

**未越狱状态下的ios插桩**

**译者：rodster(看雪ID：leixyou)  
原文链接：<http://t.cn/RVta1GJ> 原文作者：Adrian Villa**

** **

**微信公众号：看雪iOS安全小组 我们的微博：weibo.com/pediyiosteam**

**我们的知乎：zhihu.com/people/pediyiosteam**

**加入我们：看雪iOS安全小组成员募集中：http://bbs.pediy.com/showthread.php?t=212949  
[看雪iOS安全小组]置顶向导集合贴： http://bbs.pediy.com/showthread.php?t=212685**

未越狱状态下的IOS插桩

本文描述了插入一个未越狱的ios应用的一个进程。由于在最新IOS版本无法越狱并且要求在最新的IOS版本上测试应用，在未越狱环境下找到评估IOS应用的方法很有必要。在本文中我们会展示配置我们所需要的环境的步骤，重签名IPA文件是为了在环境被配置好并且插入二进制程序来跟踪功能和修改设备的行为之后访问应用信息。

注意：我们将使用的工具是基于MacOS的，因此在整个过程中要求一台Apple MacOS并且安装好Xcode7。

**配置环境**

**创建开发证书**

第一步是在Xcode创建开发者证书。这将允许我们去生成signing identities和供应配置文件。因为Xcode7可以免费创建开发者配置文件。这样做，首先打开Xcode设置窗口然后是如下所示的Accounts标签：

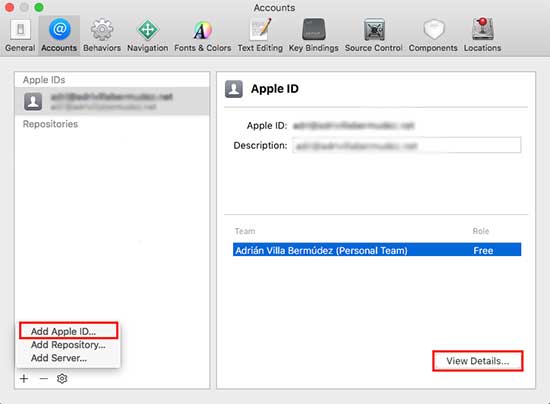


图1 截图所示是我们需要在settings窗口的Accounts标签的点击按钮

一旦我们添加了AppleID，接下去的是从”view details..”选项创建sign identities。本指南的目的仅仅是获得IOS开发所需的signing identity。点击”Create”按钮，在identity创建之后。然后点击”Done”按钮完成操作。

**创建授权文件和配置****开发者配置文件**

一旦开发者配置文件被创建，接下来的一步是生成授权文件和在IOS设备中配置它。一个授权文件在所有IOS应用中都是必须的，它使在运行前的每个应用所做的安全检查都是可信的。例如，检查用来签名app的证书是合法的，检查AppleOD，确认（允许）一个应用程序所请求的权利等等。

我们需要用我们的开发者配置文件生成授权文件，因为重签名过程（见之后）将会用包含我们AppleID和证书的授权文件替代原来的文件。我们将会展示我们所开发的应用所在的设备，设备允许我们访问应用容器、日志等等

创建一个为空的IOS应用工程，修补授权文件问题（如下图所示）并且在设备中配置它。

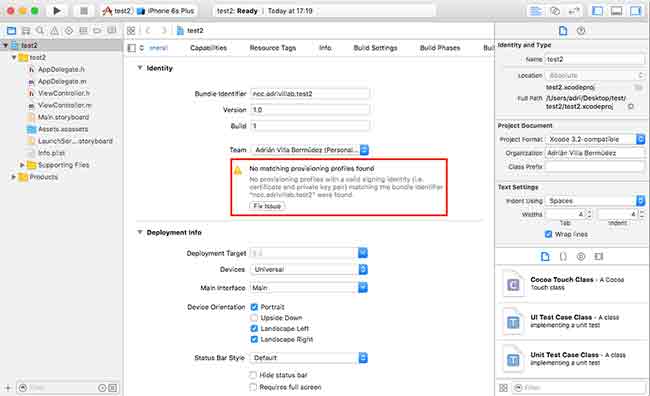


图2 屏幕截图展示授权文件问题

在配置授权后，转到”General settings->Profile”并且接受开发者授权。

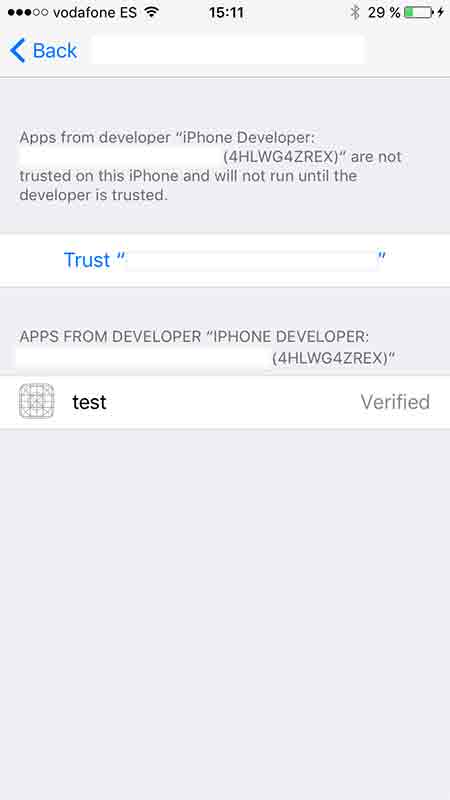


图3 – 屏幕截图所示设备内的开发者授权

为了在已创建的空工程中包含” embedded.mobileprovision”文件，首先找到应用容器，然后从中提取文件。容器的位置能够在Xcode生成的文件中的” Identity and type”菜单可以找到。

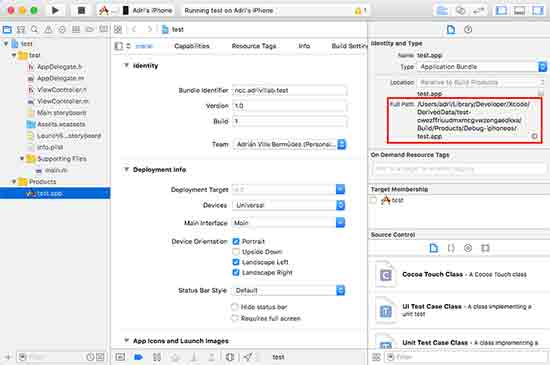


图4 截图所示显示文件路径的Xcode部分

一旦被定位，就可以打开容器文件夹并且复制文件到一个安全的位置

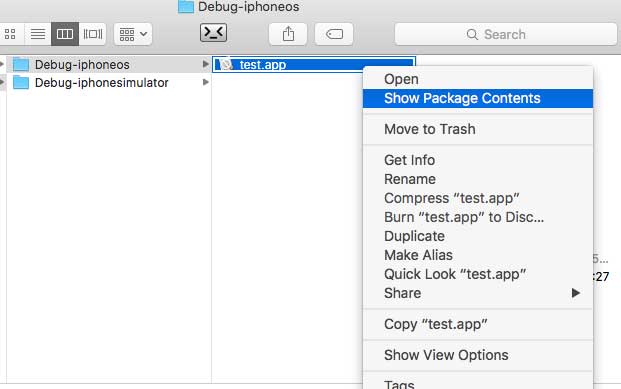


图5-屏幕截图展示的是如何打开一个容器文件夹

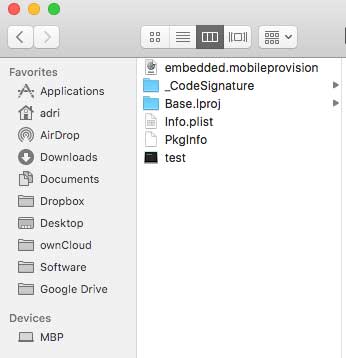


图6 – 截图屏幕所示容器里面的文件，包括授权文件

**1 重签名IPA文件**

IOS支持加载app配置的过程强制用开发者证书签名每一个应用。在应用被配置后，开发者可以访问容器文件夹或者在未越狱的设备上看调试消息日志。因此，为了正确地在未越狱设备上评估一个应用，在安装应用前，有必要在我们的控制下用开发者证书重签名。

重签名的过程涉及几个步骤：

1. 重签名应用的二进制程序和库
2. 插入一个新的授权文件
3. 修改权限和plist文件

为了加快这个过程，可以使用一个叫做node-applesign(<https://github.com/nowsecure/node-applesign>)的工具，有NowSecure的Sergi Alvarez (@pancake)开发。

在首次使用node-applesign之前，需要安装node.js和它的依赖。在这之后，可以从repository复制或者下载并且最终我们执行” ipa-resign.js”文件。使用这个工具重签名一个应用，但这儿有一些要求：

-一个开发者identity hash-id；

-授权文件的路径

-IPA文件的路径

使用node-applesign工具能够获得Identity hash-id，指定-L选项，或者security

$ ./ipa-resign.js -L

$ security find-identity -v -p codesigning

在选择正确的identity之后，使用这个工具重签名IPA。接下来的命令重签名一个IPA文件，替代原来的文件，以便允许我们更容易地在设备中找到我们的应用。

$ ./ipa-resign.js -r -i 23CE9621AA7324F4E89481C6630591603A2ACD5E -m

~/pentest/iOS/myprovision.mobileprovision ~/pentest/iOS/test.ipa

如果很成功地完成所有步骤，就可以使用Devidce Xcode或者使用更好的ios-deploy命令行工具配置应用了。

注意：如果过程中终端显示” Error: This IPA is encrypted”，就需要多做一步，要求使用越狱设备和像Clutch的app解密二进制程序。

**2 二进制插桩**

在这个知识点，在配置前可能需要修改IPA文件以便插入作为库的新资源。因为目标站点Frida (<http://www.frida.re/>)将被用来插入二进制程序性能中并且插入应用。

Frida，文档描述它为“是一个动态代码插桩套件。它让你注入javascript代码片段到Windows，Mac，Linux,IOS和android上的本地app。Frida也提供你一些基于Frida API的简单工具。这些可以被用来帮助和调整你的需求，或者充当如何使用API的例子”。

Frida提供了一个叫做“FridaGadget.dylib”的动态库，可以用来插入在未越狱设备上新的已开发的应用程序。要想做到这个的话，就必须插入一个库作为这个应用的一个新工程的资源。额外地，这也可能是一个已经存在编译过的应用使用这个库，这是我们现在的方案，仅仅通过修改二进制程序和插入一个新的加载命令。

在继续下去之前，有必要至少对IOS应用的二进制结构理解一点点。IOS应用使用了一个叫做Mach-O二进制格式，Match Object文件格式的简称，它是a.out(汇编输出)的替代者。Mach-O被大部分基于Mach内核的系统所使用，像MacOS（先前已知地OSX）和IOS。

Mach-O（IOS二进制格式）二进制结构被划分为几个区域。Header包含关于二进制程序公共信息：byte order (magic number), CPU type, amount of load commands等等。Load commands部分就像一个二进制内容的表单：它描述了段的位置，symbols table, dynamic symbols table等等。每个load command包含了应用程序代码和数据，像symbol tables, dynamic symbol tables等等

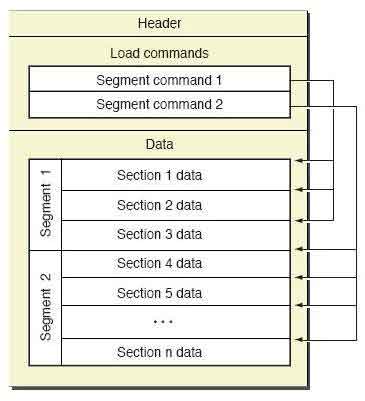


图7 – 屏幕截图展示了Match-O结构

插入一个新库的过程涉及多个步骤：

1. 插入一个库到应用容器
2. 在二进制的load commands区域插入load command
3. 在头部区域增大load command计数器
4. 在头部区域增大二进制数大小

再次为了加快和帮助这个过程，可以使用一个叫做insert\_dylib(<https://github.com/Tyilo/insert_dylib>)的工具。首先，复制或克隆repository，用Xcode打开它并且编译这个工具。现在这个环境和工具都已经准备并且配置好，可以开始二进制程序的插桩。

接下来的步骤需要完善一遍开始插桩：

1.解压IPA文件然后复制应用容器(在二进制程序的同等级目录)中的FridaGadget.dylib文件：

$ cd Payload/Test.app/

$ curl -O https://build.frida.re/frida/ios/lib/FridaGadget.dylib

2.修改二进制程序然后用insert\_dylib工具插入load command。指定strip-codesign选项以确保之后重签名过程运转得很好：

$ ./insert\_dylib --strip-codesig --inplace @executable\_path/FridaGadget.dylib

Payload/Test.app/Test

3.重签名IPA文件。这将要求解压“payload”文件夹到一个IPA文件，然后使用ipa-resign工具。如之前所解释的那样。

4.由于ios-deploy工具的一个特征，再次解压IPA文件很有必要，另外，如果我们配置IPA文件，应用容器反而会跑出一个信号错误。

5.现在，可以在调试模式下使用ios-deploy配置应用程序。应用程序必须在调试模式下执行，因为我们没有被苹果签名。

$ ios-deploy --bundle Payload/Test.app --id

92c8551cb5486d13d35140a597318cb3a3926807 -d

6.一旦应用程序开始运行，动态库就启动Frida服务。等待知道Frida服务开始在调试控制台显示信息。

7.一旦Frida服务在设备上监听，很有可能使用Frida工具箱开始相互影响进程。第一步就是确认gadget是否被正确地配置使用例如frida-ps。

$ frida-ps -Uai

8.在本点FridaGadget在等待一个输入，为插桩留出余地。由于未越狱设备的一个问题，攻破一个新进程并且在新进程上应用插桩。而不是在原进程中应用它。在这个指南中，我们使用在frida-script (https://github.com/as0ler/frida-scripts)项目中可获得的脚本，脚本允许我们跟踪已知的功能。如果我们想，我们可以开发我们自己的脚本，插入具体的功能。

$ python frida-scripts/hooker.py -S re.frida.Gadget

9.在这之后，应用程序的执行代码将被重新开始。这样做，需要使用frida命令，命令允许我们与Frida相互作用。

$ frida -U -f re.frida.Gadget

[USB::iPhone::[re.frida.Gadget]] -> %resume

一旦所有的步骤完成，很有可能看见阿紫hooker.py输出消息上，显示调用了跟踪API和发送给函数的参数。

我希望那你能喜欢这篇文章。我现在正工作与一个工具/脚本上，这会使整个过程变得容易。

我衷心的感谢Sergi Martinez (@zlowram)在技术上支持我激励我。

Adrian Villa (@AdriVillaB)

Adrian.Villa@nccgroup.trust

Published date: 12 October 2016

Written by: Adrian Villa