# Chapter 9 Practice 3: Save and share Sight in WeChat

## 9.1 WeChat

WeChat is one of the most outstanding IM App in the mobile Internet industry. In China, it is the daily necessity of most netizens. WeChat’s launch image is as shown in figure 9-1; it seems that there is a little sorrow in its great power.



Figure 9-1 Launch image of WeChat

In October 3rd, 2014, WeChat has updated to version 6.0 and added a new feature, Sight i.e. short videos. It was so fun that my WeChat friends started to share all kinds of Sights, as shown in figure 9-2.

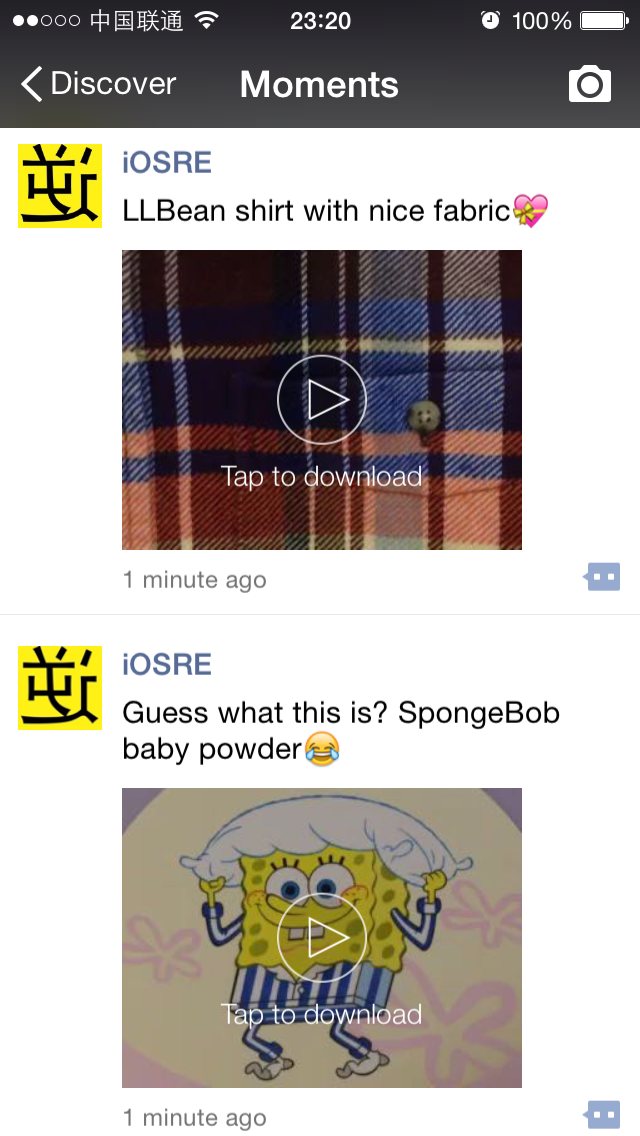


Figure 9-2 Sight

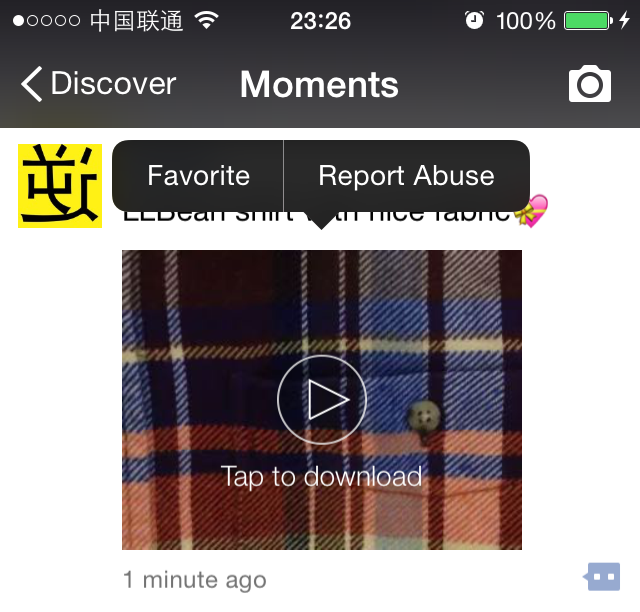


Figure 9-3 Menu of Sight

Although we can already mark our interested Sights via long press menus (as shown in figure 9-3), I’m not satisfied yet; it’d be better if those Sights can be downloaded or shared on other platforms. So, the goal of this chapter is adding two options to long press menu of Sight, which are “Save to Disk” and “Copy URL” respectively.

The size of WeChat 6.0 is bigger than 80 MB; it’s rather complicated reversing it. As usual, before reversing, we need to analyze and modeling the target, then make a plan and carry it out. The following operations are done on WeChat 6.0 on iOS 8.1, iPhone 5. After the publication of this book, WeChat will probably update to a higher version, there will be some tiny changes in the following operations, but the general ideas stay the same. For the analysis of the latest WeChat, please keep following <http://bbs.iosre.com>.

## 9.2 Tweak prototyping

### 9.2.1 Observe Sight view and look for cut-in points

First, switch Sights’ autoplay in “WeChat” → “Me” → “Settings” → “General” → “Sights in Moments” to “Never”, as shown in figure 9-4.

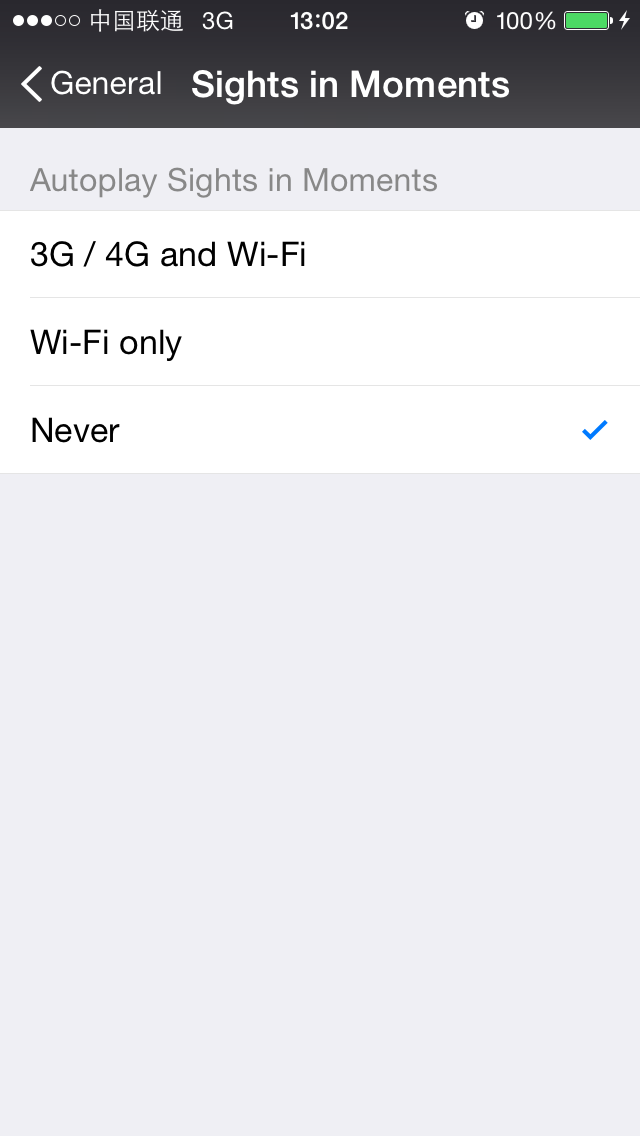


Figure 9-4 Never autoplay Sights in Moments

Let’s review figure 9-3 and think together: “Favorite” and “Report Abuse” will pop up after we long press the Sight view. Doesn’t this phenomenon indicate that the Sight view can already respond to long press gestures? So, we only need to find the gesture action selector and hook it, then we can pop up our custom menu with options “Save to Disk” and “Copy URL” just inside this function.

There is a line of words “Tap to download” under the play button in Sight view, which means WeChat will download the Sight to iOS first, and then play it offline. Therefore, we can conclude that a download URL already exists in a Sight, and the downloaded Sight is saved somewhere on iOS. Luckily, the URL and the downloaded Sight happen to be our goal of this chapter, if we can find their locations in WeChat, our job is done. After the previous 2 practices, I believe your understanding of MVC has became deeper: If we manage to get the V of a Sight, we can get its M, which contains the Sight’s download URL and video objects.

OK, now we know that WeChat has already invented the wheel, we just need to find and make use of it. In order to speed up our reversing process, we won’t be overly sticking to the execution logic of WeChat with IDA or LLDB, but try our best to look for clues in class-dump headers, and then verify our guesses to reach the goal of locating the Sight.

### 9.2.2 Get WeChat headers using class-dump

First decrypt WeChat with dumpdecrypted, which is explained in details in chapter 4. It is worth mentioning that the executable name of WeChat is not “WeiXin” (which is Chinese pinyin) or “WeChat”, but “MicroMessenger”. After we get MicroMessenger.decrypted, drag and drop it to IDA before continuing. Then use class-dump to export its headers.

snakeninnysiMac:~ snakeninny$ class-dump –S –s -H ~/MicroMessenger -o ~/header6.0

After executing the above command, 5225 headers are generated, as shown in figure 9-5.

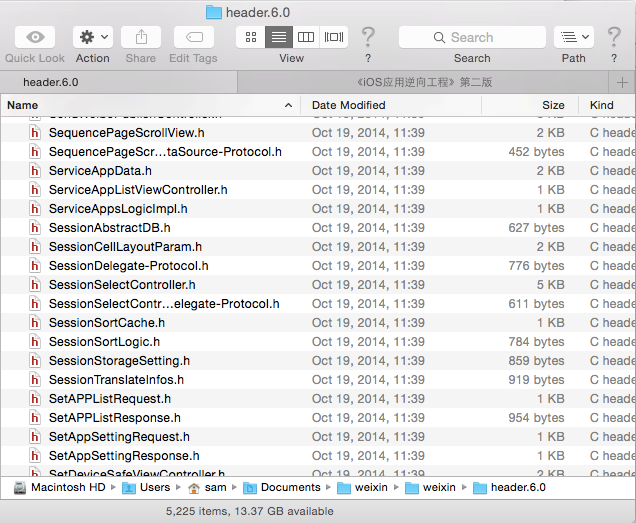


Figure 9-5 Headers of WeChat 6.0

WeChat has the most headers among all Apps I have ever seen, going through all these files one by one is mission impossible for a single person. Such a big project is unlikely to be completed by one single team, perhaps Tencent just splits WeChat into several subprojects, for example, Moments subproject, IM subproject, drift bottle subproject, Sight subproject, etc. For each subproject, there’s one team in charge. At last, all subprojects are merged into one big project, namely WeChat.

### 9.2.3 Import WeChat headers into Xcode

Import all WeChat headers to an empty Xcode project, as shown in figure 9-6.

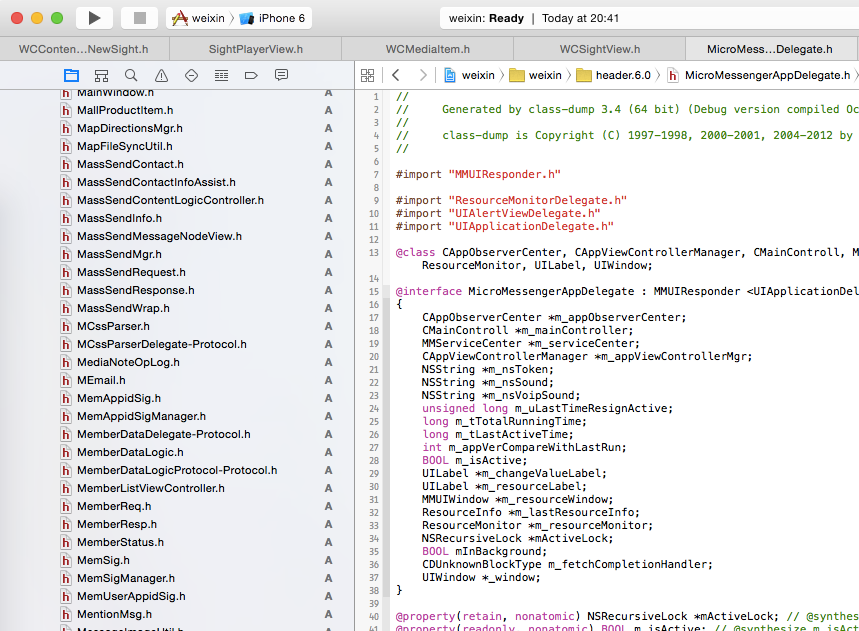


Figure 9-6 WeChat headers in Xcode

The built-in search and highlight functions in Xcode help display the code beautifully and neatly. Now, let’s cut into the code via WeChat’s UI.

### 9.2.4 Locate the Sight view using Reveal

There is no need to introduce how to configure Reveal again. Launch WeChat and enter Moments to find a Sight, then use Reveal to see the view hierarchy, as shown in figure 9-7.

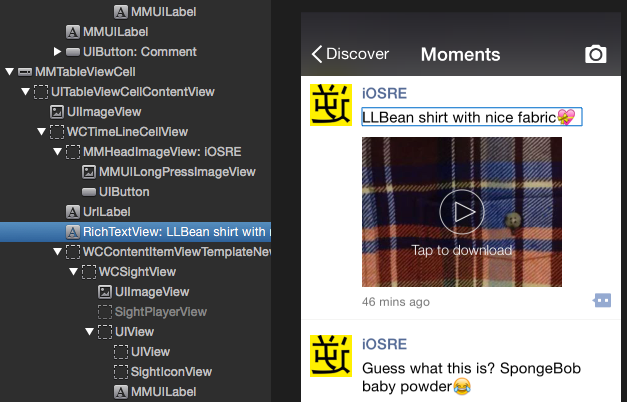


Figure 9-7 Use Reveal to see the UI layout of WeChat

In figure 9-7, text “LLBean shirt with nice fabric” can be discovered easily in both sides, indicating their correspondence. Keep checking around RichTextView, the Sight view is very conspicuous, as shown in figure 9-8.

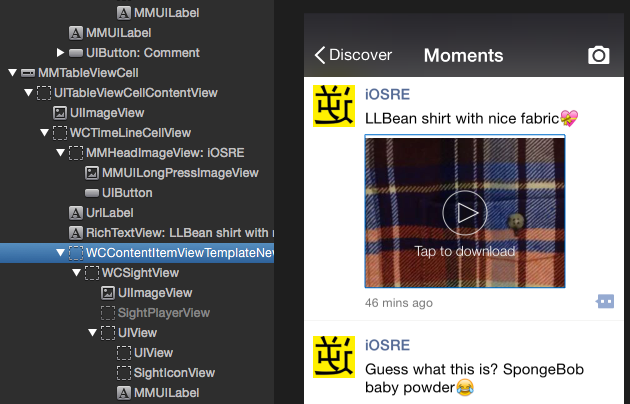


Figure 9-8 Locate the Sight view

The Sight view is an object of WCContentItemViewTemplateNewSight. Do you still remember the indent principle mentioned in the section of recursiveDescription? According to the rule of “the view with more indentation is the subview of the view with less indentation”, WCContentItemViewTemplateNewSight’s subviews include WCSightView, and WCSightView’s subviews include UIImageView and SightPlayerView. Because “Sight” is the nickname of short videos in WeChat, these classes with the name “sight” should be given more attention.

### 9.2.5 Find the long press action selector

Commonly we use addGestureRecognizer: to add a long press gesture recognizer in iOS. Since long press a Sight view shows a menu, the long press gesture is very probably to be added right on the Sight view. Since this view is an object of WCContentItemViewTemplateNewSight, let’s see what’s in its header file:

@interface WCContentItemViewTemplateNewSight : WCContentItemBaseView <WCActionSheetDelegate, SessionSelectControllerDelegate, WCSightViewDelegate>

……

- (void)onMore:(id)arg1;

- (void)onFavoriteAdd:(id)arg1;

- (void)onLongTouch;

- (void)onShowSightAction;

- (void)onLongPressedWCSightFullScreenWindow:(id)arg1;

- (void)onLongPressedWCSight:(id)arg1;

- (void)onClickWCSight:(id)arg1;

……

@end

In the header, methods with keywords “LongTouch” and “LongPressed” are very likely to be our targets. Now, IDA should have finished the initial analysis, let’s take a look at the implementation of these methods. onLongTouch first, as shown in figure 9-9.

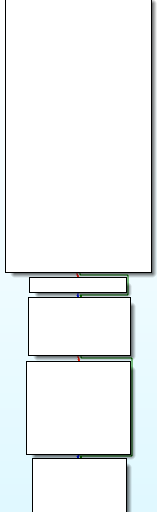


Figure 9-9 onLongTouch

The execution flow of this method is very simple. Look through the method body quickly, “UIMenuController” can be easily found, as shown in figure 9-10.

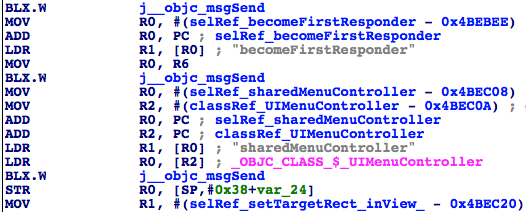


Figure 9-10 onLongTouch

“Favorite” stands out too, as shown in figure 9-11.

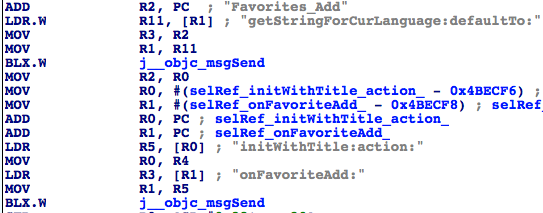


Figure 9-11 onLongTouch

Unless WeChat is intentionally confusing us with these keywords, [WCContentItemViewTemplateNewSight onLongTouch] must be the response method of long press gestures. No need to hurry, let’s take a look at methods with keyword “LongPressed”, as shown in figure 9-12.



Figure 9-12 onLongPressedWCSightFullScreenWindow:

It seems that it records some information, then calls onShowSightAction. Double click onShowSightAction to see its implementation, as shown in figure 9-13.



Figure 9-13 onShowSightAction

In figure 9-13, we know that a WCActionSheet object is created from the very beginning. From its name, we can guess that WCActionSheet acts like UIActionSheet, but we didn’t see any action sheet when we long press the Sight, so onLongPressedWCSightFullScreenWindow: is probably not the method we want.

The last method, onLongPressedWCSight:, is shown in figure 9-14.

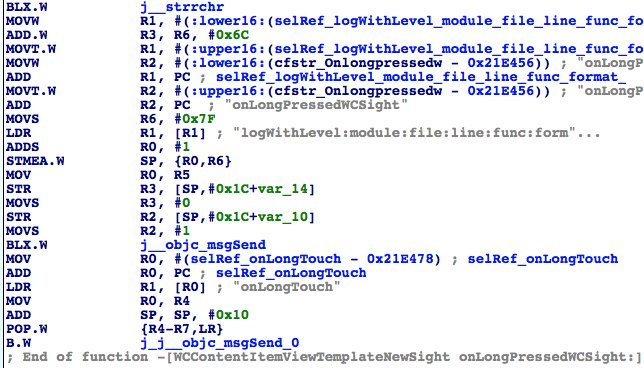


Figure 9-14 onLongPressedWCSight:

From figure 9-14, we can see it records some information, then calls onLongTouch, which proves our conjecture indirectly. Now, let’s debug onLongPressedWCSightFullScreenWindow: and onLongTouch using LLDB. Firstly, attach debugserver to MicroMessenger:

snakeninnysiMac:Documents snakeninny$ ssh root@localhost -p 2222

FunMaker-5:~ root# debugserver \*:1234 -a MicroMessenger

debugserver-@(#)PROGRAM:debugserver PROJECT:debugserver-320.2.89

for armv7.

Attaching to process MicroMessenger...

Listening to port 1234 for a connection from \*...

Waiting for debugger instructions for process 0.

Then check the ASLR offset of WeChat:

(lldb) image list -o -f

[ 0] 0x00000000 /private/var/mobile/Containers/Bundle/Application/E4EBD049-1A75-4830-BC65-0132C0EBC1CA/MicroMessenger.app/MicroMessenger(0x0000000000004000)

[ 1] 0x022dc000 /Library/MobileSubstrate/MobileSubstrate.dylib(0x00000000022dc000)

……

The ASLR offset of WeChat is surprisingly 0x0. Next, let’s check the base addresses of onLongPressedWCSightFullScreenWindow: and onLongTouch, as shown in figure 9-15 and 9-16.

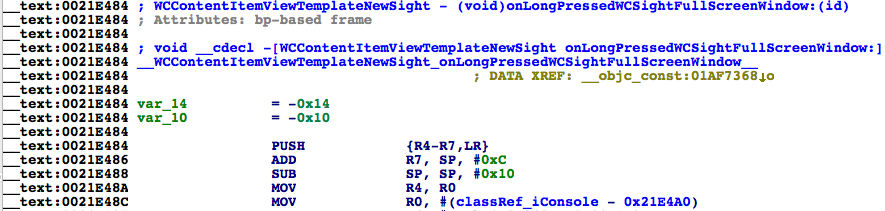


Figure 9-15 onLongPressedWCSightFullScreenWindow:

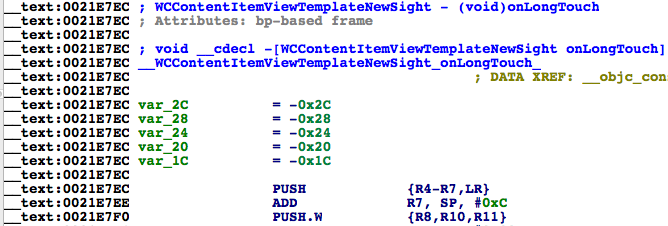


Figure 9-16 onLongTouch

The base addresses of them are 0x21e484 and 0x21e7ec. Set 2 breakpoints on them then long press the Sight view to see whether these breakpoints are triggered:

(lldb) br s -a 0x21e484

Breakpoint 3: where = MicroMessenger`\_\_\_lldb\_unnamed\_function9789$$MicroMessenger, address = 0x0021e484

(lldb) br s -a 0x21e7ec

Breakpoint 4: where = MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger, address = 0x0021e7ec

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x0021e7ec MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger, queue = 'com.apple.main-thread, stop reason = breakpoint 4.1

frame #0: 0x0021e7ec MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger

MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger:

-> 0x21e7ec: push {r4, r5, r6, r7, lr}

0x21e7ee: add r7, sp, #12

0x21e7f0: push.w {r8, r10, r11}

0x21e7f4: sub sp, #32

(lldb) p (char \*)$r1

(char \*) $0 = 0x017fdc2b "onLongTouch"

(lldb) c

Process 184500 resuming

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x0021e7ec MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger, queue = 'com.apple.main-thread, stop reason = breakpoint 4.1

frame #0: 0x0021e7ec MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger

MicroMessenger`\_\_\_lldb\_unnamed\_function9791$$MicroMessenger:

-> 0x21e7ec: push {r4, r5, r6, r7, lr}

0x21e7ee: add r7, sp, #12

0x21e7f0: push.w {r8, r10, r11}

0x21e7f4: sub sp, #32

(lldb) p (char \*)$r1

(char \*) $1 = 0x017fdc2b "onLongTouch"

As we can see, onLongTouch was called twice, and onLongPressedWCSightFullScreenWindow was never called. Take another look at onLongPressedWCSight:, its base address is shown in figure 9-17.

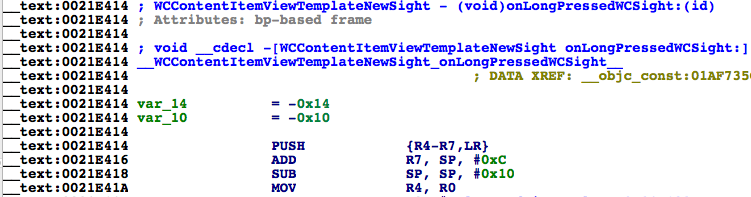


Figure 9- 17 onLongPressedWCSight:

Set a breakpoint on this method to see whether it’s triggered:

(lldb) c

Process 184500 resuming

(lldb) br del

About to delete all breakpoints, do you want to do that?: [Y/n] y

All breakpoints removed. (2 breakpoints)

(lldb) br s -a 0x21e414

Breakpoint 5: where = MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger, address = 0x0021e414

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x0021e414 MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger, queue = 'com.apple.main-thread, stop reason = breakpoint 5.1

frame #0: 0x0021e414 MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger

MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger:

-> 0x21e414: push {r4, r5, r6, r7, lr}

0x21e416: add r7, sp, #12

0x21e418: sub sp, #16

0x21e41a: mov r4, r0

(lldb) p (char \*)$r1

(char \*) $2 = 0x0182c799 "onLongPressedWCSight:"

(lldb) c

Process 184500 resuming

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x0021e414 MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger, queue = 'com.apple.main-thread, stop reason = breakpoint 5.1

frame #0: 0x0021e414 MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger

MicroMessenger`\_\_\_lldb\_unnamed\_function9788$$MicroMessenger:

-> 0x21e414: push {r4, r5, r6, r7, lr}

0x21e416: add r7, sp, #12

0x21e418: sub sp, #16

0x21e41a: mov r4, r0

(lldb) p (char \*)$r1

(char \*) $3 = 0x0182c799 "onLongPressedWCSight:"

(lldb) po $r2

<WCSightView: 0x2454dc0; baseClass = UIControl; frame = (0 3; 200 150); gestureRecognizers = <NSArray: 0x87e5110>; layer = <CALayer: 0xd3be460>>

onLongPressedWCSight: was also called twice, and its argument was an object of WCSightView. By now, we have located the response method of long press gestures, which is onLongPressedWCSight: or onLongTouch. Next, we need to go further to find the trace of the Sight.

### 9.2.6 Find the controller of Sight view using Cycript

First, click “Tap to download” in the Sight view to download the video, as shown in figure 9-18.



Figure 9-18 Download the Sight

Upon successful download, “Tap to download” disappears. The procedures of getting C from V and tracking M from C are repeated so many times, so let’s get our hands dirty for now:

FunMaker-5:~ root# cycript -p MicroMessenger

cy# ?expand

expand == true

cy# [[UIApp keyWindow] recursiveDescription]

@"<iConsoleWindow: 0x2392e50; baseClass = UIWindow; frame = (0 0; 320 568); gestureRecognizers = <NSArray: 0x2391b00>; layer = <UIWindowLayer: 0x2391690>>

| <UILayoutContainerView: 0x7e71870; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x7e71830>>

| | <UITransitionView: 0x7e720b0; frame = (0 0; 320 568); clipsToBounds = YES; autoresize = W+H; layer = <CALayer: 0x7e722a0>>

……

| | | | | | | | | | | | | <WCContentItemViewTemplateNewSight: 0xd3be3e0; frame = (61 64; 200 153); clipsToBounds = YES; layer = <CALayer: 0x7e922d0>>

| | | | | | | | | | | | | | <WCSightView: 0x2454dc0; baseClass = UIControl; frame = (0 3; 200 150); gestureRecognizers = <NSArray: 0x87e5110>; layer = <CALayer: 0xd3be460>>

| | | | | | | | | | | | | | | <UIImageView: 0xd34e8d0; frame = (0 0; 200 150); opaque = NO; userInteractionEnabled = NO; layer = <CALayer: 0xd34e950>>

| | | | | | | | | | | | | | | <SightPlayerView: 0x7e50ff0; frame = (0 0; 200 150); layer = <CALayer: 0xd302770>>

| | | | | | | | | | | | | | | <UIView: 0xd37d9e0; frame = (0 0; 200 150); layer = <CALayer: 0xd37da50>>

| | | | | | | | | | | | | | | | <UIView: 0xd30d5f0; frame = (0 0; 200 150); tag = 10050; layer = <CALayer: 0x87e5650>>

| | | | | | | | | | | | | | | | <SightIconView: 0xd3be2e0; frame = (0 0; 200 150); layer = <CALayer: 0xd3be380>>

| | | | | | | | | | | | | | | | <MMUILabel: 0x7ee7530; baseClass = UILabel; frame = (0 103; 200 20); text = 'Tap to play'; hidden = YES; userInteractionEnabled = NO; tag = 10040; layer = <\_UILabelLayer: 0x7e50dd0>>

……

cy# [#0xd3be3e0 nextResponder]

#"<WCTimeLineCellView: 0x872c530; frame = (0 0; 313 243); tag = 1048577; layer = <CALayer: 0x872ce80>>"

cy# [#0x872c530 nextResponder]

#"<UITableViewCellContentView: 0x8729d80; frame = (0 0; 320 251); gestureRecognizers = <NSArray: 0x8729f80>; layer = <CALayer: 0x8729df0>>"

cy# [#0x8729d80 nextResponder]

#"<MMTableViewCell: 0x8729be0; baseClass = UITableViewCell; frame = (0 1164.33; 320 251); autoresize = W; layer = <CALayer: 0x8729b50>>"

cy# [#0x8729be0 nextResponder]

#"<UITableViewWrapperView: 0xab09890; frame = (0 0; 320 568); gestureRecognizers = <NSArray: 0xab09b00>; layer = <CALayer: 0x7e6e4b0>; contentOffset: {0, 0}; contentSize: {320, 568}>"

cy# [#0xab09890 nextResponder]

#"<MMTableView: 0x30c3200; baseClass = UITableView; frame = (0 0; 320 568); gestureRecognizers = <NSArray: 0xab09600>; layer = <CALayer: 0xab09160>; contentOffset: {0, 1090}; contentSize: {320, 3186.3333}>"

cy# [#0x30c3200 nextResponder]

#"<UIView: 0x7e3b040; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x7e3afd0>>"

cy# [#0x7e3b040 nextResponder]

#"<WCTimeLineViewController: 0x28bd200>"

We’ve got WCTimeLineViewController as expected. Meanwhile, we can guess “Time Line” is the code name of “Moments”.

### 9.2.7 Find the Sight object in WCTimeLineViewController

Look through WCTimeLineViewController.h, there are only a few properties in it; also it has no obvious methods to access M. Yet there are 2 suspicious global variables, as follows:

WCDataItem \*\_inputDataItem;

WCDataItem \*\_cacheDateItem;

But they are both NULL:

cy# #0x28bd200->\_cacheDateItem

null

cy# #0x28bd200->\_inputDataItem

null

Seems like we’ve lost. Is it time to give up? No! Let’s keep calm and carry on: Because Moments are presented as TableView, and there’s a method named tableView:cellForRowAtIndexPath: in WCTimeLineViewController, which means this class implements UITableViewDataSource protocol, so it must have a close relationship with M. Now, jump to this method in IDA, as shown in figure 9-19.

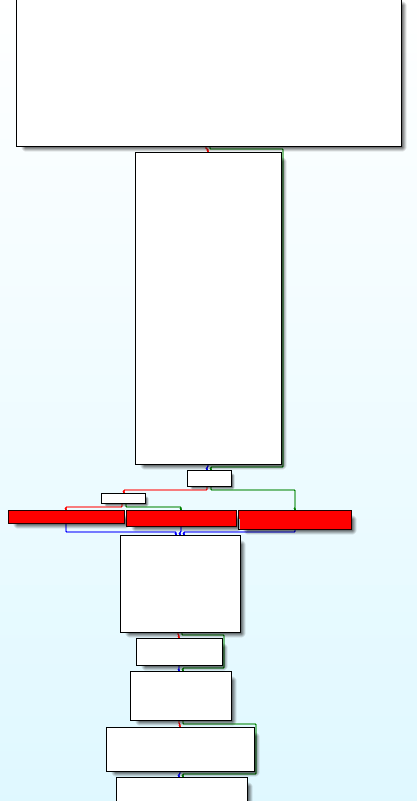


Figure 9-19 [WCTimeLineViewController tableView:cellForRowAtIndexPath:]

Look through this method, you will find 3 dark colored squares in figure 9-17 are the core of this method, other parts are just setting the background, theme and color of the cell. Let’s take a look at these 3 dark colored squares closely, as shown in figure 9-20.

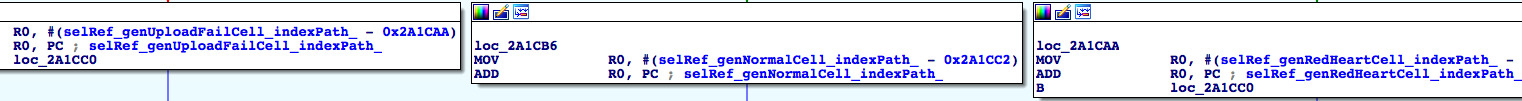


Figure 9-20 A close look at the 3 dark colored squares

From left to right, the methods are genUploadFailCell:indexPath, genNormalCell:indexPath: and genRedHeartCell:indexPath:. Which cell is for Sight? I guess it’s “NormalCell”, let’s take a look at the implementation of genNormalCell:indexPath:, as shown in figure 9-21.

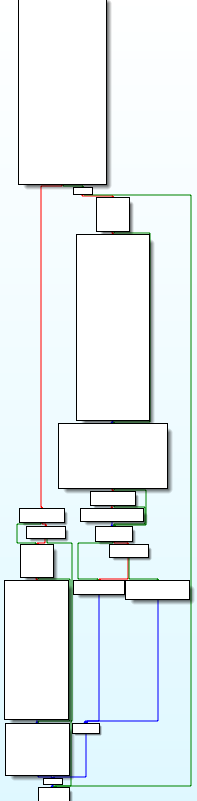


Figure 9- 21 [WCTimeLineViewController genNormalCell:indexPath:]

The logic is not complicated, if you look through the method body, a suspicious method comes up, as shown in figure 9-22.



Figure 9-22 [WCTimeLineViewController genNormalCell:indexPath:]

Infer from the name, getTimelineDataItemOfIndex: in figure 9-22 is probably the data source of the current cell. Set a breakpoint at the bottom instruction, i.e. “\_\_text:002A091C BLX.W j\_\_objc\_msgSend”, then think of a way to trigger it. We have already known that tableView:cellForRowAtIndexPath: is called when UITableView needs to display a new cell. In order to make this breakpoint break on a cell with Sight, we just need to scroll the Sight out of screen, then scroll it back. When the Sight is scrolled out, a new cell will scroll in, hence triggers the breakpoint; there’s no Sight on this cell, this kind of breakpoint doesn’t meet our requirement, so what we do is to disable the breakpoint first, then enable the breakpoint after the Sight is scrolled out of the screen completely. Now we can scroll the Sight back, the breakpoint will break on a cell with Sight:

(lldb) br s -a 0x2A091C

Breakpoint 6: where = MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, address = 0x002a091c

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 6.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) po $r0

Class name: WCDataItem, addr: 0x80f52b0

tid: 11896185303680028954

username: wxid\_hqouu9kgsgw3e6

createtime: 1418135798

commentUsers: (

)

contentObj: <WCContentItem: 0x8724c20>

We’ve got a WCDataItem object, with a WCContentItem object in it. Is the “Data” in WCDataItem a Sight? Let’s test it with LLDB by setting this WCDataItem object to NULL and see what happens. Repeat the previous operations to trigger the breakpoint on a Sight cell:

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 6.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) register write r0 0

(lldb) br del

About to delete all breakpoints, do you want to do that?: [Y/n] y

All breakpoints removed. (1 breakpoint)

(lldb) c

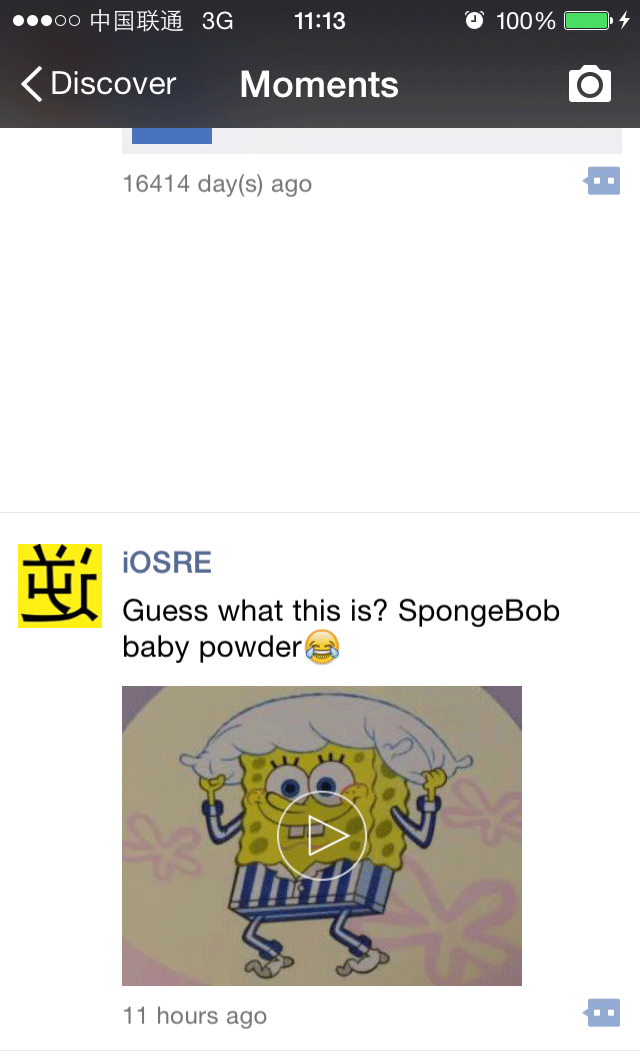


图9- 23 把返回值置NULL的效果

Figure 9-23 effect of setting the return value to NULL

The first Sight totally disappeared, as shown in figure 9-23. This phenomenon indicates that the data source of the Sight is indeed WCDataItem. Before analyzing WCDataItem, there remains one problem to be solved: How can we get a WCDataItem object from the hooked method [WCContentItemViewTemplateNewSight onLongTouch]?

### 9.2.8 Get a WCDataItem object from WCContentItemViewTemplateNewSight

Do you still remember how we get an object of WCDataItem in LLDB? The answer is getTimelineDataItemOfIndex:. Go back to figure 9-22 to see the callers and arguments of this method.

As we can see, the caller is the return value of getService:, the argument is the return value of calcDataItemIndex:, as shown in figure 9-24.

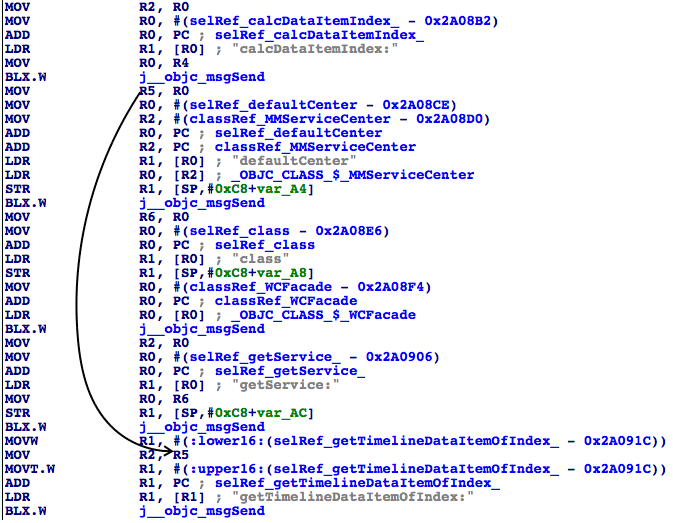


Figure 9-24 Analyze getTimelineDataItemOfIndex:

New problems emerge: How do we call getService: and calcDataItemIndex:? Let’s start from getService:. From the instruction “MOV R0, R6”, we know the caller is R6; R6 is the return value of [MMServiceCenter defaultCenter]. The argument is from the return value of [WCFacade class], as shown in figure 9-25.

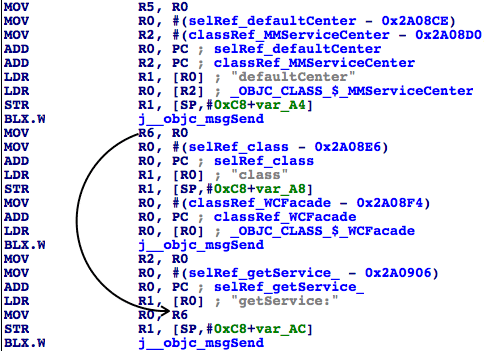


Figure 9-25 Analyze getTimelineDataItemOfIndex:

So the caller of getTimelineDataItemOfIndex: can be obtained by [[MMServiceCenter defaultCenter] getService:[WCFacade class]]. Next, let’s continue with calcDataItemIndex:. From the instruction “MOV R0, R4”, we know the caller is R4 and R4 is “self”. The argument is from the return value of [indexPath section], as shown in figure 9-26 and 9-27.

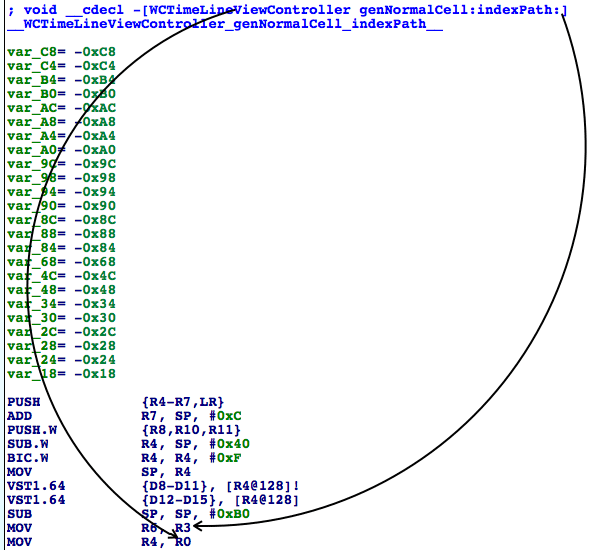


Figure 9-26 Analyze getTimelineDataItemOfIndex:

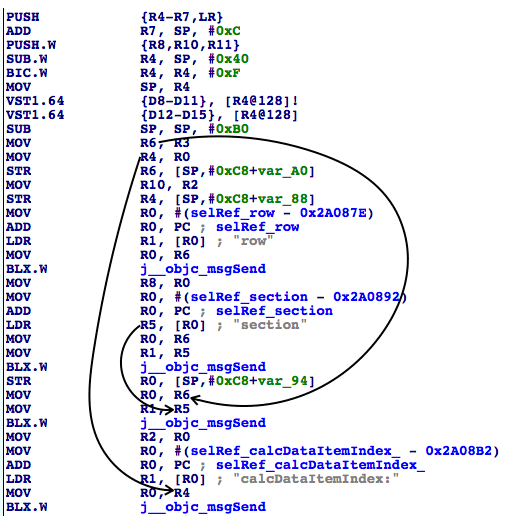


Figure 9-27 Analyze getTimelineDataItemOfIndex

Therefore, the argument of getTimelineDataItemOfIndex: can be obtained from [WCTimeLineViewController calcDataItemIndex:[indexPath section]]. Because we are inside [WCContentItemViewTemplateNewSight onLongTouch], we can get MMTableViewCell, MMTableView and WCTimeLineViewController in sequence via [self nextResponder], then get indexPath via [MMTableView indexPathForCell:MMTableViewCell], and the whole process has already been proved in section 9.2.6. Although it looks inconvenient, obtaining the WCDataItem object from WCContentItemViewTemplateNewSight conforms to standard MVC design pattern at least. It is worth mentioning that the prefixes of WCTimeLineViewController and WCContentItemViewTemplateNewSight are WC, I guess it is short for WeChat; the prefixes of MMTableViewCell and MMTableView are MM, I guess it is short for MicroMessenger. The difference of prefixes may be exactly caused by different subprojects and teams. Next, we will focus on WCDataItem and try to get the download URL and local path of the Sight.

### 9.2.9 Get target information from WCDataItem

Open WCDataItem.h and take an overview:

@interface WCDataItem : NSObject <NSCoding>

{

int cid;

NSString \*tid;

int type;

int flag;

NSString \*username;

NSString \*nickname;

int createtime;

NSString \*sourceUrl;

NSString \*sourceUrl2;

WCLocationInfo \*locationInfo;

BOOL isPrivate;

NSMutableArray \*sharedGroupIDs;

NSMutableArray \*blackUsers;

NSMutableArray \*visibleUsers;

unsigned long extFlag;

BOOL likeFlag;

int likeCount;

NSMutableArray \*likeUsers;

int commentCount;

NSMutableArray \*commentUsers;

int withCount;

NSMutableArray \*withUsers;

WCContentItem \*contentObj;

WCAppInfo \*appInfo;

NSString \*publicUserName;

NSString \*sourceUserName;

NSString \*sourceNickName;

NSString \*contentDesc;

NSString \*contentDescPattern;

int contentDescShowType;

int contentDescScene;

WCActionInfo \*actionInfo;

unsigned int hash;

SnsObject \*snsObject;

BOOL isBidirectionalFan;

BOOL noChange;

BOOL isRichText;

NSMutableDictionary \*extData;

int uploadErrType;

NSString \*statisticsData;

}

+ (id)fromBuffer:(id)arg1;

+ (id)fromServerObject:(id)arg1;

+ (id)fromUploadTask:(id)arg1;

@property(retain, nonatomic) WCActionInfo \*actionInfo; // @synthesize actionInfo;

@property(retain, nonatomic) WCAppInfo \*appInfo; // @synthesize appInfo;

@property(retain, nonatomic) NSArray \*blackUsers; // @synthesize blackUsers;

@property(nonatomic) int cid; // @synthesize cid;

@property(nonatomic) int commentCount; // @synthesize commentCount;

@property(retain, nonatomic) NSMutableArray \*commentUsers; // @synthesize commentUsers;

- (int)compareDesc:(id)arg1;

- (int)compareTime:(id)arg1;

@property(retain, nonatomic) NSString \*contentDesc; // @synthesize contentDesc;

@property(retain, nonatomic) NSString \*contentDescPattern; // @synthesize contentDescPattern;

@property(nonatomic) int contentDescScene; // @synthesize contentDescScene;

@property(nonatomic) int contentDescShowType; // @synthesize contentDescShowType;

@property(retain, nonatomic) WCContentItem \*contentObj; // @synthesize contentObj;

@property(nonatomic) int createtime; // @synthesize createtime;

- (void)dealloc;

- (id)description;

- (id)descriptionForKeyPaths;

- (void)encodeWithCoder:(id)arg1;

@property(retain, nonatomic) NSMutableDictionary \*extData; // @synthesize extData;

@property(nonatomic) unsigned long extFlag; // @synthesize extFlag;

@property(nonatomic) int flag; // @synthesize flag;

- (id)getDisplayCity;

- (id)getMediaWraps;

- (BOOL)hasSharedGroup;

- (unsigned int)hash;

- (id)init;

- (id)initWithCoder:(id)arg1;

@property(nonatomic) BOOL isBidirectionalFan; // @synthesize isBidirectionalFan;

- (BOOL)isEqual:(id)arg1;

@property(nonatomic) BOOL isPrivate; // @synthesize isPrivate;

- (BOOL)isRead;

@property(nonatomic) BOOL isRichText; // @synthesize isRichText;

- (BOOL)isUploadFailed;

- (BOOL)isUploading;

- (BOOL)isValid;

- (id)itemID;

- (int)itemType;

- (id)keyPaths;

@property(nonatomic) int likeCount; // @synthesize likeCount;

@property(nonatomic) BOOL likeFlag; // @synthesize likeFlag;

@property(retain, nonatomic) NSMutableArray \*likeUsers; // @synthesize likeUsers;

- (void)loadPattern;

@property(retain, nonatomic) WCLocationInfo \*locationInfo; // @synthesize locationInfo;

- (void)mergeLikeUsers:(id)arg1;

- (void)mergeMessage:(id)arg1;

- (void)mergeMessage:(id)arg1 needParseContent:(BOOL)arg2;

@property(retain, nonatomic) NSString \*nickname; // @synthesize nickname;

@property(nonatomic) BOOL noChange; // @synthesize noChange;

- (void)parseContentForNetWithDataItem:(id)arg1;

- (void)parseContentForUI;

- (void)parsePattern;

@property(retain, nonatomic) NSString \*publicUserName; // @synthesize publicUserName;

- (id)sequence;

- (void)setCreateTime:(unsigned long)arg1;

- (void)setHash:(unsigned int)arg1;

- (void)setIsUploadFailed:(BOOL)arg1;

- (void)setSequence:(id)arg1;

@property(retain, nonatomic) NSMutableArray \*sharedGroupIDs; // @synthesize sharedGroupIDs;

@property(retain, nonatomic) SnsObject \*snsObject; // @synthesize snsObject;

@property(retain, nonatomic) NSString \*sourceNickName; // @synthesize sourceNickName;

@property(retain, nonatomic) NSString \*sourceUrl2; // @synthesize sourceUrl2;

@property(retain, nonatomic) NSString \*sourceUrl; // @synthesize sourceUrl;

@property(retain, nonatomic) NSString \*sourceUserName; // @synthesize sourceUserName;

@property(retain, nonatomic) NSString \*statisticsData; // @synthesize statisticsData;

@property(retain, nonatomic) NSString \*tid; // @synthesize tid;

@property(nonatomic) int type; // @synthesize type;

@property(nonatomic) int uploadErrType; // @synthesize uploadErrType;

@property(retain, nonatomic) NSString \*username; // @synthesize username;

@property(retain, nonatomic) NSArray \*visibleUsers; // @synthesize visibleUsers;

@property(nonatomic) int withCount; // @synthesize withCount;

@property(retain, nonatomic) NSMutableArray \*withUsers; // @synthesize withUsers;

- (id)toBuffer;

@end

There are 4 occurrences of “path” and “url” keywords:

- (id)descriptionForKeyPaths;

- (id)keyPaths;

@property(retain, nonatomic) NSString \*sourceUrl2;

@property(retain, nonatomic) NSString \*sourceUrl;

Now let’s inspect their return values via LLDB. Repeat the previous operations to trigger the breakpoint on a Sight cell:

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 7.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) po [$r0 descriptionForKeyPaths]

Class name: WCDataItem, addr: 0x80f52b0

tid: 11896185303680028954

username: wxid\_hqouu9kgsgw3e6

createtime: 1418135798

commentUsers: (

)

contentObj: <WCContentItem: 0x8724c20>

(lldb) po [$r0 keyPaths]

<\_\_NSArrayI 0x87b5260>(

tid,

username,

createtime,

commentUsers,

contentObj

)

(lldb) po [$r0 sourceUrl2]

nil

(lldb) po [$r0 sourceUrl]

nil

Seems there is nothing special in the return values, but WCContentItem shows up so many times and grabs my attention. Obviously, the meaning of “content” is more specific than “data”, the content of the Sight may be supplied by this object. Now, take a look at WCContentItem.h:

@interface WCContentItem : NSObject <NSCoding>

{

NSString \*title;

NSString \*desc;

NSString \*titlePattern;

NSString \*descPattern;

NSString \*linkUrl;

NSString \*linkUrl2;

int type;

int flag;

NSString \*username;

NSString \*nickname;

int createtime;

NSMutableArray \*mediaList;

}

@property(nonatomic) int createtime; // @synthesize createtime;

- (void)dealloc;

@property(retain, nonatomic) NSString \*desc; // @synthesize desc;

@property(retain, nonatomic) NSString \*descPattern; // @synthesize descPattern;

- (void)encodeWithCoder:(id)arg1;

@property(nonatomic) int flag; // @synthesize flag;

- (id)init;

- (id)initWithCoder:(id)arg1;

- (BOOL)isValid;

@property(retain, nonatomic) NSString \*linkUrl; // @synthesize linkUrl;

@property(retain, nonatomic) NSString \*linkUrl2; // @synthesize linkUrl2;

@property(retain, nonatomic) NSMutableArray \*mediaList; // @synthesize mediaList;

@property(retain, nonatomic) NSString \*nickname; // @synthesize nickname;

@property(retain, nonatomic) NSString \*title; // @synthesize title;

@property(retain, nonatomic) NSString \*titlePattern; // @synthesize titlePattern;

@property(nonatomic) int type; // @synthesize type;

@property(retain, nonatomic) NSString \*username; // @synthesize username;

@end

There are 2 occurrences of “url”:

@property(retain, nonatomic) NSString \*linkUrl;

@property(retain, nonatomic) NSString \*linkUrl2;

We can get a WCContentItem object via [WCDataItem contentObj], then use LLDB to print the values of the above 2 properties. Repeat the previous operations to trigger the breakpoint on a Sight cell:

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 8.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) po [[$r0 contentObj] linkUrl]

https://support.weixin.qq.com/cgi-bin/mmsupport-bin/readtemplate?t=page/common\_page\_\_upgrade&v=1

(lldb) po [[$r0 contentObj] linkUrl2]

nil

Type this URL in browser to see what we’ve got, as shown in figure 9-28.



Figure 9-28 [[$r0 contentObj] linkUrl]

The result is not what we want. Since there is not too much content in WCContentItem.h, where could the Sight be? Back to this file, a property named mediaList attracts my attention. It is even more accurate than “content”, is the Sight hidden in it? LLDB will answer us. Repeat the previous operations to trigger the breakpoint on a Sight cell:

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 8.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) po [[$r0 contentObj] mediaList]

<\_\_NSArrayM 0x8725580>(

<WCMediaItem: 0x7e78490>

)

Now, a new class WCMediaItem appears. Let’s check WCMediaItem.h:

@interface WCMediaItem : NSObject <NSCoding>

{

NSString \*mid;

int type;

int subType;

NSString \*title;

NSString \*desc;

NSString \*titlePattern;

NSString \*descPattern;

NSString \*userData;

NSString \*source;

NSMutableArray \*previewUrls;

WCUrl \*dataUrl;

WCUrl \*lowBandUrl;

struct CGSize imgSize;

BOOL likeFlag;

int likeCount;

NSMutableArray \*likeUsers;

int commentCount;

NSMutableArray \*commentUsers;

int withCount;

NSMutableArray \*withUsers;

int createTime;

}

- (id).cxx\_construct;

- (id)cityForData;

@property(nonatomic) int commentCount; // @synthesize commentCount;

@property(retain, nonatomic) NSMutableArray \*commentUsers; // @synthesize commentUsers;

- (id)comparativePathForPreview;

@property(nonatomic) int createTime; // @synthesize createTime;

@property(retain, nonatomic) WCUrl \*dataUrl; // @synthesize dataUrl;

- (void)dealloc;

@property(retain, nonatomic) NSString \*desc; // @synthesize desc;

@property(retain, nonatomic) NSString \*descPattern; // @synthesize descPattern;

- (void)encodeWithCoder:(id)arg1;

- (BOOL)hasData;

- (BOOL)hasPreview;

- (BOOL)hasSight;

- (id)hashPathForString:(id)arg1;

- (id)imageOfSize:(int)arg1;

@property(nonatomic) struct CGSize imgSize; // @synthesize imgSize;

- (id)init;

- (id)initWithCoder:(id)arg1;

- (BOOL)isValid;

@property(nonatomic) int likeCount; // @synthesize likeCount;

@property(nonatomic) BOOL likeFlag; // @synthesize likeFlag;

@property(retain, nonatomic) NSMutableArray \*likeUsers; // @synthesize likeUsers;

- (CDStruct\_c3b9c2ee)locationForData;

@property(retain, nonatomic) WCUrl \*lowBandUrl; // @synthesize lowBandUrl;

- (id)mediaID;

- (int)mediaType;

@property(retain, nonatomic) NSString \*mid; // @synthesize mid;

- (id)pathForData;

- (id)pathForPreview;

- (id)pathForSightData;

@property(retain, nonatomic) NSMutableArray \*previewUrls; // @synthesize previewUrls;

- (BOOL)saveDataFromData:(id)arg1;

- (BOOL)saveDataFromMedia:(id)arg1;

- (BOOL)saveDataFromPath:(id)arg1;

- (BOOL)savePreviewFromData:(id)arg1;

- (BOOL)savePreviewFromMedia:(id)arg1;

- (BOOL)savePreviewFromPath:(id)arg1;

- (BOOL)saveSightDataFromData:(id)arg1;

- (BOOL)saveSightDataFromMedia:(id)arg1;

- (BOOL)saveSightDataFromPath:(id)arg1;

- (BOOL)saveSightPreviewFromMedia:(id)arg1;

@property(retain, nonatomic) NSString \*source; // @synthesize source;

@property(nonatomic) int subType; // @synthesize subType;

@property(retain, nonatomic) NSString \*title; // @synthesize title;

@property(retain, nonatomic) NSString \*titlePattern; // @synthesize titlePattern;

@property(nonatomic) int type; // @synthesize type;

@property(retain, nonatomic) NSString \*userData; // @synthesize userData;

@property(nonatomic) int withCount; // @synthesize withCount;

@property(retain, nonatomic) NSMutableArray \*withUsers; // @synthesize withUsers;

- (id)videoStreamForData;

- (id)voiceStreamForData;

@end

There are 8 occurrences of “path”:

- (id)comparativePathForPreview;

- (id)hashPathForString:(id)arg1;

- (id)pathForData;

- (id)pathForPreview;

- (id)pathForSightData;

- (BOOL)saveDataFromPath:(id)arg1;

- (BOOL)savePreviewFromPath:(id)arg1;

- (BOOL)saveSightDataFromPath:(id)arg1;

And 3 occurrences of “url”:

@property(retain, nonatomic) WCUrl \*dataUrl;

@property(retain, nonatomic) WCUrl \*lowBandUrl;

@property(retain, nonatomic) NSMutableArray \*previewUrls;

Among those methods and properties, pathForData, pathForPreview and pathForSightData are very likely to return paths; dataUrl, lowBandUrl and previewUrls are very likely to return URLs. Verify our guess ASAP with LLDB, repeat the previous operations to trigger the breakpoint on a Sight cell:

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208, queue = 'com.apple.main-thread, stop reason = breakpoint 8.1

frame #0: 0x002a091c MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 208:

-> 0x2a091c: blx 0xe08e0c ; \_\_\_lldb\_unnamed\_function70162$$MicroMessenger

0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

(lldb) ni

Process 184500 stopped

\* thread #1: tid = 0x2d0b4, 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212, queue = 'com.apple.main-thread, stop reason = instruction step over

frame #0: 0x002a0920 MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212

MicroMessenger`\_\_\_lldb\_unnamed\_function11980$$MicroMessenger + 212:

-> 0x2a0920: mov r11, r0

0x2a0922: movw r0, #32442

0x2a0926: movt r0, #436

0x2a092a: add r0, pc

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] pathForData]

/var/mobile/Containers/Data/Application/E9BE84D8-9982-4814-9289-823D5FD91144/Library/WechatPrivate/c5f5eb23e53bb2ee021b0e89b5c4bc9a/wc/media/5/60/2a16b0b62baf39924448a74fa03ff2

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] pathForPreview]

/var/mobile/Containers/Data/Application/E9BE84D8-9982-4814-9289-823D5FD91144/Library/WechatPrivate/c5f5eb23e53bb2ee021b0e89b5c4bc9a/wc/media/5/7f/cdc7939813d1a95feda4bed05f9b82

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] pathForSightData]

/var/mobile/Containers/Data/Application/E9BE84D8-9982-4814-9289-823D5FD91144/Library/WechatPrivate/c5f5eb23e53bb2ee021b0e89b5c4bc9a/wc/media/5/60/2a16b0b62baf39924448a74fa03ff2.mp4

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] dataUrl]

type[1], url[http://vcloud1023.tc.qq.com/1023\_0114929ce86949a8bfb6f7b46b6b39b8.f0.mp4]

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] lowBandUrl]

nil

(lldb) po [[[[$r0 contentObj] mediaList] objectAtIndex:0] previewUrls]

<\_\_NSArrayM 0x8725950>(

type[1], url[http://mmsns.qpic.cn/mmsns/WiaWbRORjpHsUXcNL3dNsVLDibRZ9oufPnXeJqZdlG4xhND43M87sh7DRcxttVPxAO/0]

)

From the file names, I am pretty sure that they are the Sight information we’re looking for. Whatever it is ssh or iFunBox that opens the local files; whether it be MobileSafari or Chrome that opens the URL, you can come to these conclusions:

* The value of pathForData is the local path of the Sight without suffix.
* The value of pathForPreview is the path of the Sight’s preview image without suffix.
* The value of pathForSightData is the local path of the Sight with suffix.
* The value of dataUrl is the Internet URL of the Sight.
* The value of lowBandUrl is nil, but I guess this value is not nil when the network condition is not good. In order to save bandwidth, file from this URL may be smaller than file from dataURL on size.
* The value of previewUrls is the Internet URL of the Sight’s preview images.

The prototyping of tweak is finished for now. Let’s comb our thoughts before coding.

### 9.3 Result interpretation

This practice covers Cycript, IDA and LLDB, we’ve prototyped the tweak without strictly deducing the execution logic of WeChat. Now I will do a brief summary of our thoughts.

1. Add a long press gesture to Sight view

Because there’s already a long press gesture on Sight view, there’s no need to reinvent the wheel, we just need to find the existing one and hook it. With Reveal, we can locate the Sight view easily, thus find the action selector of the long press gesture. What is worth mentioning is that the action selector will be called twice, leading to inefficiency. We need to take this situation into consideration when writing tweak, and compose a proper condition to make the method execute only once.

1. Find the Sight object in C

Although the MVC design pattern says that M can be accessed through C, in this example, we cannot find any obvious methods in C to access M. Therefore, we started with the basic data source method tableView:cellForRowAtIndexPath: to find the suspicious data source of a cell, then looked through suspicious properties and methods in headers to locate the Sight object, and finally got the wanted information. Perhaps the procedure was not so rigorous, but we reached our goal and saved our time, it was not bad, right?

### 9.4 Tweak writing

The target of this section is to replace the options of the original long press menu with “Save to Disk” and “Copy URL”. With a well-constructed prototype, we don’t have to explain it any further, let’s get coding now.

### 9.4.1 Create tweak project “ iOSREWCVideoDownloader” using Theos

The Theos commands are as follows:

hangcom-mba:Documents sam$ /opt/theos/bin/nic.pl

NIC 2.0 - New Instance Creator

------------------------------

[1.] iphone/application

[2.] iphone/cydget

[3.] iphone/framework

[4.] iphone/library

[5.] iphone/notification\_center\_widget

[6.] iphone/preference\_bundle

[7.] iphone/sbsettingstoggle

[8.] iphone/tool

[9.] iphone/tweak

[10.] iphone/xpc\_service

Choose a Template (required): 9

Project Name (required): iOSREWCVideoDownloader

Package Name [com.yourcompany.iosrewcvideodownloader]: com.iosre.iosrewcvideodownloader

Author/Maintainer Name [sam]: sam

[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]: com.tencent.xin

[iphone/tweak] List of applications to terminate upon installation (space-separated, '-' for none) [SpringBoard]: MicroMessenger

Instantiating iphone/tweak in iosrewcvideodownloader/...

Done.

### 9.4.2 Compose iOSREWCVideoDownloader.h

The finalized iOSREWCVideoDownloader.h looks like this:

@interface WCContentItem : NSObject

@property (retain, nonatomic) NSMutableArray \*mediaList;

@end

@interface WCDataItem : NSObject

@property (retain, nonatomic) WCContentItem \*contentObj;

@end

@interface WCUrl : NSObject

@property (retain, nonatomic) NSString \*url;

@end

@interface WCMediaItem : NSObject

@property (retain, nonatomic) WCUrl \*dataUrl;

- (id)pathForSightData;

@end

@interface WCContentItemViewTemplateNewSight : UIView

- (WCMediaItem \*)iOSREMediaItemFromSight;

- (void)iOSREOnSaveToDisk;

- (void)iOSREOnCopyURL;

@end

@interface MMServiceCenter : NSObject

+ (id)defaultCenter;

- (id)getService:(Class)arg1;

@end

@interface WCFacade : NSObject

- (WCDataItem \*)getTimelineDataItemOfIndex:(int)arg1;

@end

@interface WCTimeLineViewController : NSObject

- (int)calcDataItemIndex:(int)arg1;

@end

@interface MMTableViewCell : UITableViewCell

@end

@interface MMTableView : UITableView

@end

This header is composed by picking snippets from other class-dump headers. The existence of this header is simply for avoiding any warnings or errors when compiling the tweak.

### 9.4.3 Edit Tweak.xm

The finalized Tweak.xm looks like this:

#import "iOSREWCVideoDownloader.h"

static MMTableViewCell \*iOSRECell;

static MMTableView \*iOSREView;

static WCTimeLineViewController \*iOSREController;

%hook WCContentItemViewTemplateNewSight

%new

- (WCMediaItem \*)iOSREMediaItemFromSight

{

id responder = self;

while (![responder isKindOfClass:NSClassFromString(@"WCTimeLineViewController")])

{

if ([responder isKindOfClass:NSClassFromString(@"MMTableViewCell")]) iOSRECell = responder;

else if ([responder isKindOfClass:NSClassFromString(@"MMTableView")]) iOSREView = responder;

responder = [responder nextResponder];

}

iOSREController = responder;

if (!iOSRECell || !iOSREView || !iOSREController)

{

NSLog(@"iOSRE: Failed to get video object.");

return nil;

}

NSIndexPath \*indexPath = [iOSREView indexPathForCell:iOSRECell];

int itemIndex = [iOSREController calcDataItemIndex:[indexPath section]];

WCFacade \*facade = [(MMServiceCenter \*)[%c(MMServiceCenter) defaultCenter] getService:[%c(WCFacade) class]];

WCDataItem \*dataItem = [facade getTimelineDataItemOfIndex:itemIndex];

WCContentItem \*contentItem = dataItem.contentObj;

WCMediaItem \*mediaItem = [contentItem.mediaList count] != 0 ? (contentItem.mediaList)[0] : nil;

return mediaItem;

}

%new

- (void)iOSREOnSaveToDisk

{

NSString \*localPath = [[self iOSREMediaItemFromSight] pathForSightData];

UISaveVideoAtPathToSavedPhotosAlbum(localPath, nil, nil, nil);

}

%new

- (void)iOSREOnCopyURL

{

UIPasteboard \*pasteboard = [UIPasteboard generalPasteboard];

pasteboard.string = [self iOSREMediaItemFromSight].dataUrl.url;

}

static int iOSRECounter;

- (void)onLongTouch

{

iOSRECounter++;

if (iOSRECounter % 2 == 0) return;

[self becomeFirstResponder];

UIMenuItem \*saveToDiskMenuItem = [[UIMenuItem alloc] initWithTitle:@"Save to Disk" action:@selector(iOSREOnSaveToDisk)];

UIMenuItem \*copyURLMenuItem = [[UIMenuItem alloc] initWithTitle:@"Copy URL" action:@selector(iOSREOnCopyURL)];

UIMenuController \*menuController = [UIMenuController sharedMenuController];

[menuController setMenuItems:@[saveToDiskMenuItem, copyURLMenuItem]];

[menuController setTargetRect:CGRectZero inView:self];

[menuController setMenuVisible:YES animated:YES];

[saveToDiskMenuItem release];

[copyURLMenuItem release];

}

%end

### 9.4.4 Edit Makefile and control files

The finalized Makefile looks like this:

export THEOS\_DEVICE\_IP = iOSIP

export TARGET = iphone:clang:latest:8.0

export ARCHS = armv7 arm64

include theos/makefiles/common.mk

TWEAK\_NAME = iOSREWCVideoDownloader

iOSREWCVideoDownloader\_FILES = Tweak.xm

iOSREWCVideoDownloader\_FRAMEWORKS = UIKit

include $(THEOS\_MAKE\_PATH)/tweak.mk

after-install::

install.exec "killall -9 MicroMessenger"

The finalized control looks like this:

Package: com.iosre.iosrewcvideodownloader

Name: iOSREWCVideoDownloader

Depends: mobilesubstrate, firmware (>= 8.0)

Version: 1.0

Architecture: iphoneos-arm

Description: Play with Sight!

Maintainer: sam

Author: sam

Section: Tweaks

Homepage: http://bbs.iosre.com

### 9.4.5 Test

Compile and install the tweak, then launch WeChat and long press a Sight, it will show our custom menu, as shown in figure 9-29.



Figure 9-29 Custom menu

Click “Save to Disk”, the Sight will be saved to local album, as shown in figure 9-30.

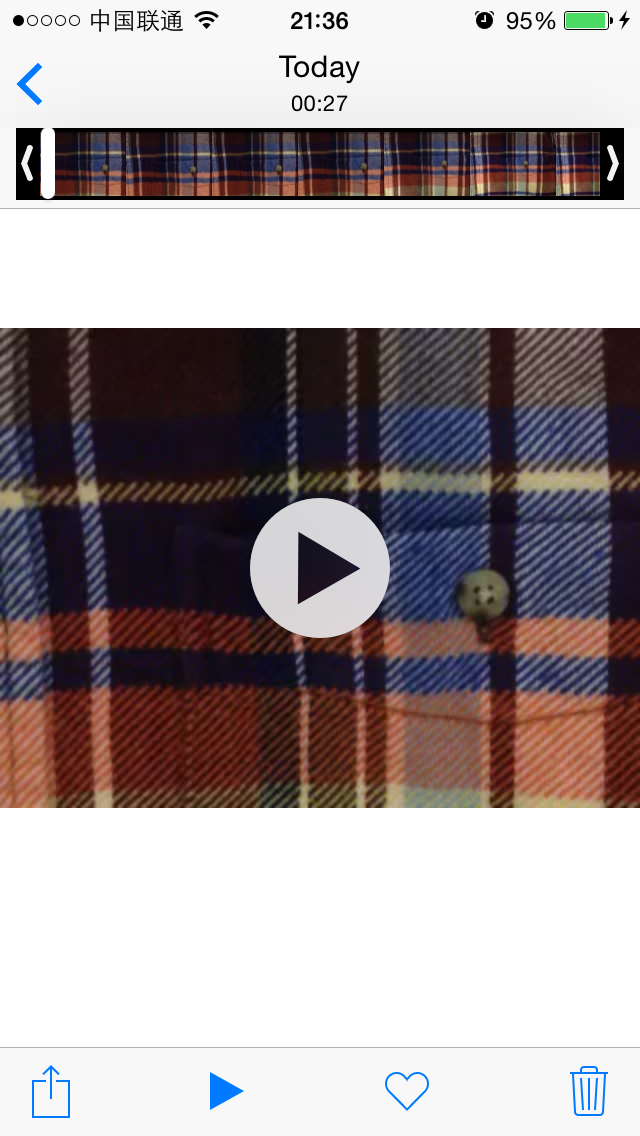


Figure 9-30 Save the Sight to local album

Click “Copy URL”, and paste it in OPlayer Lite (or any other Apps that support online video playing), as shown in figure 9-31.

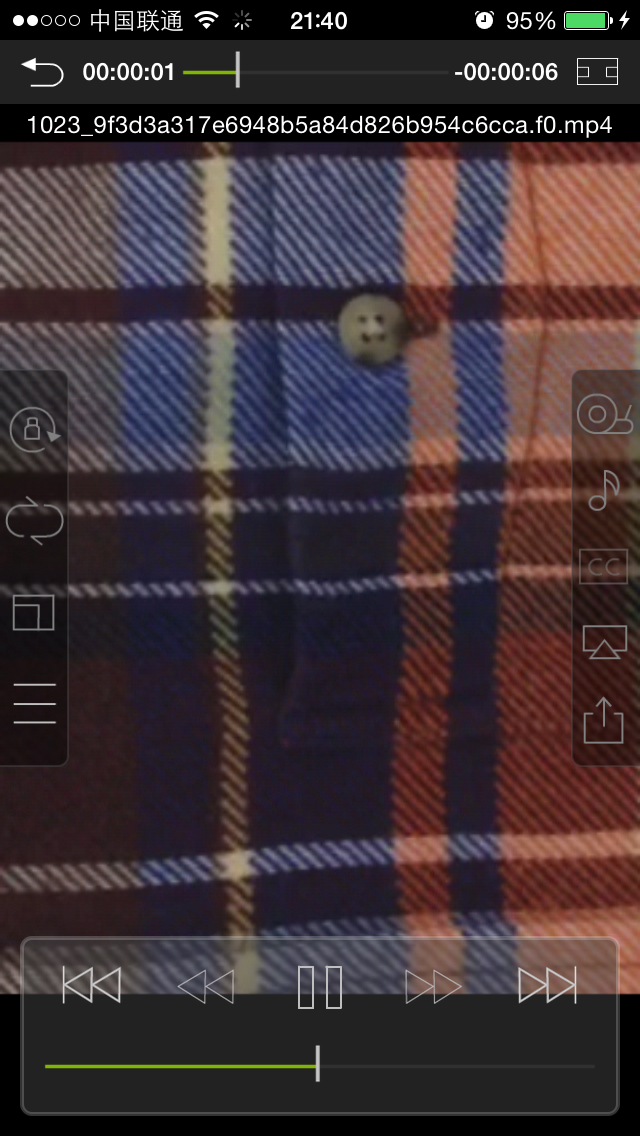


Figure 9-31 Play online Sight in OPlayer Lite

All functions work as expected, mission accomplished!

## 9.5 Easter eggs

### 9.5.1 Find the Sight in UIMenuItem

In section 9.2.7, we’ve successfully found the Sight from WCTimeLineViewController. However, the whole process was not smooth: we haven’t managed to find a direct way of accessing M via C, so we “had to” find some clues from tableView:cellForRowAtIndexPath: to meet our needs. If we jump out of MVC and try to think from the view of WeChat itself, things may get much easier.

Think with me: Long press the Sight view, a menu shows. Choose the menu option, a corresponding operation will be carried out on the Sight. In other words, there may be Sight related clues in the action selector of UIMenuItem. In figure 9-11, we have already seen the action selector of “Favorite”, which is onFavoriteAdd:, let’s check its implementation in IDA, as shown in figure 9-32.

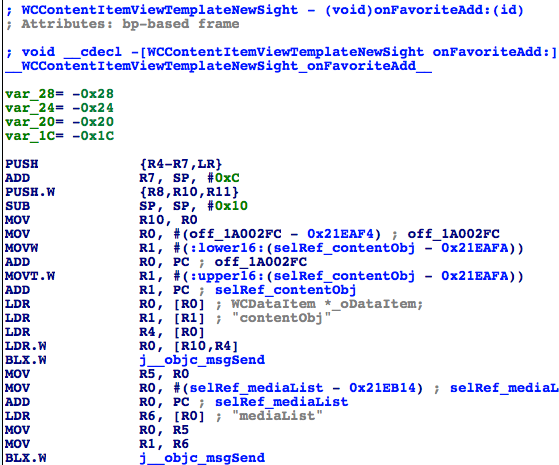


Figure 9-32 [WCContentItemViewTemplateNewSight onFavoriteAdd:]

From figure 9-32, we see our familiar WCDataItem, contentObj and mediaList at the beginning of this method. If we’ve started with this method, the whole analysis workload would reduce by half at least. It more or less enlightens us that although MVC design pattern is a common trail of thinking in iOS App reverse engineering, if we can occasionally think off the track, we may get something unexpected and have more fun.

### 9.5.2 Historical transition of WeChat's headers count

From the historical transition of WeChat's headers count as figure 9-33 to figure 9-38 show, we can see how WeChat becomes excellent step by step. A journey of a thousand miles begins with a single step, kudos to WeChat!

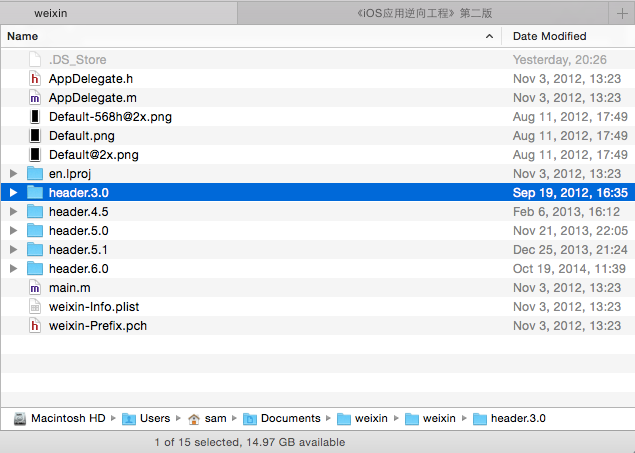


Figure 9-23 Headers directories of different WeChat versions

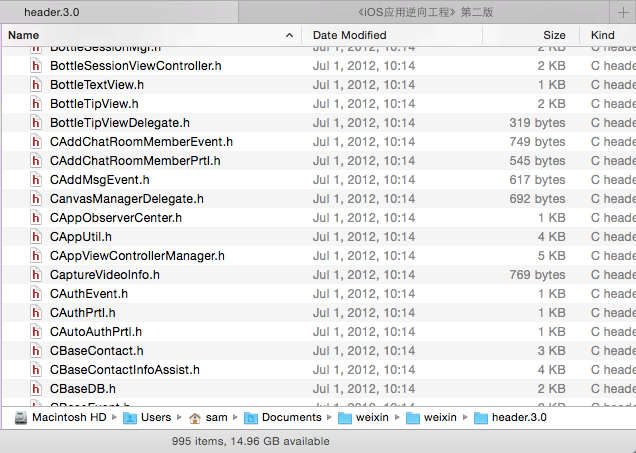


Figure 9-34 WeChat 3.0, 995 headers

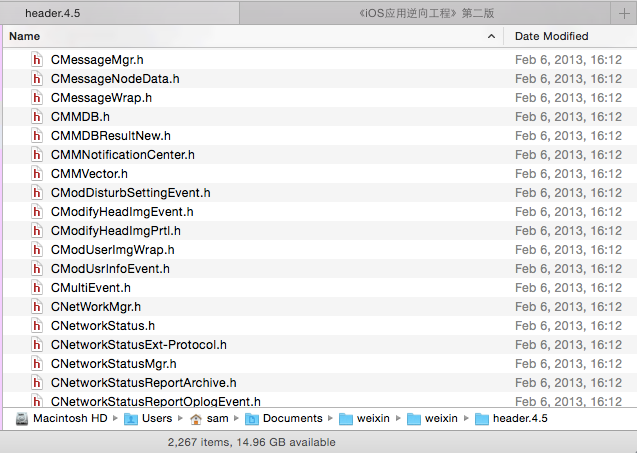


Figure 9-35 WeChat 4.5, 2267 headers

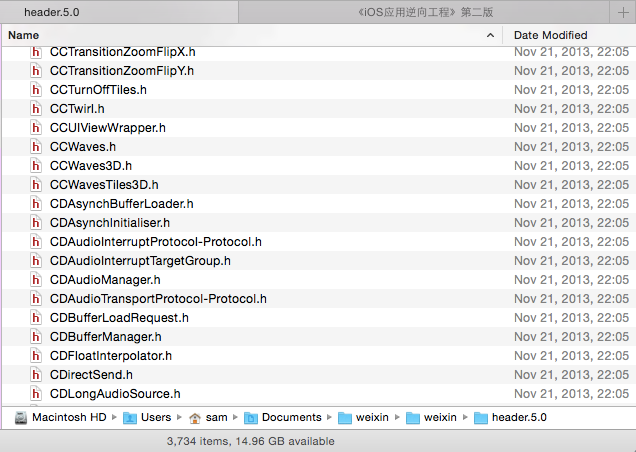


Figure 9-36 WeChat 5.0, 3734 headers

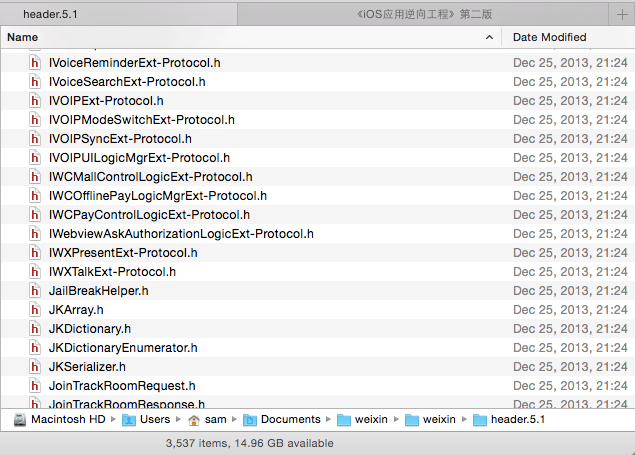


Figure 9-37 WeChat 5.1, 3537 headers



Figure 9-38 WeChat 6.0, 5225 headers

From WeChat 3.0 to WeChat 6.0, the number of headers has increased from less than 1,000 to more than 5,000, which is a 5+ times amplification. With the global popularity of WeChat, its headers count is expected to surpass 10,000 sooner or later.

## 9.6 Conclusion

WeChat is the target of this chapter, we’ve enriched Sight by adding 2 new features, i.e. " Save to Disk " and "Copy URL". As a powerful platform, WeChat possesses complicated structure and large amount of code; it was beautifully designed with clearly separated modules and well organized code. We have already learnt so much from it by just going through its headers, we can even see the different coding styles of different developers. I believe all of us can benefit a lot from studying WeChat's design pattern by reversing it. We will discuss what we find reversing WeChat on <http://bbs.iosre.com>, you are welcome to join us.