                 MOVW            R2, #501 ; gid_t
__text:0000BC2E                 ADD             R4, PC  ; "/var/mobile/Library/Logs/CrashReporter"
__text:0000BC30                 MOV             R0, R4  ; char *
__text:0000BC32                 BLX             _chown
__text:0000BC36                 MOV             R0, R4  ; char *
__text:0000BC38                 MOVW            R1, #0755 ; mode_t
__text:0000BC3C                 BLX             _chmod
```

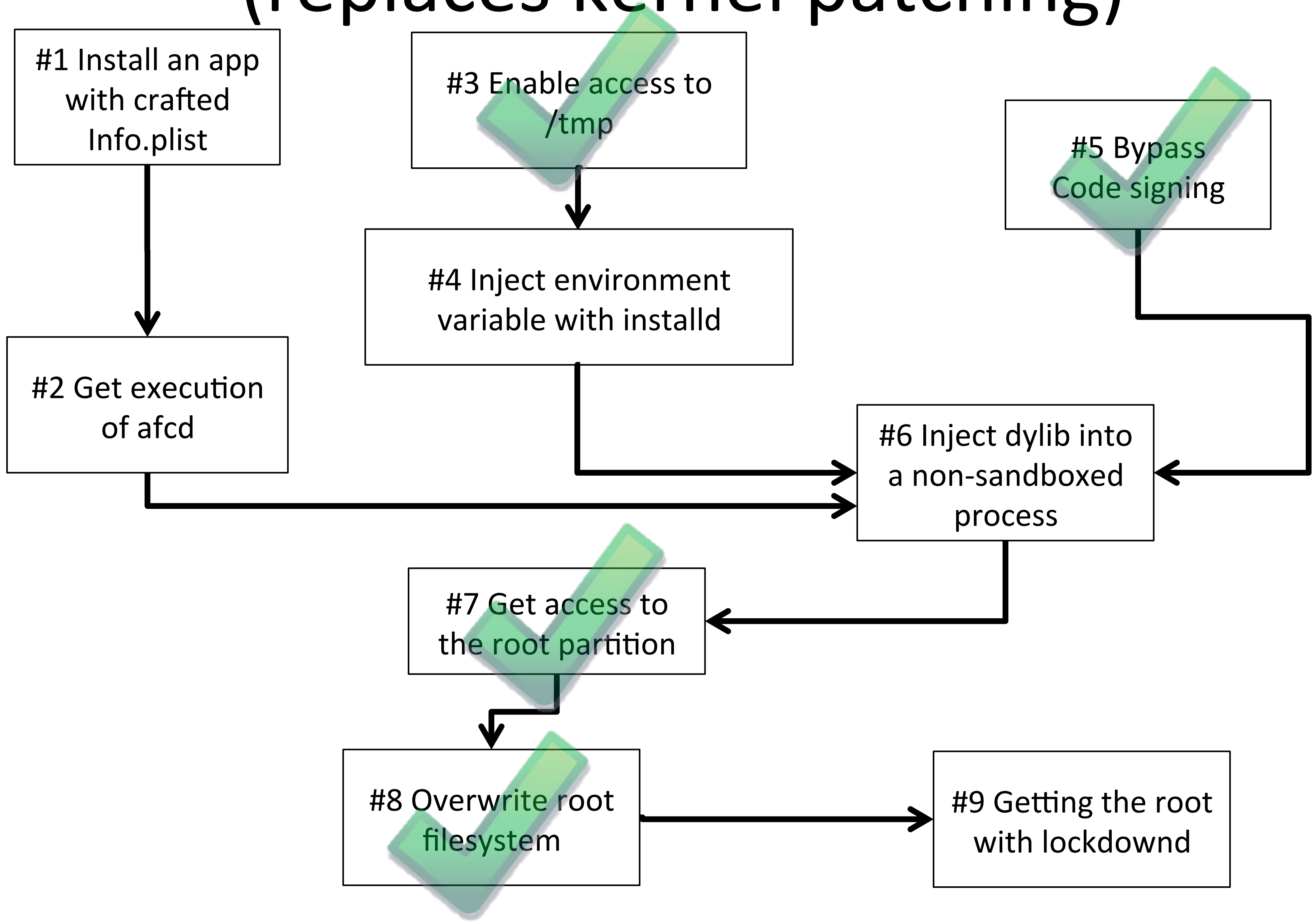
Overwriting the Root Partition with Injected dylib


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Launch a kernel exploit

Overwriting the Root Partition with Injected dylib

- By injecting our dylib into afcd running out of the sandbox, the dylib gains access to /dev/rdisk0s1s1
  - Direct read/write to the block device is possible!

Use lockdownd to obtain root (replaces kernel patching)


#1 Install an app with crafted Info.plist
#2 Get execution of afcd
#3 Enable access to /tmp
#4 Inject environment variable with installd
#5 Bypass Code signing
#6 Inject dylib into a non-sandboxed process
#7 Get access to the root partition
#8 Overwrite root filesystem
#9 Getting the root with lockdownd

Unprotected lockdownd Plist

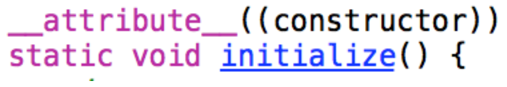
- Setting files of daemons use to be stored in LaunchDaemons directory
  - In iOS 6, plist files in LaunchDaemons are embedded in signed dyld_cache file.
  - We cannot modify settings for existing daemons
- Services.plist of lockdownd is unprotected
  - lockdownd can also launch new services/apps with root privileges.
- Modify Services.plist to run target executable under our control as root.

Modified Steps for Jailbreaking 7.1.2

- #3 Accessing /tmp
  - Install a developer signed app
  - Create symlink as same as evasi0n7 did with afcd

Modified Steps for Jailbreaking 7.1.2

- #5: Forge a dylib to have a constructor, then sign with a developer license
  - – Similar to .ctors in ELF

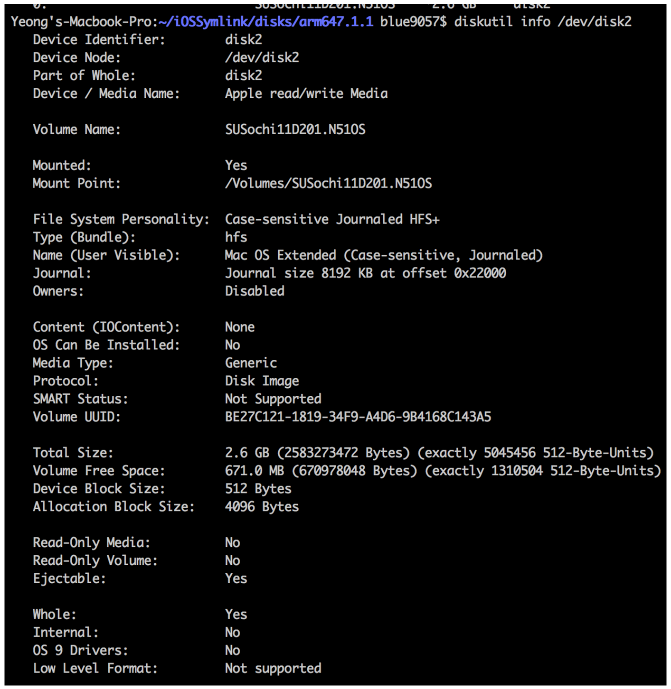


```
__attribute__((constructor))
static void initialize() {
```

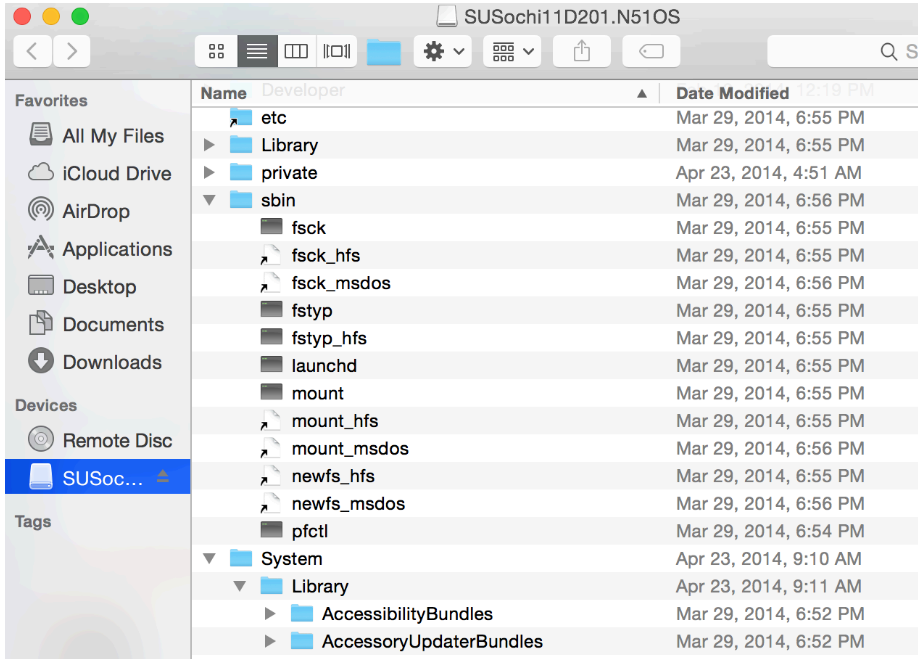
  - – Constructor is called when the dylib is loaded
    - This is before afcd initiates its own sandbox.
    - Injected dylib will be executed outside of sandbox.

Modified Steps for Jailbreaking 7.1.2

- #7: Dump root partition using syslogd exploit, then modify it
  - Download it to PC through AFC



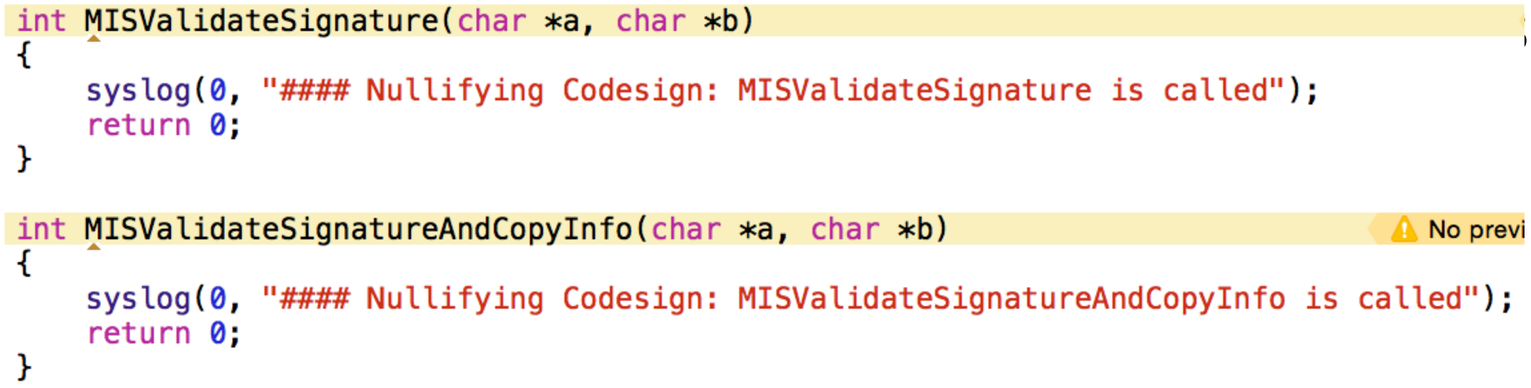
```
Yeong's-Macbook-Pro:~/iOSSymlink/disks/arm647.1.1 blue9057$ diskutil info /dev/disk2
   Device Identifier:        disk2
   Device Node:              /dev/disk2
   Part of Whole:            disk2
   Device / Media Name:      Apple read/write Media

   Volume Name:              SUSochi11D201.N51OS

   Mounted:                  Yes
   Mount Point:              /Volumes/SUSochi11D201.N51OS

   File System Personality:  Case-sensitive Journaled HFS+
   Type (Bundle):            hfs
   Name (User Visible):      Mac OS Extended (Case-sensitive, Journaled)
   Journal:                  Journal size 8192 KB at offset 0x22000
   Owners:                   Disabled

   Content (IOContent):      None
   OS Can Be Installed:      No
   Media Type:               Generic
   Protocol:                 Disk Image
   SMART Status:             Not Supported
   Volume UUID:              BE27C121-1819-34F9-A4D6-9B4168C143A5

   Total Size:               2.6 GB (2583273472 Bytes) (exactly 5045456 512-Byte-Units)
   Volume Free Space:        671.0 MB (670978048 Bytes) (exactly 1310504 512-Byte-Units)
   Device Block Size:        512 Bytes
   Allocation Block Size:    4096 Bytes

   Read-Only Media:          No
   Read-Only Volume:         No
   Ejectable:                Yes

   Whole:                    Yes
   Internal:                 No
   OS 9 Drivers:             No
   Low Level Format:         Not supported
```



Name	Date Modified
etc	Mar 29, 2014, 6:55 PM
▶ Library	Mar 29, 2014, 6:55 PM
▶ private	Apr 23, 2014, 4:51 AM
▼ sbin	Mar 29, 2014, 6:56 PM
fsck	Mar 29, 2014, 6:55 PM
fsck_hfs	Mar 29, 2014, 6:55 PM
fsck_msdos	Mar 29, 2014, 6:56 PM
fstyp	Mar 29, 2014, 6:55 PM
fstyp_hfs	Mar 29, 2014, 6:55 PM
launchd	Mar 29, 2014, 6:55 PM
mount	Mar 29, 2014, 6:55 PM
mount_hfs	Mar 29, 2014, 6:55 PM
mount_msdos	Mar 29, 2014, 6:56 PM
newfs_hfs	Mar 29, 2014, 6:55 PM
newfs_msdos	Mar 29, 2014, 6:56 PM
pfctl	Mar 29, 2014, 6:54 PM
▼ System	Apr 23, 2014, 9:10 AM
▼ Library	Apr 23, 2014, 9:11 AM
▶ AccessibilityBundles	Mar 29, 2014, 6:52 PM
▶ AccessoryUpdaterBundles	Mar 29, 2014, 6:52 PM

Modified Steps for Jailbreaking 7.1.2

- #8: Nullifying code signing check: overriding libmis.dylib

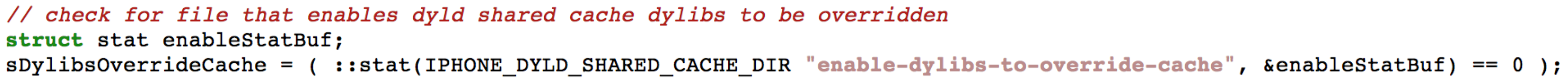


```
int MISValidateSignature(char *a, char *b)
{
    syslog(0, "#### Nullifying Codesign: MISValidateSignature is called");
    return 0;
}

int MISValidateSignatureAndCopyInfo(char *a, char *b)                    ⚠ No previ
{
    syslog(0, "#### Nullifying Codesign: MISValidateSignatureAndCopyInfo is called");
    return 0;
}
```

  – If injected, code signing check will be disabled.

Modified Steps for Jailbreaking 7.1.2

- #8: Nullifying code signing check: overriding libmis.dylib
  - Inject into /usr/lib
  - Touch /System/Library/Caches/com.apple.dyld/enable-dylibs-to-override-cache



```
// check for file that enables dyld shared cache dylibs to be overridden
struct stat enableStatBuf;
sDylibsOverrideCache = ( ::stat(IPHONE_DYLD_SHARED_CACHE_DIR "enable-dylibs-to-override-cache", &enableStatBuf) == 0 );
```

Sourcecode of dyld, from opensource.apple.com

Modified Steps for Jailbreaking 7.1.2

- #8: Nullifying code signing check: overriding libmis.dylib
  - If we make iOS to load `/usr/lib/libmis.dylib`, it will fail to boot
    - Injected libmis.dylib is signed by developer license
    - amfid must be started to allow developer license
      - Otherwise, provisioning profiles will not be loaded.
      - But amfid depends on libmis.dylib
    - A chicken-and-egg problem

Modified Steps for Jailbreaking 7.1.2

- #8: Nullifying code signing check: overriding libmis.dylib
  - Create symlink enable-dylibs-to-override-cache pointing to /tmp/bypass_codesign
  - At boot time, since tmpfs is a kind of ramdisk, it is empty!
  - dyld will not load /usr/lib/libmis.dylib
    - dyld checks existence with stat(), not lstat()
    - The original library will be loaded into amfid at boot time

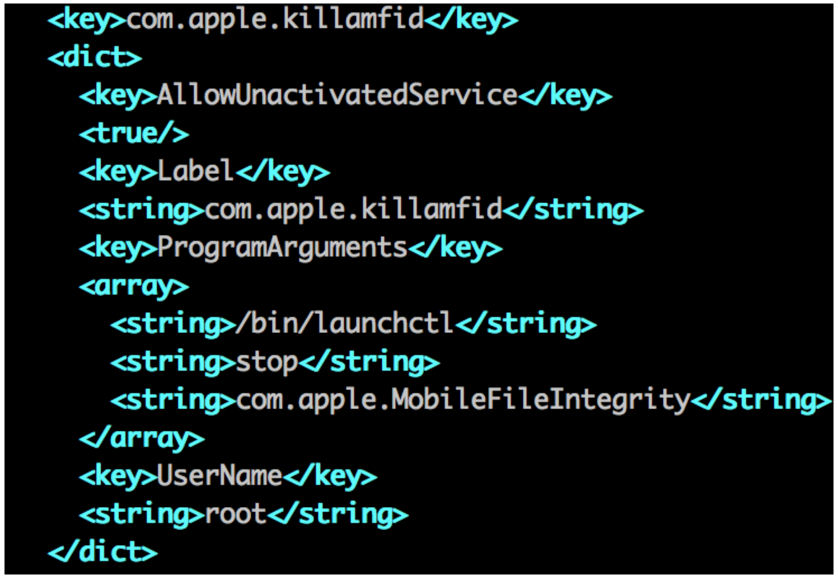
```
// check for file that enables dyld shared cache dylibs to be overridden
struct stat enableStatBuf;
sDylibsOverrideCache = ( ::stat(IPHONE_DYLD_SHARED_CACHE_DIR "enable-dylibs-to-override-cache", &enableStatBuf) == 0 );
```

Modified Steps for Jailbreaking 7.1.2

- #9: Kill amfid & installd
  - We create /tmp/bypass_codesign after the boot process
    - amfid & installd are already loaded with stock libmis.dylib
  - Then we kill and reload the daemons
    - Killing amfid requires root permissions.

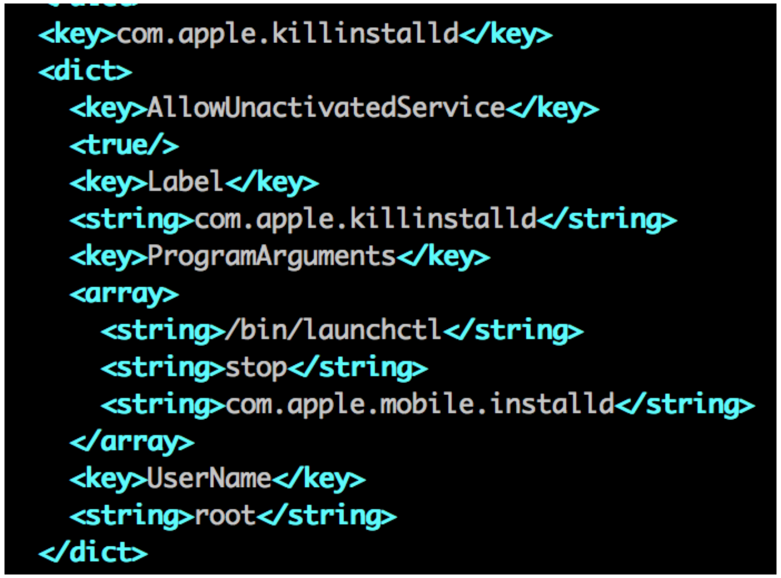
Modified Steps for Jailbreaking 7.1.2

- #9: Edit /System/Library/Lockdown/ Services.plist



```
<key>com.apple.killamfid</key>
<dict>
  <key>AllowUnactivatedService</key>
  <true/>
  <key>Label</key>
  <string>com.apple.killamfid</string>
  <key>ProgramArguments</key>
  <array>
    <string>/bin/launchctl</string>
    <string>stop</string>
    <string>com.apple.MobileFileIntegrity</string>
  </array>
  <key>UserName</key>
  <string>root</string>
</dict>
```

Script for killing amfid



```
<key>com.apple.killinstalld</key>
<dict>
  <key>AllowUnactivatedService</key>
  <true/>
  <key>Label</key>
  <string>com.apple.killinstalld</string>
  <key>ProgramArguments</key>
  <array>
    <string>/bin/launchctl</string>
    <string>stop</string>
    <string>com.apple.mobile.installd</string>
  </array>
  <key>UserName</key>
  <string>root</string>
</dict>
```

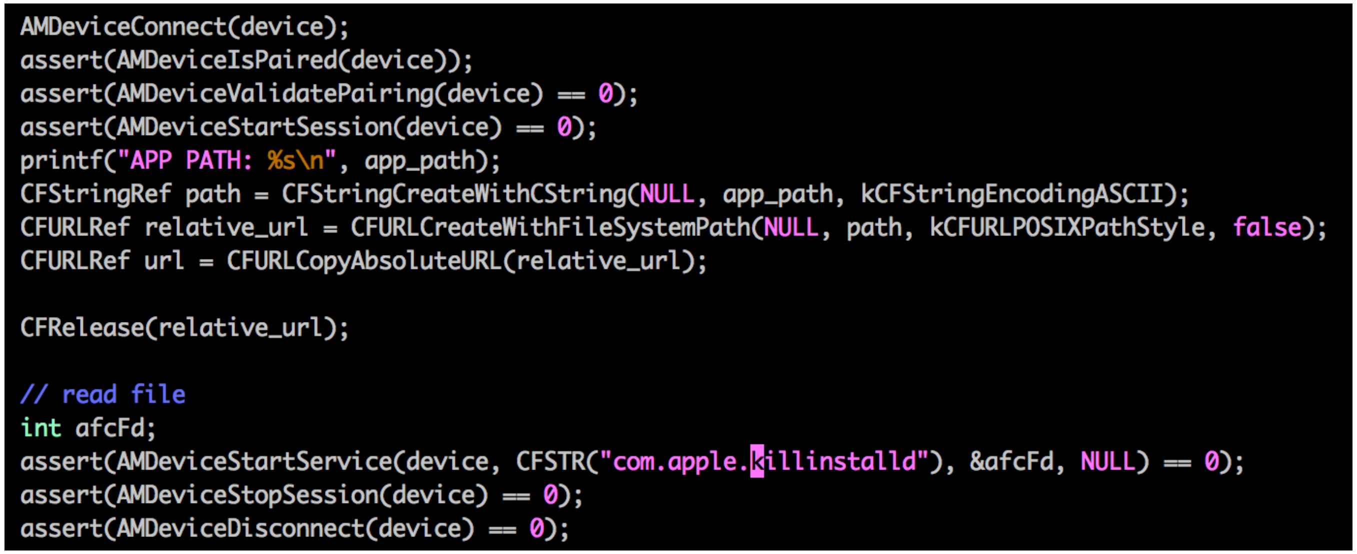
Script for killing installd

Modified Steps for Jailbreaking 7.1.2

- Writeback root partition, then reboot
  - Upload diskimage with AFC
  - open(/dev/rdisk0s1s1);
  - Write modified data…

Modified Steps for Jailbreaking 7.1.2

- #9: Kill daemons with lockdownd
  - lockdownd is a service that processes commands from USB connections.
    - Can be called by a USB connection
    - Can be called by connecting to 127.0.0.1:62078



```
AMDeviceConnect(device);
assert(AMDeviceIsPaired(device));
assert(AMDeviceValidatePairing(device) == 0);
assert(AMDeviceStartSession(device) == 0);
printf("APP PATH: %s\n", app_path);
CFStringRef path = CFStringCreateWithCString(NULL, app_path, kCFStringEncodingASCII);
CFURLRef relative_url = CFURLCreateWithFileSystemPath(NULL, path, kCFURLPOSIXPathStyle, false);
CFURLRef url = CFURLCopyAbsoluteURL(relative_url);

CFRelease(relative_url);

// read file
int afcFd;
assert(AMDeviceStartService(device, CFSTR("com.apple.killinstalld"), &afcFd, NULL) == 0);
assert(AMDeviceStopSession(device) == 0);
assert(AMDeviceDisconnect(device) == 0);
```

Demo Video

Jailbreak Complete

- Attacker can execute code outside of the sandbox
  - A dylib injected into afcd already does this
- Attacker can execute unsigned code
  - Newly started amfid & installd will load modified libmis.dylib
    - Attacker can install & run unsigned binaries
- Attacker has a privileged root process
  - Via hooking daemons running as root

Limitations

- Our exploit does not use a kernel vulnerability
  - We cannot patch the kernel
- We cannot:
  - Execve a non-container binary
    - Can be replaced with fork() & dlopen()
  - Disable sandbox of container binary
    - Can be delegated to a sandbox-free process
  - Debug the kernel

Lessons

- Jailbreak usually requires multiple vulnerabilities to achieve.
- Fixing some of vulnerabilities on the chain may block the current jailbreak attack.
- Incompletely patching the disclosed vulnerabilities still leaves the door for other attacks.

References

1. https://github.com/comex/datautils0/blob/master/make_kernel_patchfile.c
2. http://geohot.com/e7writeup.html
3. http://theiphonewiki.com/wiki/Evasi0n7 (will be updated per each write-ups)
4. https://conference.hitb.org/hitbsecconf2013ams/materials/D2T1%20-%20Pod2g,%20Planetbeing,%20Musclenerd%20and%20Pimskeks%20aka%20Evad3rs%20-%20Swiping%20Through%20Modern%20Security%20Features.pdf
5. http://theiphonewiki.com/wiki//System/Library/Lockdown/Services.plist
6. http://support.apple.com/kb/HT6162
7. http://support.apple.com/kb/HT6208
8. http://securitylearn.net/wp-content/uploads/iOS%20Resources/Apple%20iOS%204%20Security%20Evaluation%20WP.pdf
9. http://www.semantiscope.com/research/BHDC2011/BHDC2011-Slides.pdf

Questions?

- Thank you for your attention!
- Thanks to evad3rs for their jailbreak tool.
- Thanks to geohot for his detailed write-up.