Patching the iOS AIM client. A quick walkthrough

Written by MBA originally for aim-files.com

Note: Can be used as iOS reverse engineering documentation in general, as there is very little information on this online. For other applications, you must have a backend for which you are connecting to. This article will also contain information on iOS applications in general as a proper understanding is recommended.

Understanding the iPA

First, you must obtain the application of your choice. For this walkthrough I am using AIM 4.5.2.5, released around December 2010. The application must be cracked using tools like Clutch or Rasticrac otherwise you will not be able to install the application on any device (With the exception of iOS 6, which has a bug that allows you to bypass the Apple ID lock). Once you're certain you're working with the app you want and that it is cracked, you can begin patching. In this tutorial I will be using ImHex as my hex editor and Xcode as my pList editor. There are alternatives to these, for Win I recommend HxD for hex editing and notepad should do fine for pList editing, it's just not fancy and laid out neatly. (In this walkthrough I will skip the pLists as it very rare to need to do this)

Essentially, an iPA is a zip archive, within it are all the app's resources such as images, icons, nibs (compiled xibs), bundles, app binaries, executables etc. The directory structure of a generic iPA is as follows, Application.ipa > Payload > .app executable > Resources (the .app is a package essentially, Resources is not it's own separate folder). Now with the basics in mind, we shift our focus to the application binary, the extension-less file located within the app's resources. This is where all the compiled code of the application is kept, pretty much the heart of the app. It is usually the largest in file size and on Mac, it shows up as a command line executable:

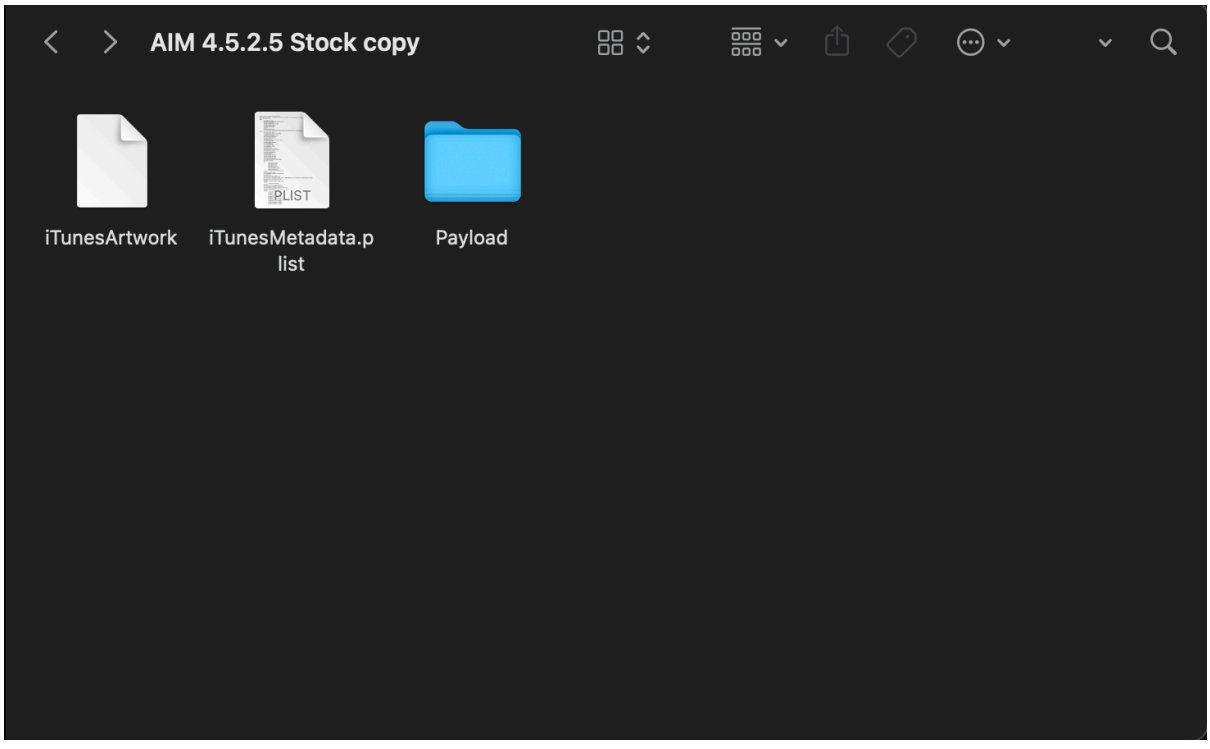
The applications binary is what will be edited, and pList with our version of AIM, not all applications will need this, with our hex editor of choice, with me using ImHex in this instance. In short, we will unpackaged the iPA, modify the binary, repackage it, and distribute.

It is worth noting that if the application will refuse to run/encounter severe issues if the binary is a different size than when before editing.

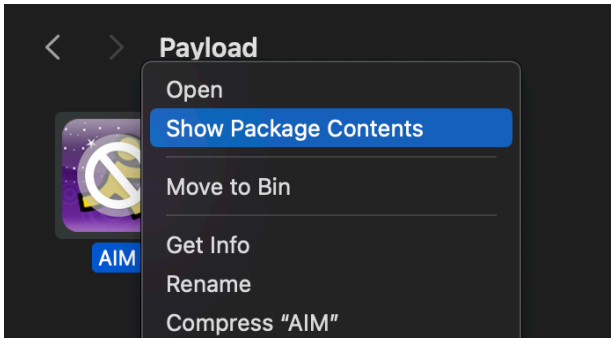
Step 1: Preparing the app

Make a working folder, this is where you should store all you are working on to avoid confusion. Once it is setup, you should place your iPA in there, it should be the only thing in that folder. We will now un-package the app, I have made this walkthrough very simple so anyone can do it.

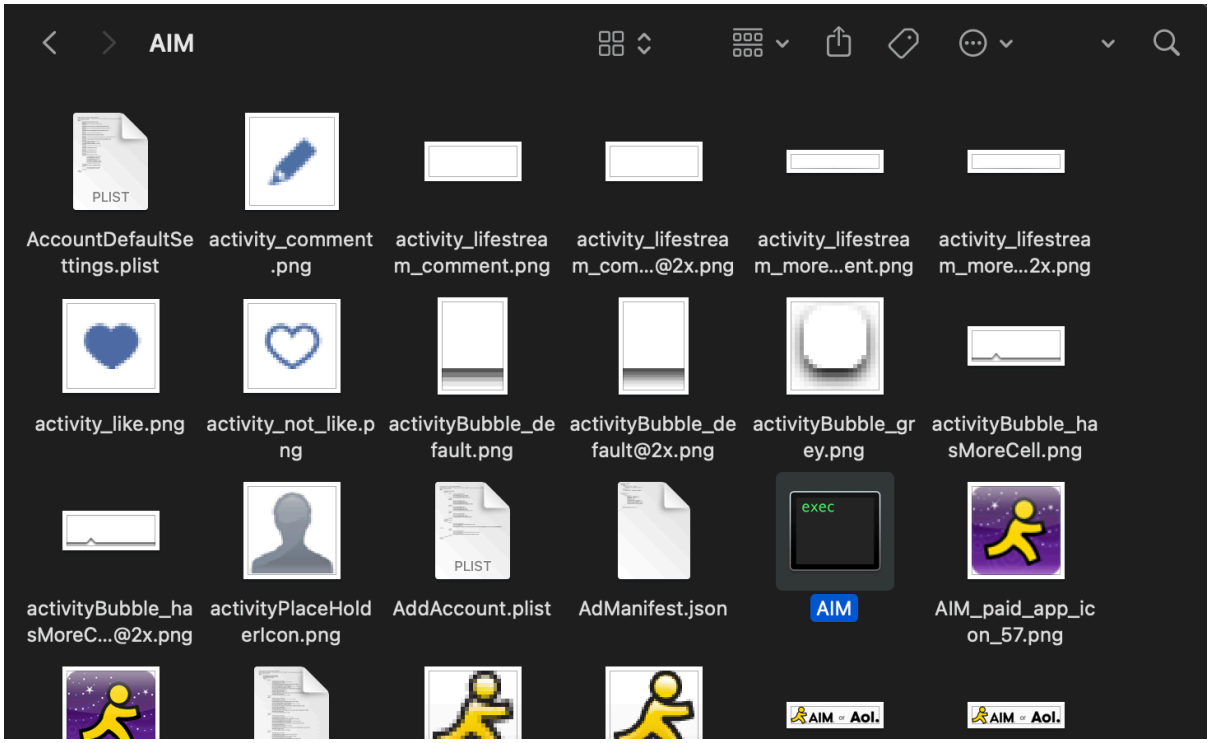
1. Change the extension of .ipa to .zip and unzip it



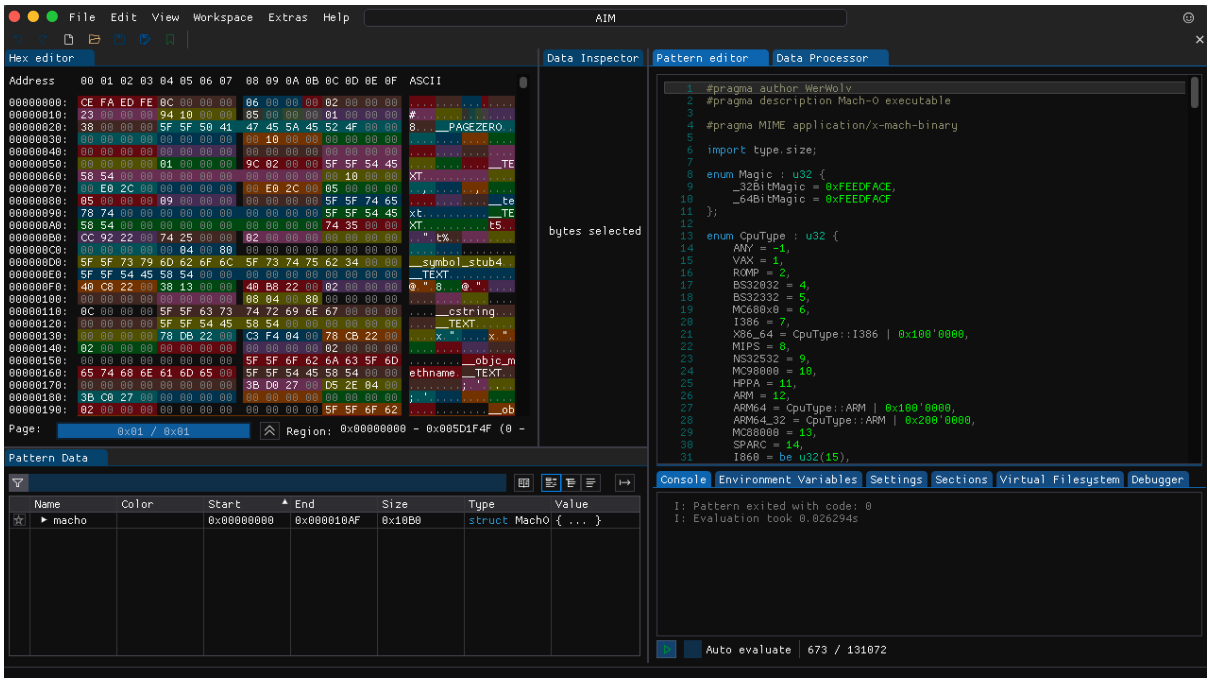
2. Within payload, display the .app package contents.



3. Find the app binary, it should not have an extension



4. Now, make a copy and paste it into your working folder. Then open this new copy up with your editor

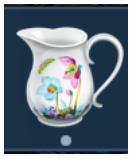


5. Now in this instance, the backends are setup for the servers we are connecting to so we don't need to worry about that. We need to replace http://api.oscar.aol.com with http://api.oscar.nina.bz and https://api.screenname.aol.com with https://api.screenname.nina.bz. Using your search feature find the servers you need to replace. URLS MUST ALWAYS BE THE SAME LENGTH OTHERWISE YOU WILL CHANGE THE BYTE SIZE.

6. Once edited, save the new binary and exit the editor. Now copy the newly saved binary and place it back into the .app resources. It should replace the old one. Now, compress the Payload folder, NOT THE ENTIRE FOLDER. Ignore the iTunesMetadata and iTunesArtwork files, JUST the payload. Once compressed, change the extension from .zip to .ipa

7. Now the app is ready to be installed on a jailbroken device with Appsync installed. On Mac, just drag the app into the phone in Finder.

Step 2: Testing the client



Now that we have made our new patched client, we need to test if it works or not. For this we need to check internet traffic using an HTTP proxy, in this instance, Charles.

You can use any generic connection instructions to setup Charles on your iOS device and PC/Mac. While connected, you should a log of all outgoing and incoming connections from your iOS device. Now open AIM and try logging in.

The connection should show up as successful, usually with 200 as the response to the clientLogin connection. If it shows up as unsuccessful, the client key may need to be validated first before it can successfully connect. If the connection completely fails however, check if you patched the binary correctly and verify correct internet settings. Note: webpages usually don't open while Charles is active. You may receive odd errors so make sure you disconnect the proxy once you're finished using Charles.

Step 3: Exception cases for iPhoneOS 3 (possibly 2 and 5 as well)

On iPhoneOS 3, and possibly 2 and 5 as well, hex patched applications refuse to open sometimes, crashing on launch. If this is the case here's a quick fix. I am not sure about Win, but on Mac, if home-brewed, you can install ldid using the following command in terminal:

brew install ldid

Now, this will install ldid. What ldid essentially is, is that it fake-signs the application binary so it can run properly. Once installed you want to copy the Pathname of your iPA's .app executable and run the following command:

ldid -S /path/to/Payload/Application.app

It should now fake-sign the app, you can now repackage the .ipa and test. This should fix the crash.