# iOS反调试方法与反注入

## 一、使用ptrace，防止GDB附加调试

1、直接调用ptrace函数：ptrace(PT\_DENY\_ATTACH, 0, 0, 0);

2、直接使用内联汇编指令：

\_\_asm\_\_ \_\_volatile\_\_

{

"mov r0, #31\n"

"mov r1, #0\n"

"mov r2, #0\n"

"mov r3, #0\n"

"mov ip, #26\n"

"svc #0x80\n"

::

}

## 二、检测是否有调试器存在

通过如下代码段，可以检测到自身是否本GDB所附加：

static bool AmIBeingDebugged(void)

// Returns true if the current process is being debugged (either

// running under the debugger or has a debugger attached post facto).

{

int junk;

int mib[4];

struct kinfo\_proc info;

size\_t size;

// Initialize the flags so that, if sysctl fails for some bizarre

// reason, we get a predictable result.

info.kp\_proc.p\_flag = 0;

// Initialize mib, which tells sysctl the info we want, in this case

// we're looking for information about a specific process ID.

mib[0] = CTL\_KERN;

mib[1] = KERN\_PROC;

mib[2] = KERN\_PROC\_PID;

mib[3] = getpid();

// Call sysctl.

size = sizeof(info);

junk = sysctl(mib, sizeof(mib) / sizeof(\*mib), &info, &size, NULL, 0);

assert(junk == 0);

// We're being debugged if the P\_TRACED flag is set.

return ( (info.kp\_proc.p\_flag & P\_TRACED) != 0 );

}

## 三、捕获int3异常

通过在程序中设置信号处理函数，并主动触发中断，也可以检测是否被调试，先来看代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <stdio.h>  #include <signal.h>  void handler(int signo)  {  printf("You can do something here!\n");  }  int main(int argc, const char \* argv[])  {  signal(SIGTRAP,handler);  \_\_asm\_\_("int3");  printf("Always say: Hello\n");  return 0;  } |

通过主动执行\_\_asm\_\_("int3"),我们触发了一个SIGTRAP的中断信号，然后我们又安插了一个信号处理函数，来捕获这个中断。一旦程序被动态调试，SIGTRAP信号会传给调试器，而不会传给进程，所以我们可以将一些关键跳转信息放在这里。只要被调试器捕获就不会走，那么程序的流程必然会更改。

## 四、CRC校验

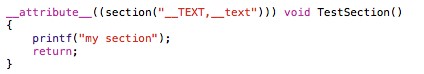
当调试器下断时，必然会将机器指令改为0xcccc等系列，所以我们可以检测代码段的CRC码。

## 五、绕过这些反调试的方法

1、文件补丁。通过llvm-mc可以将每条汇编指令转化为机器码，方便将对应的二进制修改。

2、内存补丁。通过HOOK，可以返回不同的结果。

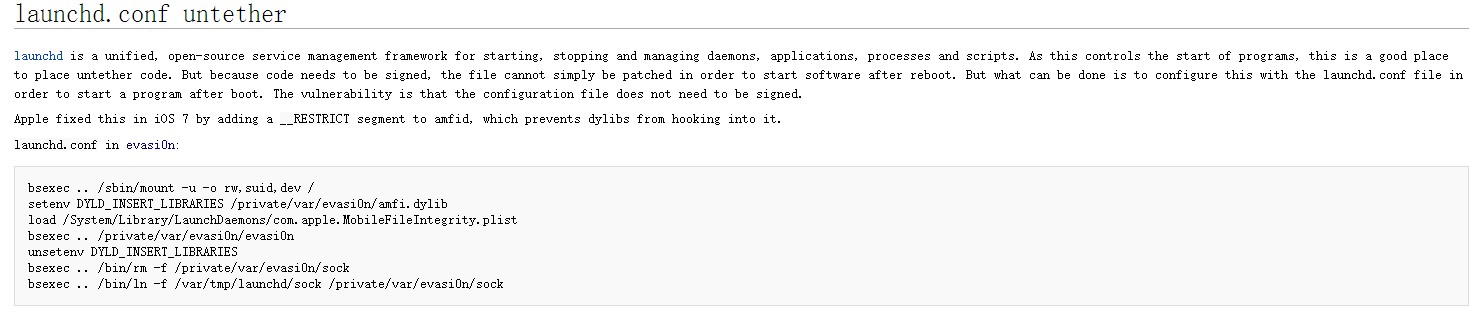
## 六、将代码编译到某个特定的段上



采用编译选项\_\_attribute\_\_，将函数编译到代码段上。

## 七、防注入

DYLIB会被注入，主要是由于DYLD\_INSERT\_LIBRARIES的环境变量所产生，因此，只要该环境变量不存在，则注入就自然失效。如下三种情况下，DYLD\_环境变量会被dyld无视，分别是：  
1. 可执行文件被setuid或setgid了；  
2. 可执行文件含有\_\_RESTRICT/\_\_restrict这个section；



3. 可执行文件被签了某个entitlements。

## 八、反CydiaSubstrate的注入

Cydia Substrate通过注入到launchd进程中去，进而拦截spwan\_fork的函数，并设置环境变量DYLD\_INSERT\_LIBRARIES，以指向/Library/MobileSubstrate/MobileSubstrate.dylib，从而达到注入到每个启动的进程中去。如果我们想达到取消MobileSubstrate.dylib的注入，必须更改这个环境变量，并同时设置\_MSSafeMode=1，这个环境变量。下面分别就几个固件做出详解：

1、iOS6上。在SpringBoard进程中，要启动每个IPA或系统程序，都会带有相应的环境变量，我们只需更改这个环境变量，进而覆盖Cydia Substrate的环境变量，从而达到反注入的目的。SBApplication中的launchSettings函数，返回BKSApplicationLaunchSettings对象，这个对象是包含启动的环境变量，通过更改这个环境变量即可。如下示例： BKSApplicationLaunchSettings st = [SBApplication launchSettings]

NSMutableDictionary\*mut = [st environment]

[mut setObject:@"1" forKey:@"\_MSSafeMode"];

[mut setObject:@"动态库路径" forKey:@" LIBRARIES "]

2、iOS7上. FBApplicationProcess中的initWithApplicationInfo$中的FBApplicationInfo有获取环境变量的方法：environmentVariables，进而可以通过跟1中一样的方法进行设置。