iOS渗透测试工具之二进制代码分析与调试篇

mss**** / 2019-01-21 09:29:00 / 浏览数 3995

原文：https://www.allysonomalley.com/2019/01/06/ios-pentesting-tools-part-4-binary-analysis-and-debugging/

本文是这个文章系列中的第四篇，也是最后一篇，在本文中，我们将为读者介绍iOS应用程序渗透测试过程中最为有用的一些工具。在本文的上半篇，我们将为读者介绍如何

在本系列文章中，我们将假设用户会使用Electra进行越狱。对于我来说，运行的系统是iOS 11.1.2，不过，本系列文章中介绍的大多数工具都适用于任意版本的iOS 11系统。

Hopper Disassembler

在本教程中，我们将用到Hopper Disassembler。Hopper是一个反编译器和反汇编器，我们可以通过它来查看待破解的应用程序的二进制文件的汇编代码。

读者可以从以下站点下载Hopper：

https://www.hopperapp.com/

虽然专业版提供了二进制文件补丁功能，对于本文来说，免费版本就够用了，因为我们只需要基本的分析和调试功能。

安装好Hopper，我们就可以着手分析目标应用程序了。在第1篇文章中，我们介绍了如何用bfinject对应用程序程序进行解码，并将.ipa/.zip文件下载到了自己的计算机上。现在，我们可以打开Hopper工具，并选择File -> Read Executable To Disassemble选项，然后选择待反汇编的应用程序的二进制文件。请记住，应用程序的二进制文件位于从设备上下载的文件中，即Payload/AppName.app。这个二进制文件

之后，我们需要等待一段时间，因为Hopper进行反汇编是需要一点时间的，具体取决于应用程序的大小和您的计算机的性能。

完成反汇编后，我们会在Hopper窗口底部看到以下内容：

```
> dataflow analysis of procedures in segment __DATA
> dataflow analysis of procedures in segment __LINKEDIT
> dataflow analysis of procedures in segment External Symbols
> Analysis pass 9/10: remaining prologs search
> Analysis pass 10/10: searching contiguous code area
> Last pass done
Background analysis ended in 4157ms
```

如果您以前从未使用过汇编代码，那么可能会对上述内容感到非常困惑。不过，对于那些刚接触汇编的人来说，汇编代码本质上是一种中间格式的代码——它是高级编程语言

运行lldb

lldb是一种功能与gdb类似的调试器，不过，在具体命令方面，两者还是有很大的不同的。

有时，方法中发生的事情是一目了然的；通常来说，通过方法的名称及其返回值类型（具体可以考察转储的头部信息），或者通过浏览其汇编代码，就能搞清楚函数的具体功

要安装lldb，首先要检查手机上是否安装了“debugserver”。为此，请打开SSH，并切换至“developer/usr/bin”。然后，查看“debugserver”二进制文件是否存在。如果没有

1. 打开XCode，然后创建一个新项目
2. 通过USB连接设备后，尝试在设备上构建/运行应用程序。这时，应该在顶部栏中看到“Preparing debugger support for iPhone...”消息。完成该操作后，该设备就会安装debugserver。

接下来，我们需要在Mac上进行一些简单的设置。为此，需要在终端中运行下列命令：

```
iproxy 1337 1337 &
```

注意：如果看到”Command Not Found”消息，说明需要安装iproxy：

```
brew install usbmuxd
```

当然，这里可以使用任何闲置的端口号，不过，一旦选定了端口号，在后续步骤中必须使用同一个端口号。

现在，当手机连接ssh后，我们需要获取目标应用程序的PID。为此，最简单的方法是运行如下所示的命令：

```
ps aux | grep AppName
```

这里所说的PID，就是输出内容中的第一个数字。

接下来，需要在手机上运行下列命令：

```
/electra/jailbreakd_client <PID> 1
```

然后执行：

```
/Developer/usr/bin/debugserver localhost:1337 -a <PID>
```

现在，我们的手机已经准备就绪了，接下来，我们需要在计算机上启动lldb，具体命令如下所示：

```
lldb
```

接下来，我们需要告诉lldb待调试的应用是谁，具体命令如下所示：

```
platform select remote-ios
```

最后，连接到目标应用程序的进程：

```
process connect connect://localhost:1337
```

现在，您应该看到连接成功相关消息，同时，应用程序将暂停执行：

利用lldb进行调试

在我们开始调试应用程序之前，我们还需要解决另一个障碍——应用商店的应用程序几乎都会启用ASLR。所谓ASLR，表示“地址空间布局随机化”。简单来说，这是一种安全

在lldb中，可以运行下列命令：

```
image dump sections AppName
```

这时，将得到如下输出：

```
SectID     Type             Load Address                                 Perm File Off.  File Size  Flags
---------- ---------------- -------------------------------------------- ---- ---------- ---------- ----------
0x00000100 container        [0x0000000000000000-0x0000000100000000)*     ---  0x00000000 0x00000000 0x00000000
0x00000200 container        [0x0000000102b54000-0x00000001043cc000)      r-x  0x00000000 0x01878000 0x00000000
0x00000001 code             [0x0000000102b5a088-0x0000000103f5b364)      r-x  0x00006088 0x014012dc 0x80000400
0x00000002 code             [0x0000000103f5b364-0x0000000103f5fab0)      r-x  0x01407364 0x0000474c 0x80000408
0x00000003 code             [0x0000000103f5fab0-0x0000000103f63fb0)      r-x  0x0140bab0 0x00004500 0x80000400
0x00000004 data-cstr        [0x0000000103f63fb0-0x00000001040b2ac4)      r-x  0x0140ffb0 0x0014eb14 0x00000002
0x00000005 regular          [0x00000001040b2ad0-0x0000000104106de0)      r-x  0x0155ead0 0x00054310 0x00000000
```

我们对这两个突出显示的值非常感兴趣。

要计算偏移量，可以借助十六进制计算器，计算红圈中的数值与篮圈中的数值之差（具体数值见上图）：

```
0x0000000102b54000 - 0x0000000100000000
```

请记下这个结果。对我来说，结果为0x2B5400。这就是我们所需要的偏移量。

现在，选择一个要在其中设置断点的方法。在Hopper中，搜索方法名，并转至其实现代码：

```
        ; ================ B E G I N N I N G   O F   P R O C E D U R E ================


                    -[ProfileHomeViewController didTapLoginButtonOnNotLoggedInView:]:
0000000100a88220            adrp        x8, #0x1018fb000                        ; Objectiv
0000000100a88224            nop
0000000100a88228            ldr         x2, [x8, #0xa40]                        ; qword_va
0000000100a8822c            adrp        x8, #0x101dd0000                        ; &@select
0000000100a88230            ldr         x1, [x8, #0x2a8]                        ; argument
0000000100a88234            movz        x3, #0x0
0000000100a88238            movz        x4, #0x0
0000000100a8823c            movz        x5, #0x0
0000000100a88240            b           imp___stubs__objc_msgSend               ; objc_msg
                          ; endp
```

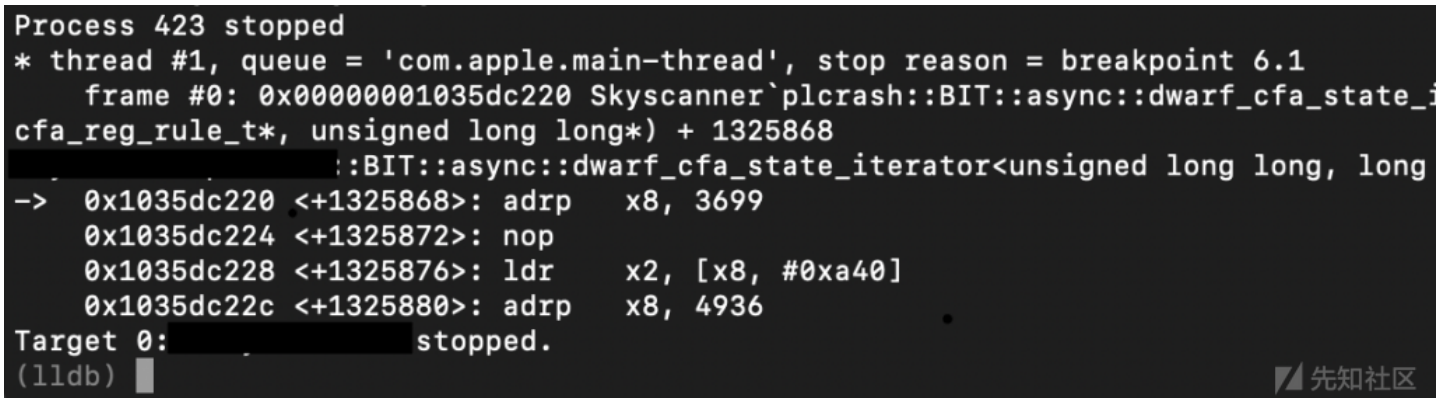
请记下该方法的起始地址。（就这里来说，该地址为0000000100A88220）

现在，我们需要回到lldb中，并通过运行以下命令来设置断点：

```
br s -a 0x2b54000+0x0000000100a88220
```

注意，第一个值是我们计算的偏移量，第二个值是我要调试的方法的入口点。如果您没有看到任何错误消息，说明一切正常。这时，可以键入“c”命令，以继续执行该应用程序

现在，在应用程序中，切换至要调用的方法所在的位置。就本文来说，我选择的是登录按钮。执行该操作时，lldb应在断点处暂停执行：



现在，我们就可以开始调试了！

下面是一些最常用的命令：

`s`

单步进入下一条指令。我们可以重复调用该命令，以监视程序的执行流程。

`c`

继续执行，直到命中下一个断点。

`register read -A`

显示各个寄存器的内容。这对于查看参数、局部变量和返回值来说非常有用。我们可以在每次调用“s”命令之后调用它，以了解每一步中发生了什么事情。

`po $reg`

读取单个寄存器中存储的值。我们可以根据需要，将“reg”替换为所需寄存器的名称。

`register write reg 123`

将新值写入寄存器。该命令对于替换参数、返回值或其他局部变量来说非常有用。

当然，上面介绍的内容，只是lldb丰富功能中的一小部分而已。更多的命令，可以参考下面链接中的命令对照表，它给出了与gdb软件对应的等价命令，这对于熟悉gdb的人

https://lldb.llvm.org/lldb-gdb.html

好了，本文到此结束，感谢大家的阅读！