Oct 7, 2015

iOS App 的逆向工程: Hacking on Lyft

with Conrad Kramer (https://twitter.com/conradev)

(https://twitter.com/intent/tweet?text=iOS%20App%20%E7%9A%84%E9%80%86%E5%90%91%E5%B7%A5%E7%A8%8B%3A%20Hacking%20on%20Lyft%20https%3A%2F%2Frealm.io%2Fcn%2Fnews%2Fconrad-kramer-reverse-engineering-ios-apps-lyft%2F%20via%20%40realm%20%40conradev%20Lyft)

如果你曾经想要知道某一部分代码如何工作，或者很同情某些人程序里的 Bug，你要是有代码，通常可以看看那部分代码。但是，如果没有代码可怎么办？ 在这个演讲里，Conrad 讲到了很多可以逆向 App 的概念和工具，这些方法和工具可以用来 debug 别人的库和你自己的代码。他还展示了逆向 iOS 版本的 Lyft （译者注：Lyft 是美国 Uber 之外的另一款打车软件），并且成功注入代码，探测网络流量，给我们活灵活现地展示逆向的艺术。通过 Conrad 的逆向技术，你也能成功地把 App Store 里所有的 App 的代码都暴露在你面前。

Speaker Deck

Talk by Conrad Kramer

Full Screen

Previous Slide Next Slide
Previous Next

介绍 (0:00)

我是 Conrad Kramer (https://twitter.com/conradev)，工作在 Workflow (https://workflow.is) 的一枚 iOS 工程师，今天要跟大家聊聊 iOS Apps 逆向工程那些事儿。通常来说，逆向工程就是试图只通过最终产物来了解它背后工作原理的过程。在 iOS app 的世界里，这意味着你在没有源代码的前提下，只通过从 App Store 下载的 .app 来发现你想要的信息。

在做逆向工程的时候，会遇到两个痛点：

1. 使用的工具 通常很难用，因为这些工具本身就缺乏文档，而且很不出名。他们总是崩溃出错，而且没人去修复这些错误。另外，花时间找到他们以及用好他们都很难。
2. 一旦你开始用这些工具了，清楚自己的目标也很重要，因为有太多事情可以干了。此外，你还能用工具去逆向苹果自己的框架，尽管他是闭源的。我们可以用这些技术来探索 UIKit 的某个 bug，或者某个 app 里的 bug。

所以逆向的第一步是像自己发问："为什么这个 bug 会发生？他们在 UI 里用了哪些组件？" 比如，如果你很好奇一个 app 里是否用了 collection view 或者 table view"，你其实很容易回答这个问题，看一眼就好了。但是如果你问："他们的 REST API 是什么结构的？"，这个时候逆向工程就能帮你一探究竟。

逆向 Lyft (1:44)

我决定通过一个项目来开始这个主题，目标是：Lyft's iOS app (https://itunes.apple.com/us/app/lyft-taxi-bus-app-alternative/id529379082?mt=8)，它整个都是用 Swift 完成的。首先要做的事是我们了解我们感兴趣的点。假设我对 REST API 很感兴趣，比如我想要在另一个平台或者在 web 上写一个 Lyft 客户端，我可以看看他们的 API 然后产生好奇："他们的 URL scheme 是什么样子的？"，当你想对 Lyft 做深度链接的时候，你就不得不解决这个问题。对 Workflow (https://workflow.is) 来说，相关性就更高了，因为 Workflow (https://workflow.is) 就是一个自动工具，我们集成了很多不同的 app。我总是需要做些类似的事情，来发现一些不公开的私有 URL scheme。

窥视 Lyft (2:48)

从 iTunes 下载下来的 Lyft 是一个 .IPA 文件（其实就是 zip 格式的）。它有很多的元数据，像 Info.plist 文件，资源文件，图片，本地化的字符串以及很多类似的文件。这里面也包含一些可执行文件，那些被编译了的代码就在里面。另外，因为 Lyft 是用 swift 写的，所以还包含很多附加的框架。

Lyft 现在看来是一个完完全全的黑盒子。我们不能直接的看透它，但我们依然有能力从特定的一些角度窥视到。一种方法就是通过监测网络流量来观察他的 API, 同时还能看到有哪些信息被它发送到了 Lyft 的服务器，这些通常对 App 而言是没有什么伤害的。

然后，我们还能 注入代码，真的非常爽。当 app 运行的时候，你可以打探打探这些 app 里被实例化而且存活的对象。当停止运行的时候，我们可以尝试去研究下 .app 是如何被组织的，同时通过特定的视角来看这些代码。

探测网络流量 - Charles 演示 (4:08)

Charles (http://www.charlesproxy.com) 是一个 HTTPS 代理工具，它可以让流量在发往服务器前拦截网络流量。我们可以用这个工具来观察所有从 Lyft app 流向服务器的请求。

Lyft 事实上通过 SSL 加密了它的所有流量。但这些都很好破，通过中间人攻击 (https://zh.wikipedia.org/wiki/%E4%B8%AD%E9%97%B4%E4%BA%BA%E6%94%BB%E5%87%BB)的方法来解决这个问题。其实就是替换了加密证书。

当你在用 Lyft 的时候，Lyft 不断的把你的地理位置发给 Lyft 的服务器。 对车辆的请求和取消等操作会以格式化后的 JSON 在 Charles 中呈现出来。用 Charles，你会看到 app 里和网络交互的所有内容。

向 Lyft 里注入代码 - Cycript 演示 (7:44)

下一个工具真的很赞，叫 Cycript (http://www.cycript.org)。使用这个工具的时候需要一个越狱设备，注入代码到 app 里。Cycript 可以让你看到别人的代码，当然你自己写的 app 也逃不过。这个工具是 Objective-C 和 Javascript 的混合 app，只用输入 Objective-C 的代码到控制台里，就能在 app 运行这些代码。它的交互编程环境（REPL）比 LLDB 的要好很多。尽管它不能中断，设置断点等等，但他对运行时代码非常友好。你可以直接输入下面的代码，基本上都是 Objective-C ：

```
var application = [UIApplication sharedApplication];
[application openURL:[NSURL URLWithString:@"https://google.com"]];
```

我 SSH 进入我的破解过的 iPhone 6 以后，打开 Lyft，运行 Cycript，我可以通过选择功能来获取一个运行时的类实例，比如 view controller 的，或者统计类的实例。作为开发者，要确定设置 ACL 来防止特定 api key 的权限问题。如果 app 能拿到这些 key，那 Cycript 也一定能拿到。

你也可以修改 app 里的 views。比如，用 UIApp.keyWindow.recursiveDescription 通过内存寻址，把 “呼叫车辆” 的按钮变成绿色。

```
var b = new Instance (ADDRESS)
b.backgroundColor = [UIColor greenColor]
```

Cycript 甚至还支持 tab 智能提示。这个功能不但对于逆向很有用，你还可以来测试你自己的 app，比如快速换个颜色测试测试效果什么的。

Q: 用 Crcript 测试自己的 app 的时候，设备一定需要越狱么？

Conrad: 对你自己的 app 的话，并不需要。在 cycript.org (http://www.cycript.org) 官网上，提供了一些如何把这个工具嵌入到你自己的 app 的文档。

解密可执行文件 - *dumpdecrypted* 演示 (11:28)

接下来，我们就要来感受下 app 运行时真正的代码了。这个稍微有些复杂，而且需要设备越狱，我们的演示会比较简单。因为有能力重签名我们的设备，所以苹果会加密商店里所有的 app，防止 app 被大家共享。然而，对于一个越狱设备来说，所有的这些加密的 app 都是可以被解密的。我从其他人那里 fork 了一个 repo，叫 dumpdecrypted (https://github.com/conradev/dumpdecrypted)，原作者写这个是为了导出一个 app 的资源，我 fork 了一份，是为了让它能够支持所有的框架，毕竟现在很多 app 都含有 frameworks。

用它的时候，你只要简单的 clone 到本地，然后执行 `make`，再在你越狱后的设备上对 app 执行一下。我对 Lyft 的 app 执行了这个，很快就解密了所有的文件。你看这些文件的时候，会发现所有的 framework 都以 `.decrypted` 后缀结尾。比较有意思的一些模块是 Lyft，LyftKit 和 LyftSDK，但是我们还发现了它还用了 SocketRocket，Stripe，Pusher，Mixpanel 等等的库。

分析可执行文件 - IDA 演示 (13:47)

要分析我们导出的文件，查看源代码，我们要到 IDA (https://www.hex-rays.com/products/ida/index.shtml)，和 Hopper (http://www.hopperapp.com) 及 class-dump (https://github.com/nygard/class-dump) 类似的一个工具。当在编译一个 app 的时候，XCode 会创建一个由汇编组成的可执行文件。IDA 能很漂亮的输出这些可执行文件里的汇编代码，并且相互连接起来，你因此会看到一个由汇编代码组成的图。尽管 IDA 很贵，但是它的免费版本 (https://www.hex-rays.com/products/ida/support/download_demo.shtml) 已经够用了。除了 64 位以外，基本都有。

要找到 Lyft 的 URL scheme，我们可以在 IDA 里执行一个简单的搜索，关键字是 `openURL` ，在类似的反编译工具中，Objective-C 是相对友好和容易的，因为它对于工作原理是相对透明的，Swift 就相对较差，很多工具还没有很好的支持 Swift，Swift 的汇编代码也更混乱，它的类信息很难被提取出来，不过，多练习练习，看看这些汇编，你会发现一些技巧来捕获你所想找的信息。

使用 IDA 的图表视野，我们发现了 `_TZFV4Lyft15DeepLinkManager13handleOpenURLfMS0_FCSo5NSURLSb`，看起来是个很乱的字符串，然而我们依然可以发现 `LyftDeepLinkManager`, `handleOpenURL`, 和 `NSURL` 这几个关键词。这些看似随机的字符串其实是有解释的，虽然没有文档来说明。如果想要了解这些字符串的含义，可以看看 Mike Ash 的博客里一篇叫做 Friday Q&A

(https://mikeash.com/pyblog/friday-qa-2014-08-15-swift-name-mangling.html) 的文章，这些模糊的东西都有解释。比如 `_T` 意思是这是一个 Swift 的符号，`F` 意思是这是一个函数，`4Lyft` 是一个模块名称，等等。

寻找 Lyft 的 URL Scheme (18:23)

想要从我们所见之中发现 URL scheme，我们就要用开发者的思维来思考。

一个 URL scheme 结构大概是这样的：

```
lyft://action?parameter=value
```

我们现在要去找到 `action` 是什么，`parameter` 可能会是哪些。我首先找到了 `DeepLinkAble` Swift 协议，了解了深度链接的工作原理。通过搜索 `DeepLinkAble` 关键字，我在一个类里发现了诸如 `ride`，`help`，`invite`，`profile` 的请求对象。

我们对如何开始一次打车很好奇，我们可以看一下 Lyft 的 `DeepLinkToRide` 类，看看他是如何工作的。想要看这些，你甚至不需要了解汇编，你只需知道如何搜索和扫描汇编中你需要的信息即可。就好比你即使不会法语，却经常看见到法语，看的足够多次后，你总是能悟出些东西来。很多时候 IDA 展示给我们的完全就是一种外语，其实我第一次尝试逆向 Swift 的时候，就是这样一种感受。

通过大致浏览 `DeepLinkToRide` 的图。我发现了一些不同的字符串，比如 “pickup”，“［latitude］”，“［longtitude］”，“destination” 等等。IDA 还展显示了 ”ridetype“，后面经过测试发现这是个 action，通过查看 “ridetype”，我们发现了 “lyft”, “lyft_line”, “lyft_plus”, 和 “access”，这些都是出行方式。

构造这些不同的 URL 部分，然后测试请求。我发现请求一个出行的 scheme 方式如下：

```
lyft://ridetype
      ?id=lyft_line
      &pickup[latitude]=0
      &pickup[longitude]=0
      &destination[latitude]=0
      &destination[longitude]=0
```

现在，我明白了 Lyft 是如何工作的了，我可以集成到 Workflow 里了。这个过程其实就是二进制分析。尽管这些看起来很复杂，但事实上，只要像个开发者一样思考，其实也那么的难。

Q&A (22:50)

Q: 证书绑定（pinning certificates）能否防止中间人攻击？

Conrad：证书绑定是一个用来防止中间人攻击的方法，在实践中确实很有用，前提是你的手机未越狱。比如 Twitter 就在用证书绑定技术。然而，用逆向工具 Cycript (http://www.cycript.org) 能够轻松破解。AFNetworking (https://github.com/AFNetworking/AFNetworking) 支持证书绑定，只要设置一下 `SSLPinningMode` 这个属性。然而… 我们可以用 Cycript，再把它修改成 `none`，如果你的 iPhone 越狱了，或者被人控制了。所以，并没有一种方案能够彻底防止你的流量被检测，如果没有越狱，倒是一种很好的保护方法。

Q: 是否推荐类似 cocoapods-keys (https://github.com/orta/cocoapods-keys) 这样的工具来混淆字符串？

Conrad: 字符串混淆有以下的好处：

- 如果你用苹果框架里的私有 API，能很容易的躲避过苹果审查的自动扫描工具。
- 如果那个人并没有一部越狱的 iPhone，或者它不知道如何解析混淆，他可能很难找到 app 里的字符串。

然而，你无法永远的把字符串藏起来。比如，在 Cycript 中，你可以轻易地解开混淆。所以字符串混淆也不是没法破的保护方案。

Q: class-dump (https://github.com/nygard/class-dump) 和 dumpdecryped (https://github.com/conradev/dumpdecrypted) 的区别是啥？

Conrad: 他们事实上是不同的工具。*dumpecrypted* 能将一个 App Store 加密的工具解密。而 *class-dump* 能将一个解密后得二进制文件去除他的 Objective-C 接口文件，跟 IDA 有点像，不过很遗憾的是：它不支持 Swift。

Q: 导出 Apple framework 的库，比如：this one (https://github.com/JaviSoto/iOS9-Runtime-Headers)，是不是也是用的 class-dump？

Conrad: 是的，苹果的框架没有加密，可以被轻易地导出所有的class。这意味着那些把苹果的私有interface 放到GitHub上的，你可以轻易地搞定他们。

Q: 逆向工程是否存在法律问题？Lyft 会不会不同意调用他们的私有代码和 api？

Conrad: 从法律上讲，我觉得没有太多问题。我们通常会跟所有的合作伙伴去聊到我们在做的事情。比如：我刚刚发现的这些东西我都会和 Lyft 讨论一下，在他们允许后集成到 Workflow 里。这些技术其实也有很多道德上的考虑。不过话说回来，这些技术也能阻止开发者们胡乱搞，就比如之前逆向 Twitter 发现他们上传用户手机里装的 App 列表到他们的服务器上。所以凡事总有两面，逆向工程只是个工具，善用就好了。

Q: 如何查看苹果框架的反汇编代码？

Conrad: 这个的过程跟我之前展示的一个很像，只不过你不再需要一个越狱后的设备。当你把线插到设备上的时候，iTunes 实际上获取到了设备的符号，你可以在 Xcode 里找到一个大概叫 “device symbol” 文件夹，在 IDA 或者 class-dump 里打开 Apple 的 frameworks，然后展开分析。这些都是没加密的，而且可以直接拿来用。在修复一些 beta 版本的 UIKit 的 bug 的时候很好使。
