JAILBREAK DETECTION THE MODERN WAY

COOCOOROOCON 2022/11/20

VADIM YEGOROV

WHOAMI

Vadim Yegorov
@vadimszzz on Github

Worked as a Full Stack Engineer more than 2 years
Right now I am Security Researcher for 2.5 years

Part of a team and active contributor in OWASP MSTG
iOS Security teacher for large corporations teams at CyberEd (HackerU branch)

Introduction

Jailbreak detection

- iOS
  - Closed operating system
  - JailBreaks bypass iOS security to get (almost) full access

- JailBreak detection
  - Used by banking applications and games
  - To make sure that the environment is “safe”...
  - ...or to block cheats/cracks

- Security researchers need to
  Assess / reverse protected applications

Debugging an iOS app

- Without a JailBreak
  - With ptrace (lldb / frida) → app needs the get-task-allow entitlement
  - By injecting frida code → app needs to be repackaged
  - In both case, you need to resign the application but it has a lot of side effects:
    - Different Team ID
    - Files are modified

- With a JailBreak
  - No entitlements are required
  - Frida is able to attach to any process

Jailbreak detection evasion

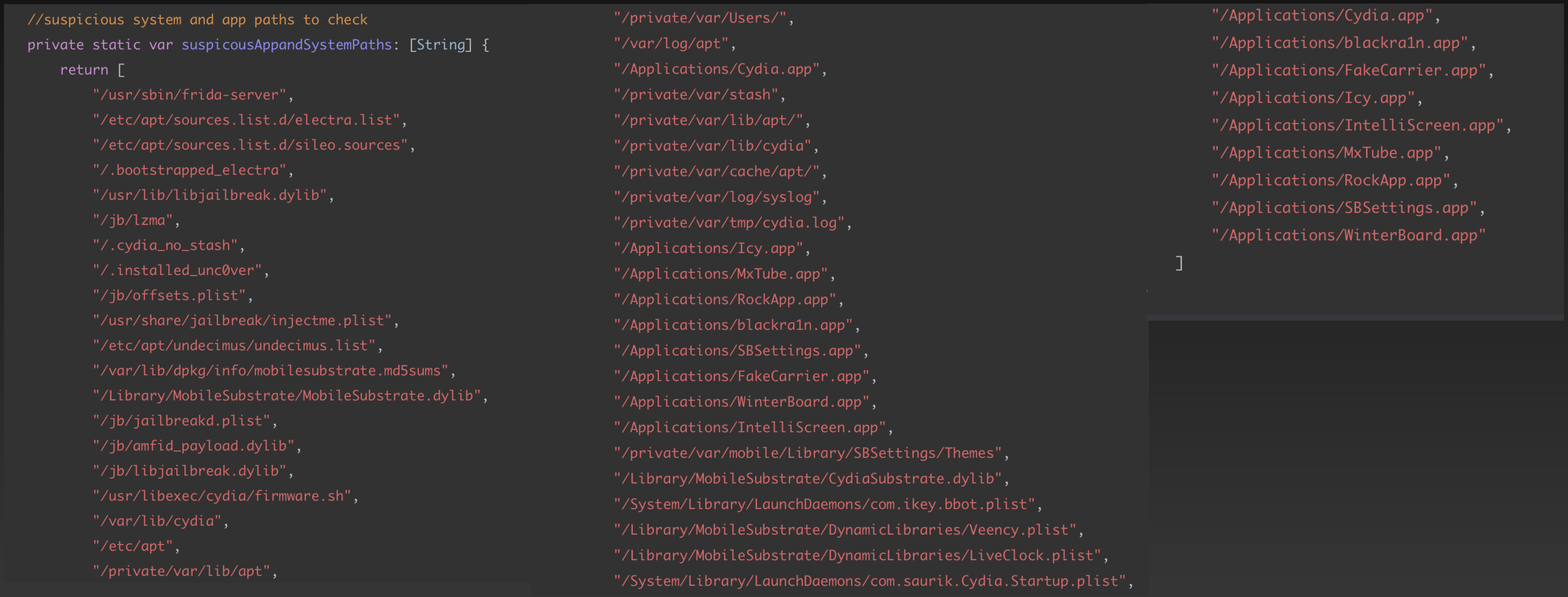
Jailbreak detection mechanisms are added to reverse engineering defence to make running the app on a jailbroken device more difficult.

- Checking for files or directories common to jailbroken devices, such as Cydia
- Checking for elevated directory permissions (i.e. more directories with “write” permission)
- Checking to see if an app can successfully write files outside of its sandbox
- Checking cydia:// protocol handler

They can be accessed, reverse-engineered, and evaded by attackers.

File based checks

Check for files and directories typically associated with jailbreaks, such as:



```swift
//suspicious system and app paths to check
private static var suspicousAppandSystemPaths: [String] {
    return [
        "/usr/sbin/frida-server",
        "/etc/apt/sources.list.d/electra.list",
        "/etc/apt/sources.list.d/sileo.sources",
        "/.bootstrapped_electra",
        "/usr/lib/libjailbreak.dylib",
        "/jb/lzma",
        "/.cydia_no_stash",
        "/.installed_unc0ver",
        "/jb/offsets.plist",
        "/usr/share/jailbreak/injectme.plist",
        "/etc/apt/undecimus/undecimus.list",
        "/var/lib/dpkg/info/mobilesubstrate.md5sums",
        "/Library/MobileSubstrate/MobileSubstrate.dylib",
        "/jb/jailbreakd.plist",
        "/jb/amfid_payload.dylib",
        "/jb/libjailbreak.dylib",
        "/usr/libexec/cydia/firmware.sh",
        "/var/lib/cydia",
        "/etc/apt",
        "/private/var/lib/apt",
```

```swift
        "/private/var/Users/",
        "/var/log/apt",
        "/Applications/Cydia.app",
        "/private/var/stash",
        "/private/var/lib/apt/",
        "/private/var/lib/cydia",
        "/private/var/cache/apt/",
        "/private/var/log/syslog",
        "/private/var/tmp/cydia.log",
        "/Applications/Icy.app",
        "/Applications/MxTube.app",
        "/Applications/RockApp.app",
        "/Applications/blackra1n.app",
        "/Applications/SBSettings.app",
        "/Applications/FakeCarrier.app",
        "/Applications/WinterBoard.app",
        "/Applications/IntelliScreen.app",
        "/private/var/mobile/Library/SBSettings/Themes",
        "/Library/MobileSubstrate/CydiaSubstrate.dylib",
        "/System/Library/LaunchDaemons/com.ikey.bbot.plist",
        "/Library/MobileSubstrate/DynamicLibraries/Veency.plist",
        "/Library/MobileSubstrate/DynamicLibraries/LiveClock.plist",
        "/System/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist",
```

```swift
        "/Applications/Cydia.app",
        "/Applications/blackra1n.app",
        "/Applications/FakeCarrier.app",
        "/Applications/Icy.app",
        "/Applications/IntelliScreen.app",
        "/Applications/MxTube.app",
        "/Applications/RockApp.app",
        "/Applications/SBSettings.app",
        "/Applications/WinterBoard.app"
    ]
```

File based checks

Most often, these are checked using the
`-(BOOL)fileExistsAtPath:(NSString*)`
path method in NSFileManager or
`FileManager.default.fileExists(atPath: path)`

However, there are also applications that use lower-level C functions like fopen(), stat(), or access().

Checking File Permissions

Checking the permissions of specific files and directories on the system.

For example /private directory.

There are different ways of performing these checks such as using NSFileManager and C functions like statfs(), open(), utimes(), stat(), pathconf(), stat64(), fopen().

Checking File Permissions

Swift:

```
do {
    let pathToFileInRestrictedDirectory = "/private/jailbreak.txt"
    try "This is a test.".write(toFile: pathToFileInRestrictedDirectory, atomically: true,
encoding: String.Encoding.utf8)
    try FileManager.default.removeItem(atPath: pathToFileInRestrictedDirectory)
    // Device is jailbroken
} catch {
    // Device is not jailbroken
}
```

Checking File Permissions

Objective-C:

```
NSError *error;
NSString *stringToBeWritten = @"This is a test.";
[stringToBeWritten writeToFile:@"/private/jailbreak.txt" atomically:YES
         encoding:NSUTF8StringEncoding error:&error];
if (error==nil) {
   // Device is jailbroken
   return YES;
} else {
   // Device is not jailbroken
   [[NSFileManager defaultManager] removeItemAtPath:@"/private/jailbreak.txt" error:nil];
}
```

Checking Protocol Handlers

You can check protocol handlers by attempting to open a Cydia URL. The Cydia app store, which practically every jailbreaking tool installs by default, installs the **cydia://** protocol handler.

Swift:

```
if let url = URL(string: "cydia://package/com.example.package"),
UIApplication.shared.canOpenURL(url) {
    // Device is jailbroken
}
```

Objective-C:

```
if([[UIApplication sharedApplication] canOpenURL:[NSURL
URLWithString:@"cydia://package/com.example.package"]]){
    // Device is jailbroken
}
```

Why it's not enough

Better ways of Jailbreak detection

Try to block/detect debuggers

1. PT_DENY_ATTACH

```
ptrace(PT_DENY_ATTACH);
```

2. Try to “kill” its own pid with the 0-signal
3. Check if PTRACE is flagged

```
void try_kill() {
  const int pid = getpid();
  int ret = kill(pid, 0);
}
```

A value of 0 will cause error checking. This can be used to check the validity of pid.

PTRACE detect:

```
inline bool ptrace_detect() {
  int32_t opt[4] = {
    CTL_KERN,
    KERN_PROC,
    KERN_PROC_PID,
    getpid(),
  };
  kinfo_proc info;
  sysctl(opt, 4, &info, sizeof(kinfo_proc), nullptr, 0);
  return info.kp_proc.p_flag & P_TRACED;
}
```

Better ways of Jailbreak detection

Check if the parent pid is launchd

getppid() == 1

Try to detect if the rootfs is writable

getfsstat64(), statvfs()

Try to load an invalid signature

fcntl(F_ADDSIGS)

Check signature state

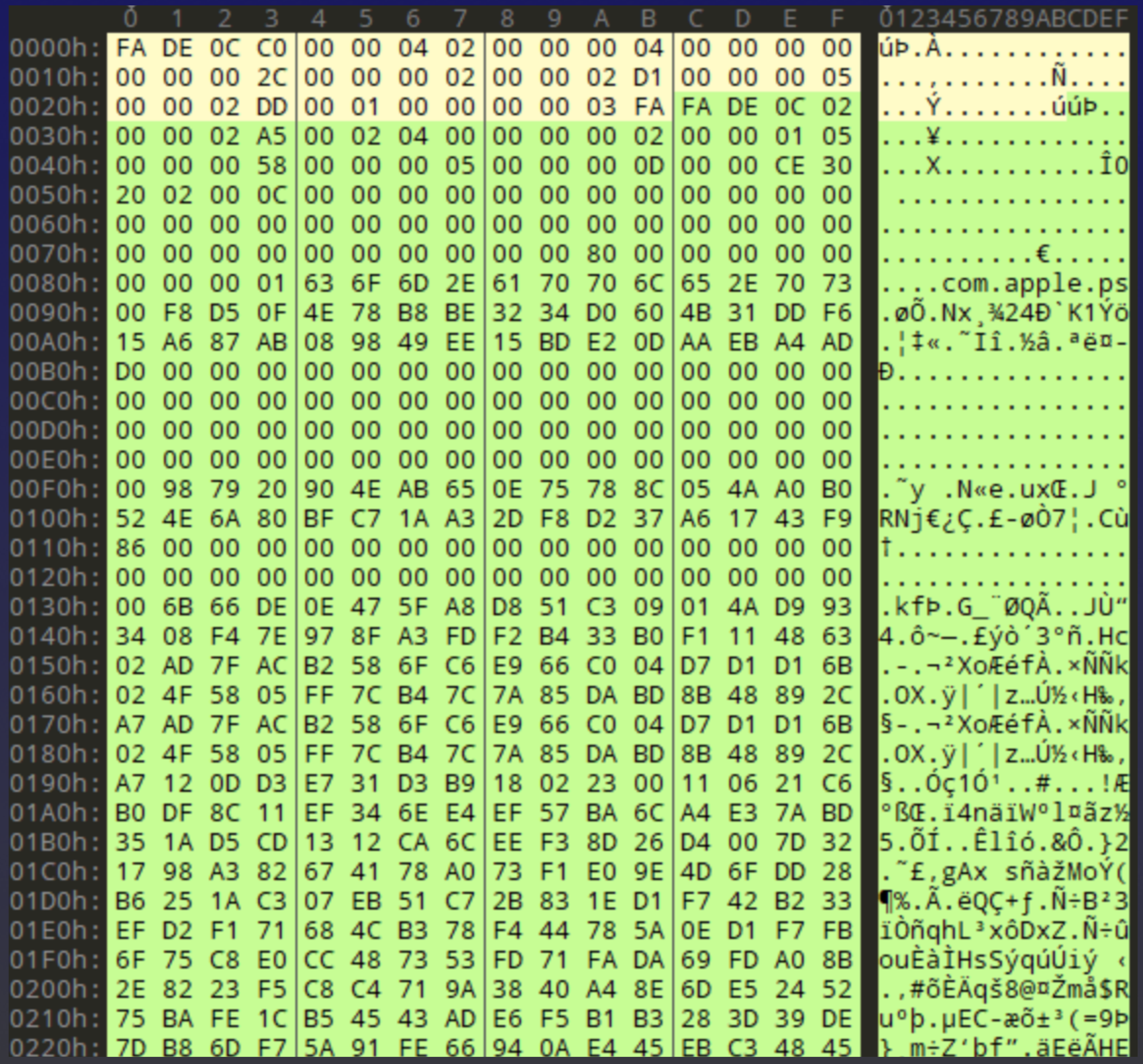
csops(CS_OPS_MARKKILL)

Check signature directly, check code integrity

CRC, derive constants from the code, check API entries, etc.

We can check the integrity directly.

Better ways of Jailbreak detection



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: FA DE 0C C0 00 00 04 02 00 00 00 04 00 00 00 00  úÞ.À............
0010h: 00 00 00 2C 00 00 00 02 00 00 02 D1 00 00 00 05  ...,.......Ñ....
0020h: 00 00 02 DD 00 01 00 00 00 00 03 FA FA DE 0C 02  ...Ý.......úúÞ..
0030h: 00 00 02 A5 00 02 04 00 00 00 00 02 00 00 01 05  ...¥............
0040h: 00 00 00 58 00 00 00 05 00 00 00 0D 00 00 CE 30  ...X..........Î0
0050h: 20 02 00 0C 00 00 00 00 00 00 00 00 00 00 00 00   ...............
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0070h: 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00  ..........€.....
0080h: 00 00 00 01 63 6F 6D 2E 61 70 70 6C 65 2E 70 73  ....com.apple.ps
0090h: 00 F8 D5 0F 4E 78 B8 BE 32 34 D0 60 4B 31 DD F6  .øÕ.Nx¸¾24Ð`K1Ýö
00A0h: 15 A6 87 AB 08 98 49 EE 15 BD E2 0D AA EB A4 AD  .¦‡«.˜Iî.½â.ªë¤-
00B0h: D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ð...............
00C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00F0h: 00 98 79 20 90 4E AB 65 0E 75 78 8C 05 4A A0 B0  .˜y .N«e.uxŒ.J °
0100h: 52 4E 6A 80 BF C7 1A A3 2D F8 D2 37 A6 17 43 F9  RNj€¿Ç.£-øÒ7¦.Cù
0110h: 86 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  †...............
0120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0130h: 00 6B 66 DE 0E 47 5F A8 D8 51 C3 09 01 4A D9 93  .kfÞ.G_¨ØQÃ..JÙ“
0140h: 34 08 F4 7E 97 8F A3 FD F2 B4 33 B0 F1 11 48 63  4.ô~—.£ýò´3°ñ.Hc
0150h: 02 AD 7F AC B2 58 6F C6 E9 66 C0 04 D7 D1 D1 6B  .-.¬²XoÆéfÀ.×ÑÑk
0160h: 02 4F 58 05 FF 7C B4 7C 7A 85 DA BD 8B 48 89 2C  .OX.ÿ|´|z…Ú½‹H‰,
0170h: A7 AD 7F AC B2 58 6F C6 E9 66 C0 04 D7 D1 D1 6B  §-.¬²XoÆéfÀ.×ÑÑk
0180h: 02 4F 58 05 FF 7C B4 7C 7A 85 DA BD 8B 48 89 2C  .OX.ÿ|´|z…Ú½‹H‰,
0190h: A7 12 0D D3 E7 31 D3 B9 18 02 23 00 11 06 21 C6  §..Óç1Ó¹..#...!Æ
01A0h: B0 DF 8C 11 EF 34 6E E4 EF 57 BA 6C A4 E3 7A BD  °ßŒ.ï4näïWºl¤ãz½
01B0h: 35 1A D5 CD 13 12 CA 6C EE F3 8D 26 D4 00 7D 32  5.ÕÍ..Êlîó.&Ô.}2
01C0h: 17 98 A3 82 67 41 78 A0 73 F1 E0 9E 4D 6F DD 28  .˜£‚gAx sñàžMoÝ(
01D0h: B6 25 1A C3 07 EB 51 C7 2B 83 1E D1 F7 42 B2 33  ¶%.Ã.ëQÇ+ƒ.Ñ÷B²3
01E0h: EF D2 F1 71 68 4C B3 78 F4 44 78 5A 0E D1 F7 FB  ïÒñqhL³xôDxZ.Ñ÷û
01F0h: 6F 75 C8 E0 CC 48 73 53 FD 71 FA DA 69 FD A0 8B  ouÈàÌHsSýqúÚiý ‹
0200h: 2E 82 23 F5 C8 C4 71 9A 38 40 A4 8E 6D E5 24 52  .‚#õÈÄqš8@¤Žmå$R
0210h: 75 BA FE 1C B5 45 43 AD E6 F5 B1 B3 28 3D 39 DE  uºþ.µEC-æõ±³(=9Þ
0220h: 7D B8 6D F7 5A 91 FE 66 94 0A E4 45 EB C3 48 45  }¸m÷Z‘þf”.äEëÃHE
```

Check signature directly

We can check the integrity of the signature of our binary. This check starts by opening the main app binary from the disk, seek till kSecCodeMagicEmbeddedSignature sequence **FA DE 0C C0**, read the entitlements and calculate the checksum.

```
enum {
  kSecCodeMagicRequirement = 0xfade0c00,        /* single requirement */
  kSecCodeMagicRequirementSet = 0xfade0c01,     /* requirement set */
  kSecCodeMagicCodeDirectory = 0xfade0c02,      /* CodeDirectory */
  kSecCodeMagicEmbeddedSignature = 0xfade0cc0   /* single-architecture signature */
  kSecCodeMagicDetachedSignature = 0xfade0cc1   /* detached multi-architecture
signature */
  kSecCodeMagicEntitlement = 0xfade7171,        /* entitlement blob */
};
```

Better ways of Jailbreak detection

API-based Detection

fork() - sandboxd does not deny on jailbroken

system(NULL) - returns 1 on jailbroken, because /bin/sh exists.

OpenSSH service detection

Check loopback for 22 (OpenSSH) and 44 (checkra1n) opened ports.

Check if some Jailbreak libraries are loaded in your process

Can use dlopen / memory scanning / dyld internal structures etc.

```
private static func checkDYLD() -> Bool {
    let suspiciousLibraries = [
        "FridaGadget",
        "frida",
        "cynject",
        "libcycript"
    ]
    for libraryIndex in 0..<_dyld_image_count() {

        guard let loadedLibrary = String(validatingUTF8: _dyld_get_image_name(libraryIndex)) else { continue
}
        for suspiciousLibrary in suspiciousLibraries {
            if loadedLibrary.lowercased().contains(suspiciousLibrary.lowercased()) {
                return true
            }
        }
    }
    return false
}
```

Check if your process is instrumented

Try to detect frida

```swift
private static func isFridaRunning() -> Bool {
    func swapBytesIfNeeded(port: in_port_t) -> in_port_t {
        let littleEndian = Int(OSHostByteOrder()) == OSLittleEndian
        return littleEndian ? _OSSwapInt16(port) : port
    }

    var serverAddress = sockaddr_in()
    serverAddress.sin_family = sa_family_t(AF_INET)
    serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1")
    serverAddress.sin_port = swapBytesIfNeeded(port: in_port_t(27042))
    let sock = socket(AF_INET, SOCK_STREAM, 0)

    let result = withUnsafePointer(to: &serverAddress) {
        $0.withMemoryRebound(to: sockaddr.self, capacity: 1) {
            connect(sock, $0, socklen_t(MemoryLayout<sockaddr_in>.stride))
        }
    }
    if result != -1 {
        return true
    }
    return false
}
```

Summary

Thanks

Vadim A. Yegorov
San Francisco, CA, USA

@vadimszzz on Github
vadimszzz@airmail.cc