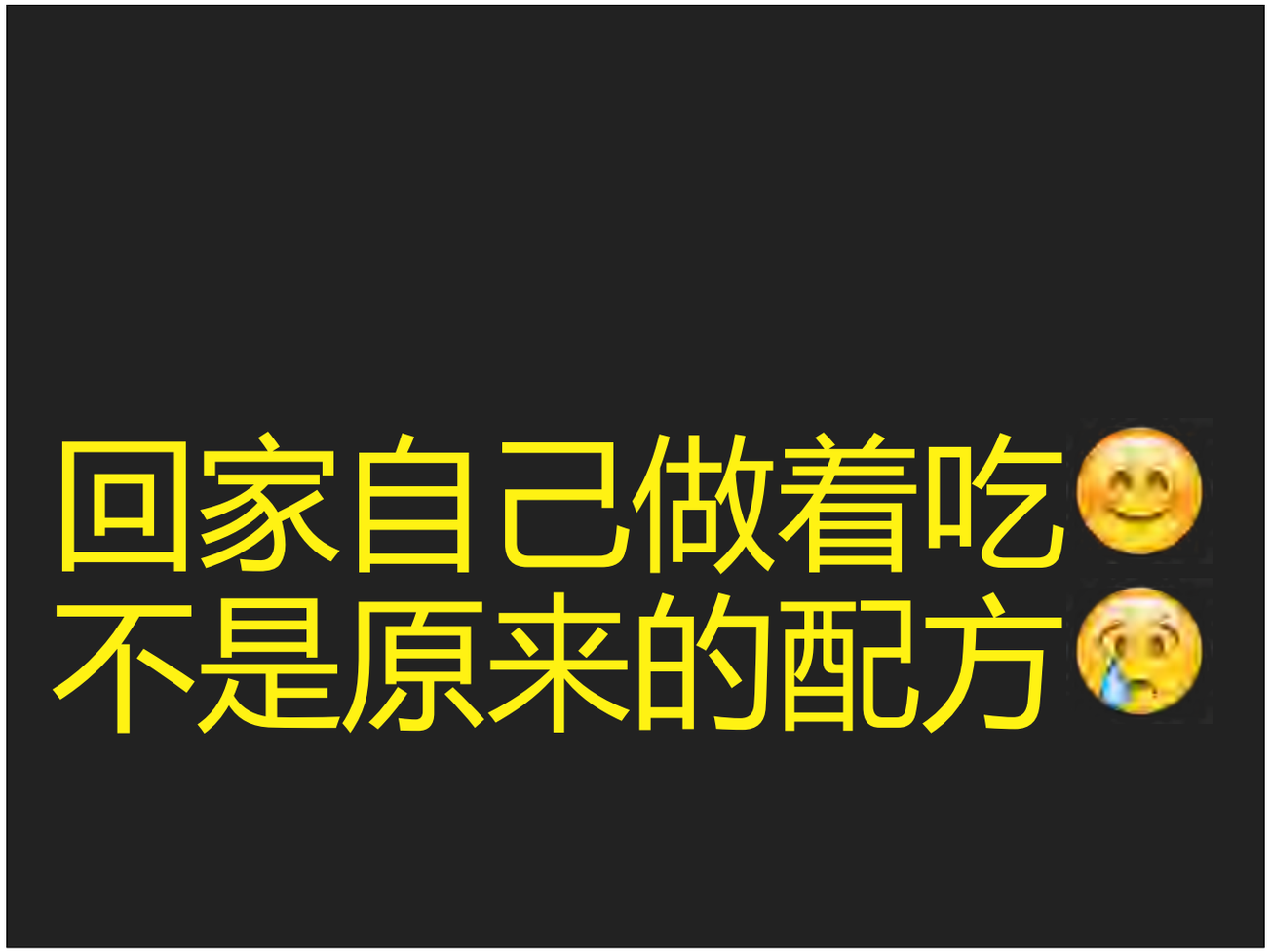
沙梓社

iOS逆向工程一隅

大家好，我是沙梓社，《iOS应用逆向工程》的第一作者。
我的网名叫“大名狗剩”，很多朋友可能对这个称呼更熟悉一些。
我从事iOS逆向工程的学习和工作，已经有6个年头了；在座的估计也有很多低调的高手是这个行业的资深从业人员。毕竟北京是我们国家互联网行业最发达的城市。►

从我懂事后开始算起，这是我第8次来北京。
我还记得第1次来北京玩的时候，是大三升大四的暑假。那个夏天，我住在同学家，白天跟同学一起出去感受北京的风土人情，晚上回到他的家里，他妈妈做饭，大家一起吃，感觉非常好。▶️

有一天，我们准备去爬长城。
阿姨就说，爬长城挺累的，犒劳犒劳你们，晚上给你们包饺子吃。
听说北方不是过年吃饺子，过生日吃饺子，国庆节吃饺子，大大小小的庆祝活动都要吃饺子，所以我那个同学很高兴，非常期待。
但是呢，我的兴致不高。为啥呢?
我的妈妈那边，外公外婆，来自陕北延安那边的穷山沟沟里，穷嘛，饮食就不太讲究，虽然也遵从我们的习俗，逢年过节包饺子吃，但我从来没有觉得家里的饺子好吃过，每次吃饺子我都像完成任务一样，“必须吃15个，吃不完不让出去玩”，所以对饺子的印象不是太好。
我当时心里还想，要不爬完长城在外边吃点再回去得了，后来拗不住面子，就算了。
爬完长城，白天累了一天，晚上吭哧吭哧回去了。
很戏剧的一件事发生了。不知道是饿了还是怎么，觉得阿姨包的饺子太好吃了，毫不夸张地说，是我到目前为止吃过最好吃的饺子。说它的皮儿特别薄呢也不是，说它的肉特别多呢也不是，就是感觉皮儿和馅儿的搭配，口感特别好。▶️

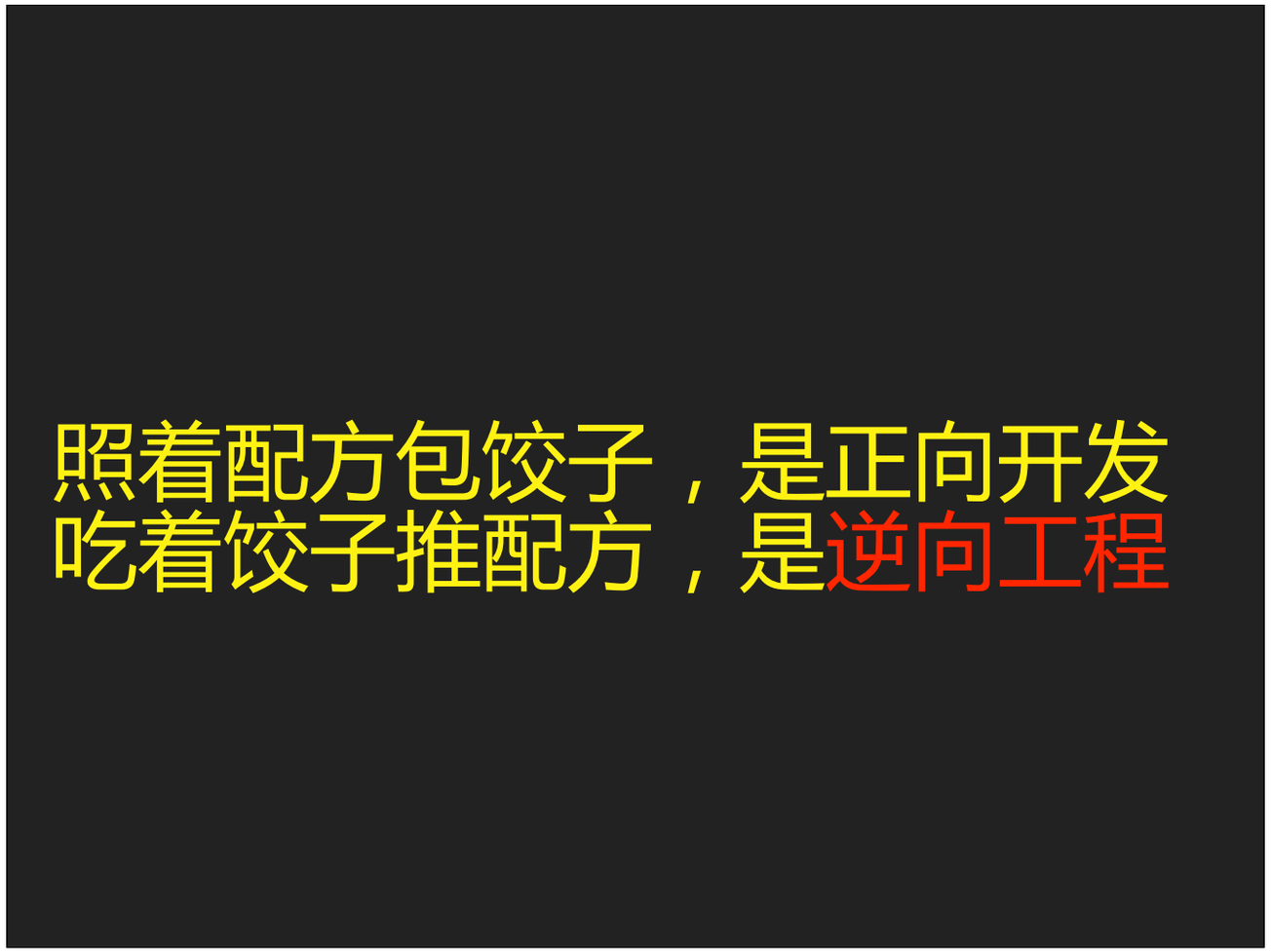


既然这饺子这么好吃，那就想着学两手回去自己做着吃嘛。但是呢▶
不是原来的配方，自然也没有熟悉的味道，自己做的就始终做不出来那个味道，不好吃。那怎么办呢?
如果饺子是朋友做的，就跟我的情况一样，配方呢，是可能可以打听到的;
但是，如果这个饺子是餐馆做的，那餐馆肯定是不会把配方给我的，对吧?
这个是商业机密，人家就指着这个挣钱呢。怎么会给你呢?

好了，在这种情况下，我们吃到了好吃的饺子，想回去自己尝试着包一下，但是拿不到它的配方，怎么办呢?

会包饺子的人，都知道做饺子大概就是这么几步：▶和面、拌饺子馅、包饺子、煮饺子。对不对?
但是呢，虽然说这样能把饺子包出来，能吃，但是好不好吃，影响饺子口感的主要在于配方的细节。和面的时候加不加其他东西？绊馅的时候，放多少茴香，多少肉，煮的时候，煮多长时间，等等。这些细节不搞清楚，我们就不可能做出一模一样的饺子，对吧?

好了，那应该如何搞清这些细节呢？▶️



其实从刚才吃饺子这件事儿里，我们就可以一窥逆向工程的面目。▶读图……

在刚才的场景中，如果我们掌握了逆向工程的技能，就可以达到我们学习饺子配方的目的了。▶️

对于包饺子来说，它的配方是是水、面粉、火候这些东西。►
但是对于iOS逆向工程来说，它的配方内容当然就不是这些东西了，而是API以及它们的调用顺序。►
iOS逆向工程的效果，就是在闭源的环境下，通过还原程序所使用的API及它们的调用顺序，从而倒推出它的实现原理。

也就是说，我们通过品味煮好的饺子，来分析出它在包的时候用了多少面粉，馅里各种材料的比例，煮了多久等等细节，来还原出这个饺子的配方。▶️

解决啥问题？

好了，当我们通过这个饺子的例子，对iOS逆向工程的概念有了初步的认识后，大家可能想知道，它可以解决什么问题呢？►

咱们还是回到饺子上。
在吃完好吃的饺子，酒足饭饱之后，基于饺子，我们可能会有一些其他的想法，比如说：►
如果在和面的时候打个鸡蛋，饺子皮会不会更劲道一些？►
我口味不重，感觉饺子馅有点太咸了，如果少放点盐，饺子口感会不会更好一些？►
有的朋友呢，喜欢吃酸甜口。如果饺子醋里加点白糖，会不会更合我们的胃口呢？▶️

- 增加新的功能模块。
- 分析已有功能模块：
  - 修复bug；
  - 修改定制；
  - 学习提高。
- 换个角度看待自己曾经熟悉的事物。

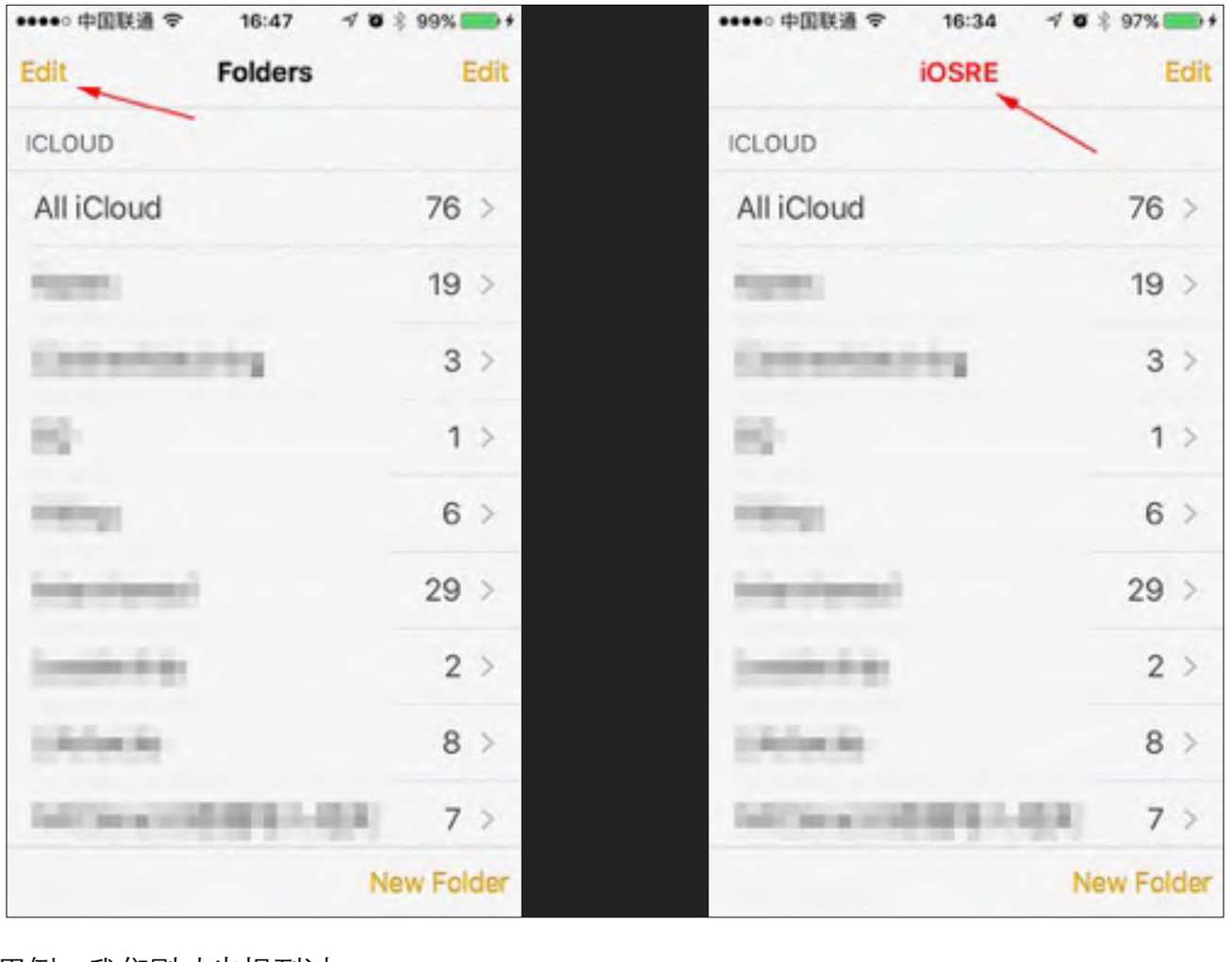
我们把刚才对饺子提出的这些问题，对应到iOS逆向工程中。可以解决的问题主要有：▶️

我们给面粉里打一个鸡蛋，相当于给现有的程序增加新的功能模块▶
我们吃饺子觉得有点咸，或者说觉得饺子醋不够甜，相当于我们分析了现有的功能模块▶

饺子太咸了，少放点盐，就相当于修复bug了；大家如果感兴趣，可以看看论坛上干货分享区我写的一篇文章，就通过逆向工程来修复了QQ国际版iOS客户端的一个bug▶️

我喜欢酸甜口味，给饺子醋里加点糖，相当于我们根据自己的需求来定制、修改现有的功能模块▶
当然，饺子的配方就是我们逆向出来的，相当于我们学习了其他的好产品，来提高自己的水平▶
除此之外呢，我认为最重要的一点，就是逆向工程，能够促使我们从另一个角度来看待自己曾经熟悉的事物，从而激发出非常多的灵感；绝大多数时候，不是做不到，而是没想到。

一旦我们养成了逆向思维的习惯，它就会自然而然地去促使我们去“Think Different”。就我个人来说，这是我学习逆向工程中最大的收获▶️



这里的2张截图，就是iOS逆向工程的2个用例。我们刚才也提到过：

左边的，是“给现有的程序增加新的功能模块”的例子，我给记事本增加了一个新的导航栏按钮；

右边的，是“修改定制已有功能模块”的例子，我把记事本标题的内容和字体颜色都给换掉了。

当然，这2个例子只是证明，逆向工程可以干这些事，至于这些事的意义大不大，那就另说了。

不过呢，以此为出发点，我们可以做的事情就比纯正向开发多了很多，相当于打开了另一扇门。在这扇门后面，哪些事情是有意义的，值得做的，就需要大家发挥各自的聪明才智，来集思广益了。

关于这些细节，欢迎大家到我们的论坛上来讨论；等会儿的讨论环节或者是我的分享结束后，我们也可以进一步地沟通交流。►

咋解决问题？

刚才我们抛出了iOS逆向工程能够解决的问题，接下来我们看看，如何解决这些问题。在我所专注的iOS应用层上，解决逆向工程问题，其实是有比较明显的规律可循的►

- 观察、猜测，寻找分析切入点；

解决iOS应用层的逆向工程问题，一般呢，分为8个步骤。第一步，读文字。
当我们对一个App感兴趣的时候，绝大多数情况下是我们在界面上看到了或者听到了感兴趣的东西，尤其是看到了感兴趣的东西，对吧……简单介绍。
我拿自己日常使用的一个App，Pixlr，来举个例子▶️

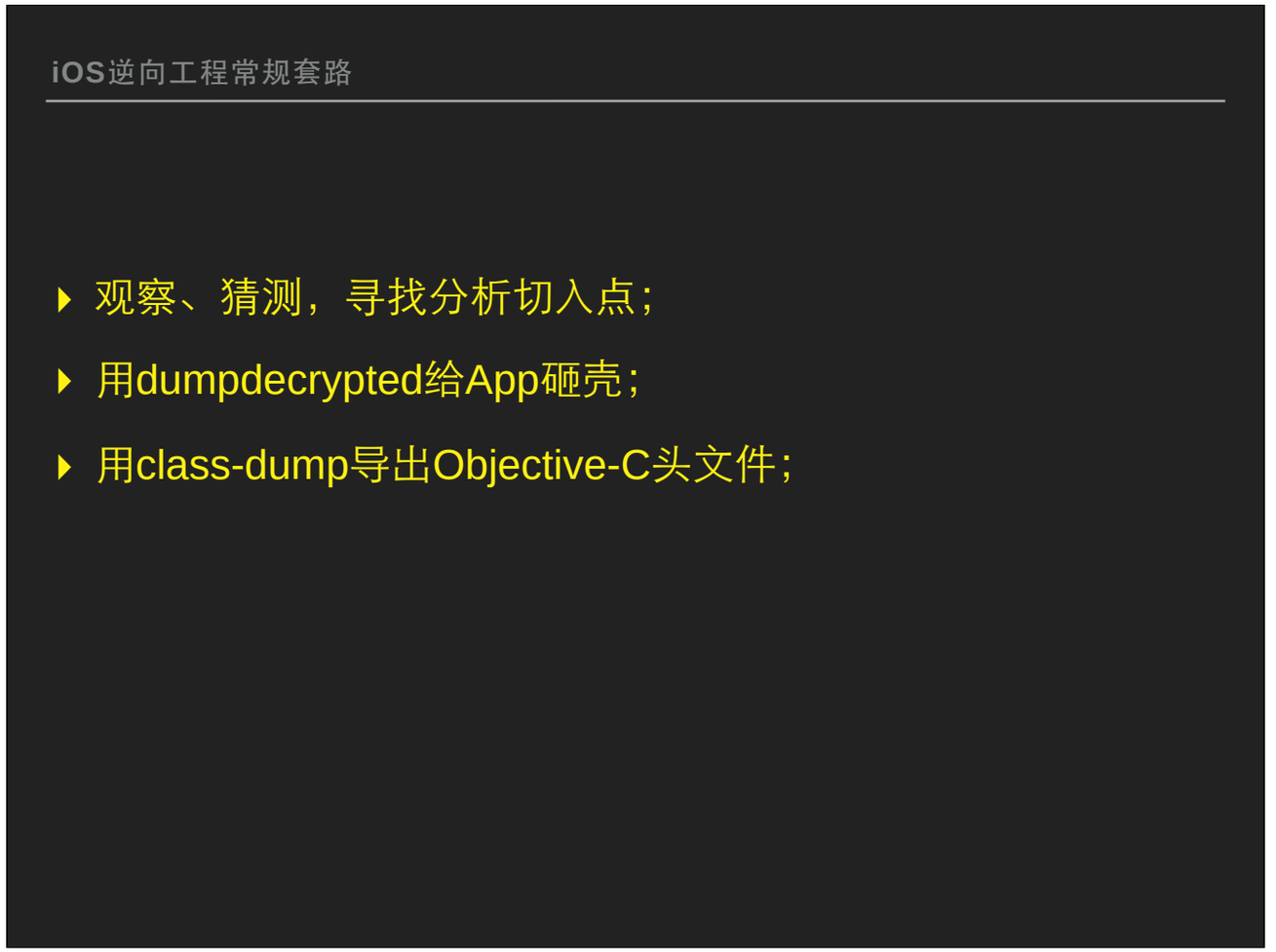
比如，这是一个图片处理软件，我们可以给照片加上各种滤镜，来美化照片►Hagrid这个滤镜的效果，就是右边这2张照片所展示出来的，上面的图在经过滤镜处理后，锐化了很多。我所感兴趣的，就是这个滤镜是怎么做的，它的实现原理是什么。►

- 观察、猜测，寻找分析切入点；
- 用dumpdecrypted给App砸壳；

第二步，读文字。
所有从AppStore下载的App，苹果都会给它加密，用逆向工具分析时，看到的都是密文、乱码。那么想要对App进行分析，就要先把它解密，业界俗称砸壳，用到的工具主要是dumpdecrypted。►

```
FunMaker-SE:~/Containers/Data/Application/1A3596AC-1B07-4772-8C00-C7048DE75BE9/Documents
mobile$ DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/containers/Bundle/Application/
8AC7077C-0479-4494-88B5-0A4E25FAF7D7/PixlrExpressPlus.app/PixlrExpressPlus
mach-o decryption dumper

DISCLAIMER: This tool is only meant for security research purposes, not for application
crackers.

iOSRE: uid = 501, euid = 501, gid = 501, egid = 501.

[+] detected 64bit ARM binary in memory.
[+] offset to cryptid found: @0x100034d98(from 0x100034000) = d98
[+] Found encrypted data at address 00004000 of length 7438336 bytes - type 1.
[+] Opening /private/var/containers/Bundle/Application/8AC7077C-0479-4494-88B5-0A4E25FAF7D7/
PixlrExpressPlus.app/PixlrExpressPlus for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a plain MACH-O image
[+] Opening PixlrExpressPlus.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Setting the LC_ENCRYPTION_INFO->cryptid to 0 at offset d98
[+] Closing original file
[+] Closing dump file
FunMaker-SE:~/Containers/Data/Application/1A3596AC-1B07-4772-8C00-C7048DE75BE9/Documents
mobile$ ls
PixlrExpressPlus.decrypted  dumpdecrypted.dylib
```

砸壳是在命令行下操作的，大概的过程，就是先把二进制文件读到内存中，待解密之后把已经解密的部分拷贝到本地，然后把头部的加密标识给改掉。最后生成的这个后缀是decrypted的文件，就是砸壳之后的明文二进制文件►



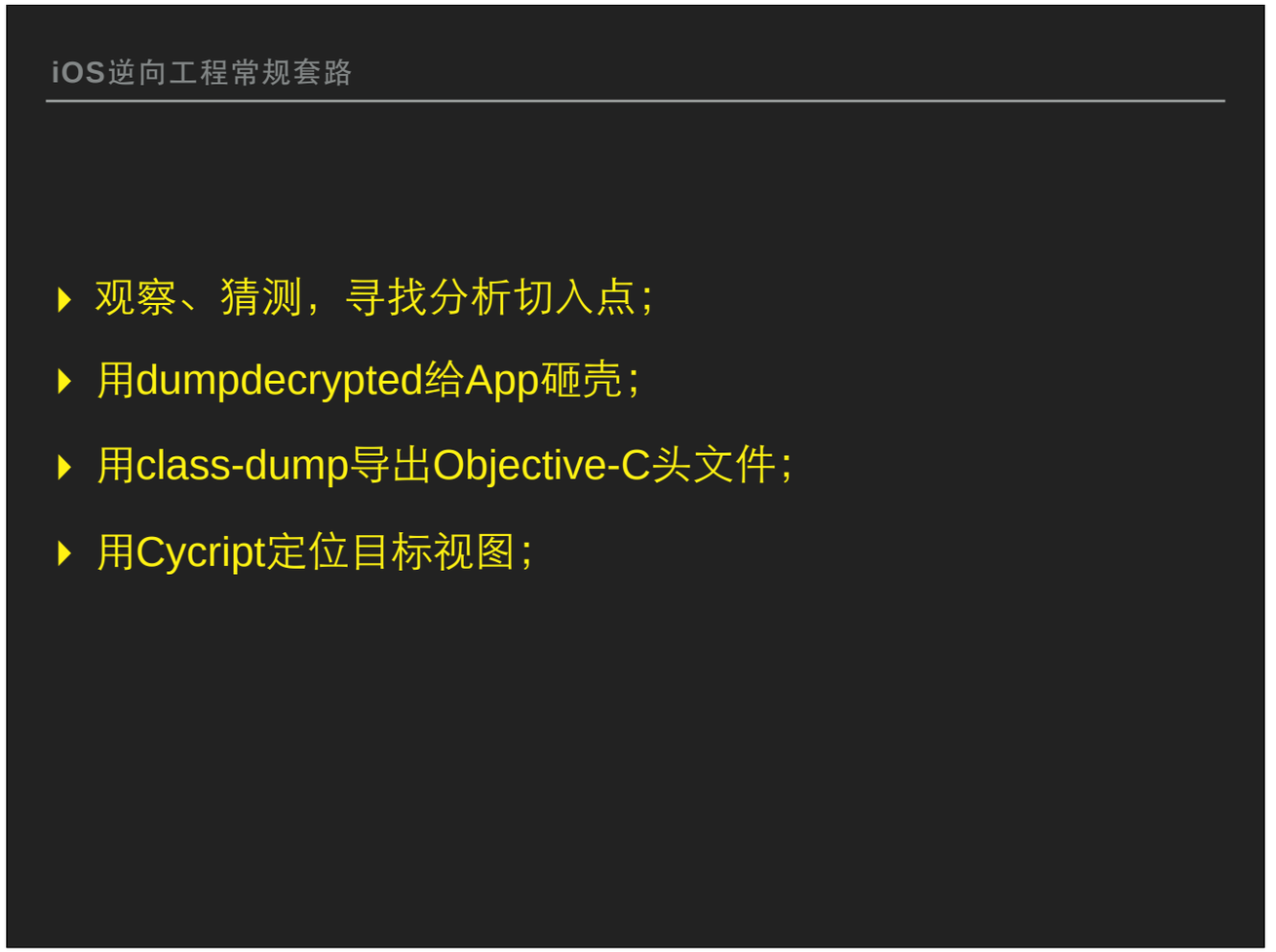
第三步，读文字。
大家应该知道，Objective-C是一门很强调“运行时”的语言，它的很多功能不是在编译时，而是在运行时决定的。要支持这种“运行时”特性，常见的可执行文件和动态链接库都在文件头里存放了很多信息，这些信息可以用来完整地还原一个二进制文件的Objective-C头文件。
这段话说的有点绕啊，我们直接看看效果►

```
FunMaker-AMBP:Pixlr snakeninny$ ls
PixlrExpressPlus.decrypted
FunMaker-AMBP:Pixlr snakeninny$ class-dump -SsH ./PixlrExpressPlus.decrypted -o ./
FunMaker-AMBP:Pixlr snakeninny$ ls
ACGCoverVerticalOverCurrentContextAnimatedTransition.h         IMRenderView.h
ACGDownloadManager.h                IMRenderViewController.h
ACGFeatureManager-Session.h               IMRenderViewDelegate-Protocol.h
ACGFeatureManager.h                 IMRenderViewDismissDelegate-Protocol.h
ACTStub.h                 IMRenderingUtilities.h
ADTEaseBackIn.h                    IMRequest.h
ADTEaseBackInOut.h                 IMRequestBuilder.h
ADTEaseBackOut.h                IMRequestStatus.h
ADTEaseLiner.h                     IMResizeProperties.h
ADTEaseNoParam-Protocol.h                IMResponse.h
ADTEaseParam1-Protocol.h                 IMRichMediaDelegate-Protocol.h
...
IMPublisherProvidedInfo.h                __ARCLiteKeyedSubscripting__-Protocol.h
IMRdbmsDataStore.h                 awManifestManager.h
FunMaker-AMBP:Pixlr snakeninny$
```

这是class-dump之前和之后，目录下文件的对比。dump前，只有一个砸过壳的decrpted文件；dump之后，多了一大堆头文件►

```
//
//     Generated by class-dump 3.5 (64 bit).
//
//     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2013 by Steve Nygard.
//

#import "NSObject.h"

__attribute__((visibility("hidden")))
@interface PXRDevice : NSObject
{
}

+ (_Bool)isIOS8;
+ (_Bool)isIPadMini;
+ (_Bool)isIPhone4_4s;
+ (_Bool)isIPhone5_5s;
+ (_Bool)isIPhone6Plus_6sP;
+ (_Bool)isIPhone6_6s;

@end
```

打开其中一个头文件，跟我们自己写的基本没啥差别，可读性相当高。从这些方法名出发，我们可以从很大程度上了解这个类的作用，给我们接下来的逆向工程提供了相当深入的线索。►



第4步，读文字。

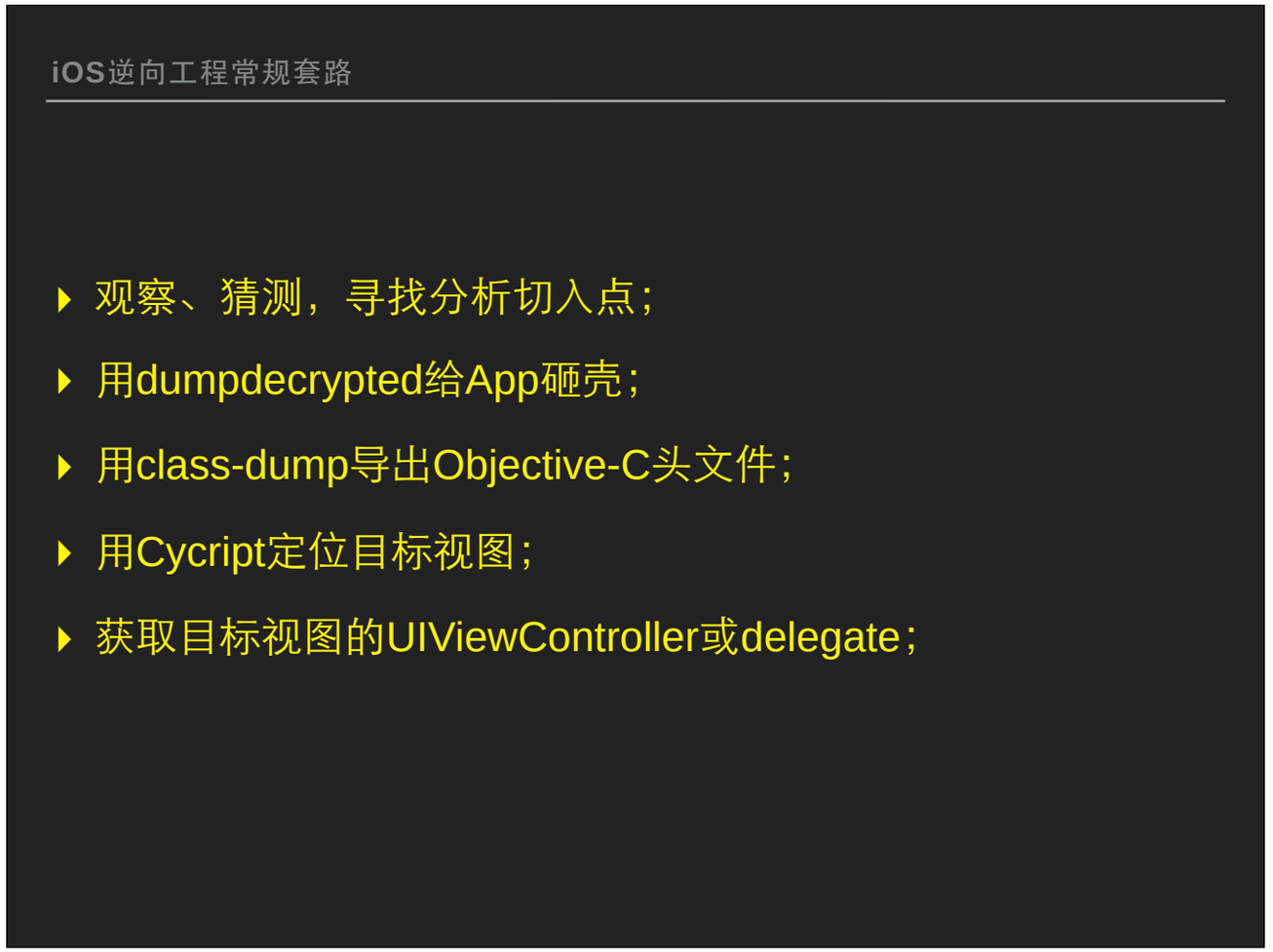
刚才提到，我们感兴趣的，一般是在界面上观察到的现象。从这一步开始，我们就会从界面入手，用Cycript这个工具来定位我们感兴趣界面的类名和各种属性。►

```
FunMaker-SE:~/Containers/Data/Application/1A3596AC-1B07-4772-8C00-C7048DE75BE9/Documents
mobile$ cycript -p PixlrExpressPlus
cy# [[UIApp keyWindow] recursiveDescription].toString()
...
   |    |    |    |    |    |    |    |    | <PXRPackDetailViewCell: 0x13040c0a0;
baseClass = UICollectionViewCell; frame = (215 210; 100 100); clipsToBounds = YES; opaque =
NO; layer = <CALayer: 0x13040bfe0>>
   |    |    |    |    |    |    |    |    |    | <UIView: 0x13040c2d0; frame = (0 0;
100 100); gestureRecognizers = <NSArray: 0x13040de00>; layer = <CALayer: 0x13040c440>>
   |    |    |    |    |    |    |    |    |    |    | <UIImageView: 0x13040c460; frame
= (0 0; 100 100); opaque = NO; autoresize = RM+BM; userInteractionEnabled = NO; layer =
<CALayer: 0x13040c610>>
   |    |    |    |    |    |    |    |    |    |    | <UIView: 0x13040c700; frame = (0
85.5; 100 14.5); autoresize = RM+BM; layer = <CALayer: 0x13040c870>>
   |    |    |    |    |    |    |    |    |    |    |    | <CAGradientLayer:
0x13040e180> (layer)
   |    |    |    |    |    |    |    |    |    |    | <UILabel: 0x13040c940; frame =
(0 85.5; 100 14.5); text = 'Hagrid'; opaque = NO; autoresize = RM+BM; userInteractionEnabled
= NO; layer = <_UILabelLayer: 0x13040cb50>>
   |    |    |    |    |    |    |    |    |    | <UIImageView: 0x130031900; frame = (3
308.5; 594 2.5); alpha = 0; opaque = NO; autoresize = TM; userInteractionEnabled = NO; layer
= <CALayer: 0x12ed1fe70>>
   |    |    |    |    |    |    |    |    |    | <UIImageView: 0x130031ab0; frame = (314.5
510; 2.5 7); alpha = 0; opaque = NO; autoresize = LM; userInteractionEnabled = NO; layer =
<CALayer: 0x130009320>>
...
cy#
```

Cycript，是Cydia之父Saurik开发的一款工具，能够动态地注入其他进程，来方便地运行我们的代码。这段代码展示的，就是我注入了PixIr进程后，打印出当前界面所有UIView的效果。

如果这个界面比较复杂，就会像这个图上一样，信息量非常大。因为我们刚才感兴趣的滤镜是“Hagrid”，►所以一个简单的全文搜索，就可以迅速定位到“Hagrid”所在的控件，是一个UILabel。

同时我们可以看到，这种缩进的显示方式，很明确地展示了不同UIView之间的关系，缩进多的，是缩进少的subview。所以这个UILabel的上层，就是一个PXRPackDetailViewCell，它是UICollectionViewCell的一个子类。►



第5步，读文字。
在MVC设计模式里，View只是一个展板，它不承载具体的业务逻辑或者数据逻辑，对吧。
所以我们的核心其实在这个View对应的Controller和Model上。►

```
cy# [[[UIWindow keyWindow] rootViewController] _printHierarchy].toString()
"<UINavigationController 0x12e0c4a00>, state: appeared, view:
<UILayoutContainerView 0x12df56490>\n   | <LaunchViewController
0x12df48c70>, state: disappeared, view: <UIView 0x12df149a0> not in the
window\n   | <MainEditorViewController 0x12e03c600>, state: appeared, view:
<UIView 0x12f554720>"
```

要拿一个界面的controller，是比较简单的，我们只需要调用这个私有方法，“_printHierarchy”，►然后找当前状态是“appeared”的controller，就可以定位到当前界面的controller。►

```
cy# [#0x13040c0a0 nextResponder]
#"<UICollectionView: 0x12f8d2600; frame = (0 80; 320 314); clipsToBounds
= YES; autoresize = RM+BM; gestureRecognizers = <NSArray: 0x1301bce80>;
layer = <CALayer: 0x1301bb650>; contentOffset: {0, 0}; contentSize: {320,
940}> collection view layout: <UICollectionViewFlowLayout: 0x1301bd710>"
cy# [#0x12f8d2600 delegate]
#"<PXRPackDetailView: 0x1301ba110; frame = (0 0; 320 394); autoresize = W
+H; layer = <CALayer: 0x1301b7d60>>"
```

要找一个view的delegate，也很简单。我先调用一次nextResponder，拿到刚才那个PXRPackDetailViewCell的superview，发现是一个UICollectionView；然后调用一次delegate，就可以拿到它的UICollectionViewDelegate，也就是PXRPackDetailView▶️

- 观察、猜测，寻找分析切入点；
- 用dumpdecrypted给App砸壳；
- 用class-dump导出Objective-C头文件；
- 用Cycript定位目标视图；
- 获取目标视图的UIViewController或delegate；
- 在controller的头文件中寻找蛛丝马迹；

拿到了目标视图的delegate，即PXRPackDetailView之后，就可以在class-dump出的头文件中寻找这个类所在的文件，看看能不能找到什么蛛丝马迹。►

```
//
//     Generated by class-dump 3.5 (64 bit).
//
//     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2013 by Steve Nygard.
//

#import "UIView.h"

@class EffectPack, MenuItem, NSArray, NSLayoutConstraint, PXRProgressIndicator, UIButton, UICollectionView,
UILabel;

__attribute__((visibility("hidden")))
@interface PXRPackDetailView : UIView
{
    _Bool _isSetupMenuItemDownloadObserver;
    MenuItem *_packMenuItem;
    CDUnknownBlockType _effectTappedBlock;
    CDUnknownBlockType _downloadTappedBlock;
    UIView *_topContainerView;
    UILabel *_titleLabel;
    UICollectionView *_colView;
    UIButton *_downloadButton;
    NSLayoutConstraint *_downloadButtonTrailingConstraint;
    PXRProgressIndicator *_downloadProgressIndicator;
    EffectPack *_effectPack;
    NSArray *_effects;
    NSArray *_stylizeEffects;
}

- (void).cxx_destruct;
- (void)awakeFromNib;
@property(nonatomic) __weak UICollectionView *colView; // @synthesize colView=_colView;
- (id)collectionView:(id)arg1 cellForItemAtIndexPath:(id)arg2;
- (void)collectionView:(id)arg1 didHighlightItemAtIndexPath:(id)arg2;
- (void)collectionView:(id)arg1 didSelectItemAtIndexPath:(id)arg2;
…
```

这就是PXRPackDetailView的头文件，各种实例变量和方法名都很清楚。►
点击一个UICollectionView，红框里的这个方法得到调用，滤镜就生效了►

- 观察、猜测，寻找分析切入点；
- 用dumpdecrypted给App砸壳；
- 用class-dump导出Objective-C头文件；
- 用Cycript定位目标视图；
- 获取目标视图的UIViewController或delegate；
- 在controller的头文件中寻找蛛丝马迹；
- 用Hopper和LLDB的组合还原调用逻辑；

接下来，就是逆向工程的重中之重，用Hopper和LLDB，一静一动的组合拳，来还原刚才那个方法的前后逻辑了。►

在Hopper里，我们可以看到这个方法的具体实现，只不过它是以汇编语言的形式呈现给我们的。▶

大家注意这个方法名，从字面意思上，我们可以大概猜到，它就是点击了一个效果cell之后得到调用的block。这个block是怎么实现的呢？▶

```
(lldb) br s -a 0x000000000068000+0x000000010005cc80
Breakpoint 1: where = PixlrExpressPlus`-[PXRPackDetailView collectionView:didSelectItemAtIndexPath:] + 52,
address = 0x00000001000c4c80
Process 4056 stopped
* thread #1: tid = 0x5c863, 0x00000001000c4c80 PixlrExpressPlus`-[PXRPackDetailView
collectionView:didSelectItemAtIndexPath:] + 52, queue = 'com.apple.main-thread', activity = 'starting resolver
activity', 32 messages, stop reason = breakpoint 1.1
    frame #0: 0x00000001000c4c80 PixlrExpressPlus`-[PXRPackDetailView collectionView:didSelectItemAtIndexPath:]
+ 52
PixlrExpressPlus`-[PXRPackDetailView collectionView:didSelectItemAtIndexPath:]:
->  0x1000c4c80 <+52>: bl     0x100608a00               ; symbol stub for: objc_msgSend
    0x1000c4c84 <+56>: mov    x29, x29
    0x1000c4c88 <+60>: bl     0x100608a60               ; symbol stub for: objc_retainAutoreleasedReturnValue
    0x1000c4c8c <+64>: mov    x22, x0
(lldb) p (char *)$x1
(char *) $0 = 0x00000001006d94d7 "effectTappedBlock"
(lldb) ni
Process 4056 stopped
* thread #1: tid = 0x5c863, 0x00000001000c4c84 PixlrExpressPlus`-[PXRPackDetailView
collectionView:didSelectItemAtIndexPath:] + 56, queue = 'com.apple.main-thread', activity = 'starting resolver
activity', 32 messages, stop reason = instruction step over
    frame #0: 0x00000001000c4c84 PixlrExpressPlus`-[PXRPackDetailView collectionView:didSelectItemAtIndexPath:]
+ 56
PixlrExpressPlus`-[PXRPackDetailView collectionView:didSelectItemAtIndexPath:]:
->  0x1000c4c84 <+56>: mov    x29, x29
    0x1000c4c88 <+60>: bl     0x100608a60               ; symbol stub for: objc_retainAutoreleasedReturnValue
    0x1000c4c8c <+64>: mov    x22, x0
    0x1000c4c90 <+68>: cbz    x22, 0x1000c4d14          ; <+200>
(lldb) po $x0
<__NSMallocBlock__: 0x12fe04930>

(lldb) memory read --size 8 --format x 0x12fe04930
0x12fe04930: 0x000000019f449658 0x00000000c3000002
0x12fe04940: 0x0000000100088b74 0x0000000100787e00
0x12fe04950: 0x000000012e716ed0 0x000000012f000400
0x12fe04960: 0x000001a19f44bf19 0x0000000100000788
(lldb)
```

这就轮到LLDB出马了。LLDB就是大家在Xcode里写代码所用到的动态调试器，应该都不陌生，只不过在Xcode调试时，一般不会深入到汇编这一层。
这里我就不展开了，感兴趣的朋友回看一下幻灯片，有什么问题我们在私下交流，好吧！►

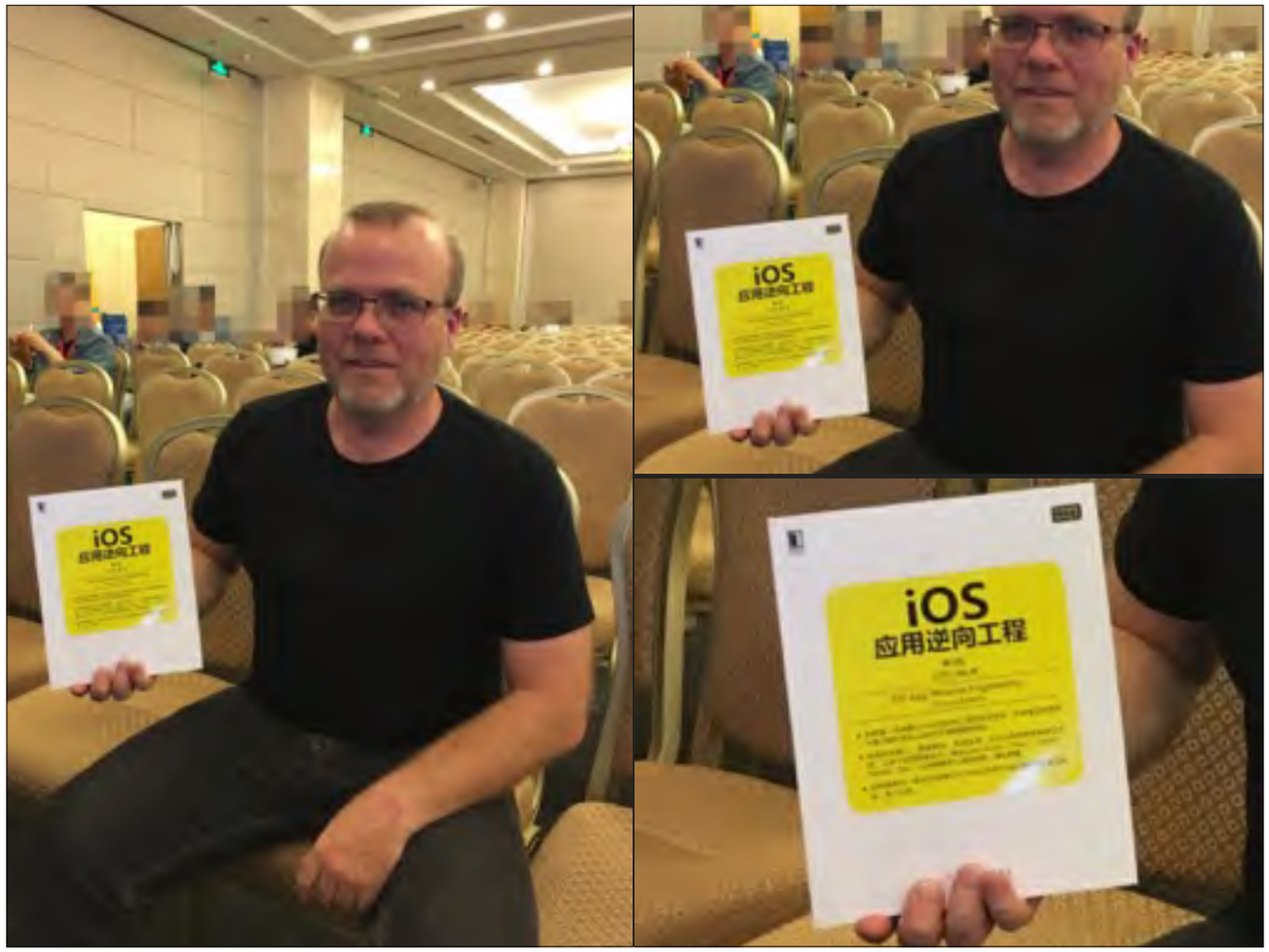
iOS逆向工程常规套路

- 观察、猜测，寻找分析切入点；
- 用dumpdecrypted给App砸壳；
- 用class-dump导出Objective-C头文件；
- 用Cycript定位目标视图；
- 获取目标视图的UIViewController或delegate；
- 在controller的头文件中寻找蛛丝马迹；
- 用Hopper和LLDB的组合还原调用逻辑；
- 用Theos编写插件。

最后一步，就是写代码了。逆向工程相关的开发，一般会用到Theos，用它可以很方便地修改第三方进程的功能逻辑，达到我们自己定制化的目的。►

```
/* How to Hook with Logos
Hooks are written with syntax similar to that of an Objective-C @implementation.
You don't need to #include <substrate.h>, it will be done automatically, as will
the generation of a class list and an automatic constructor.

%hook ClassName

// Hooking a class method
+ (id)sharedInstance {
   return %orig;
}

// Hooking an instance method with an argument.
- (void)messageName:(int)argument {
   %log; // Write a message about this call, including its class, name and arguments, to the system log.

   %orig; // Call through to the original function with its original arguments.
   %orig(nil); // Call through to the original function with a custom argument.

   // If you use %orig(), you MUST supply all arguments (except for self and _cmd, the automatically generated
ones.)
}

// Hooking an instance method with no arguments.
- (id)noArguments {
   %log;
   id awesome = %orig;
   [awesome doSomethingElse];

   return awesome;
}

// Always make sure you clean up after yourself; Not doing so could have grave consequences!
%end
*/
```

这些是Theos的基本语法，非常简单，大家一看就懂了，几乎不需要后面这些注释。我也不展开了▶



刚才那些步骤的细节，在书中都有详细的介绍，如果大家感兴趣，可以买一本看看，外面机械工业出版社的摊位就有卖的；不买，光看看也行。等会1点的时候，我会在那里接待大家，感兴趣的朋友可以过来随便聊聊天。►

有人会问，这哥们谁啊?

图中这位大哥，叫Rasmus，是PHP这门语言的创始人；这张照片是我在今年5月份DevLink，也就是我们的主办方，举办的的PHP开发者大会上给他照的。

我作为Rasmus的随身翻译，全程参与了这个活动。他给我印象最深的，是他对PHP的看法►

他认为，PHP is a hammer，也就是一把锤子。对于大家来说，重要的不是锤子本身，不是去比谁的锤子更重，材料更好，而是我们用这把锤子去做什么，我们怎么样用它，让这个社会，让这个国家变得更好。他把这个问题留给了所有PHP的用户，然后就退居二线了，所以今天我们程序员界流传着这样一句话：“PHP是世界上最好的语言”。
那么作为iOS逆向工程的推广人，我也斗胆模仿Rasmus，将iOS逆向工程定义为一把锤子。►
大家用这把锤子去做什么呢？有人用它来做黑灰产，发了笔横财；有人用它来做安全研究，把iOS给越狱了；有人学习了它的思维方式，让自己的事业更上一层楼。我想，对逆向工程的运用也要因人而异，所以，请大家尽情发挥想象力吧！我就不再举例限制大家的思维了。
希望大家都能用这把锤子，敲碎自己职业的发展瓶颈，打造人生发展的光辉前程。►

我今天的分享就到这里，谢谢大家！
这个二维码是我们的官方论坛，大家如果有任何技术相关的问题，都可以来这里畅所欲言。
接下来的时间，我们随便聊聊吧；技术、人生、理想，什么话题都可以▶

交流讨论

（本环节结束后）谢谢！►

后面还有一张二维码大图