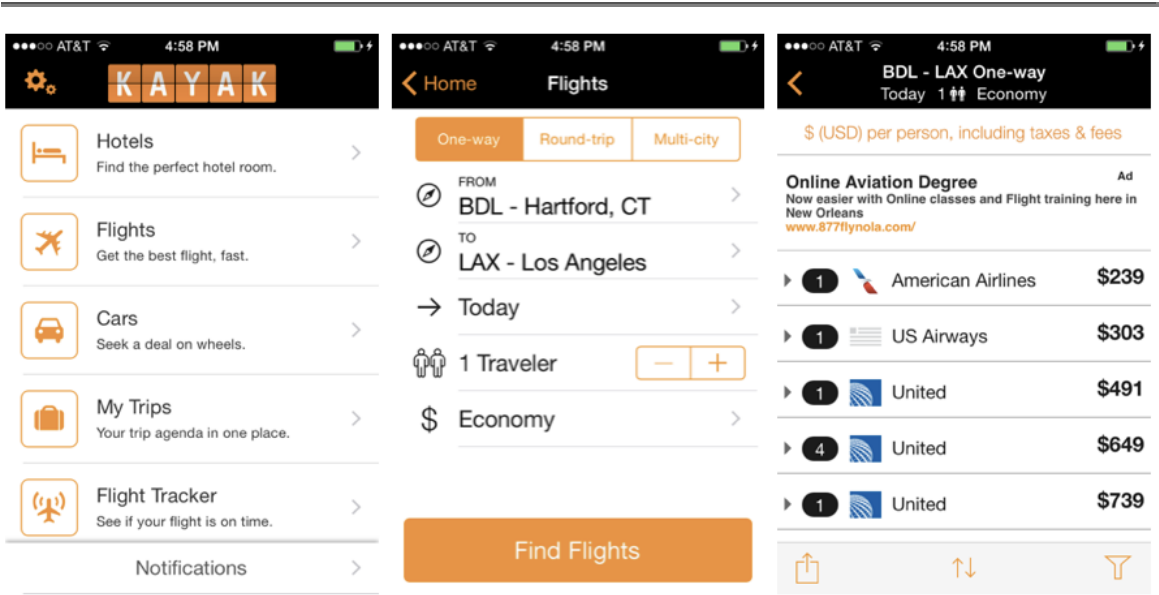


shubhro.com

# Reverse-engineering the Kayak app with mitmproxy

6-7 minutes



## Introduction

[Kayak](#)— the popular fare comparison web site— recently [discontinued](#) their API service. To the dismay of travel hackers, fare comparison APIs frequently [come and go](#). But that doesn’t mean we can’t hack together our own Kayak API.

Web scraping is the most common way to imitiate an API, but it’s vulnerable to small changes in the UI. On the other hand, *mobile* is an area where UI changes are often independent of the supporting server API. Developers often change the “look and

feel” of the mobile app, but seldom swap out the server *endpoints* from which data are obtained. For this reason, reverse-engineering mobile applications is a good way to expose APIs that we can exploit.

In this post, I’ll explain how I used [mitmproxy](#)— a popular network analysis tool— to reverse-engineer the Kayak mobile app. The result is an understanding of important server endpoints that can be accessed programmatically for personal use.

## Setup

My tool of choice for reverse-engineering mobile apps is mitmproxy because it presents network activity in a clean, hackable interface. I used Wireshark for a long time, but I actually couldn’t get the TLS decryption working and the X-Window interface was a little clunky. Mitmproxy functions as a proxy between the mobile device and the rest of the Internet. Any traffic targeting the phone must travel through mitmproxy, allowing us to analyze it.

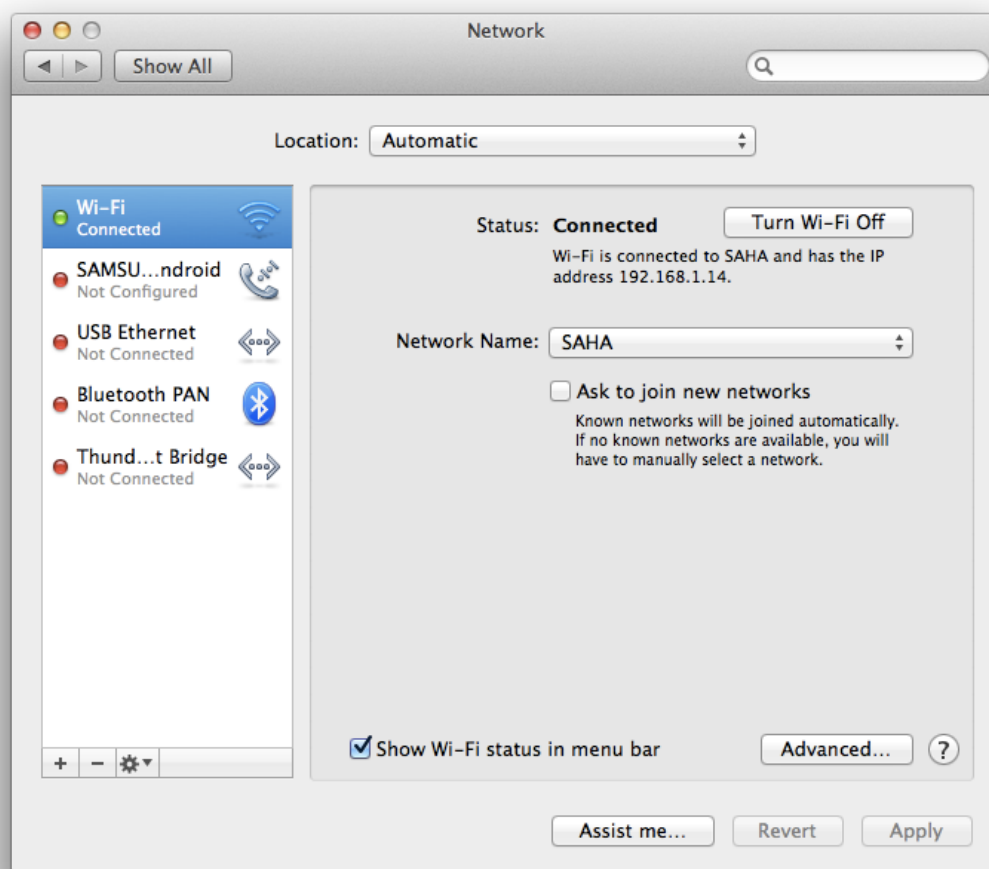
Getting started with mitmproxy is straightforward:

Run `mitmproxy` to generate certificate files in `~/.mitmproxy`. From that folder, get the `mitmproxy-ca-cert.pem` file onto your mobile device by emailing it to yourself, for example. Then follow certificate installation steps for [iOS](#) or [Android](#). Because I used the Kayak iPhone app, I’ll continue this tutorial with iOS.

Mobile apps often encrypt traffic to protect data integrity and confidentiality. Transport Layer Security (TLS) is a popular protocol for implementing this encryption. By installing the

certificate on a mobile device, we're enabling mitmproxy to decrypt its TLS traffic, which includes HTTPS requests and responses.

We need to configure the mobile device to use our computer's IP address as the proxy. On a Mac, we can find the IP address by opening up System Preferences > Network:



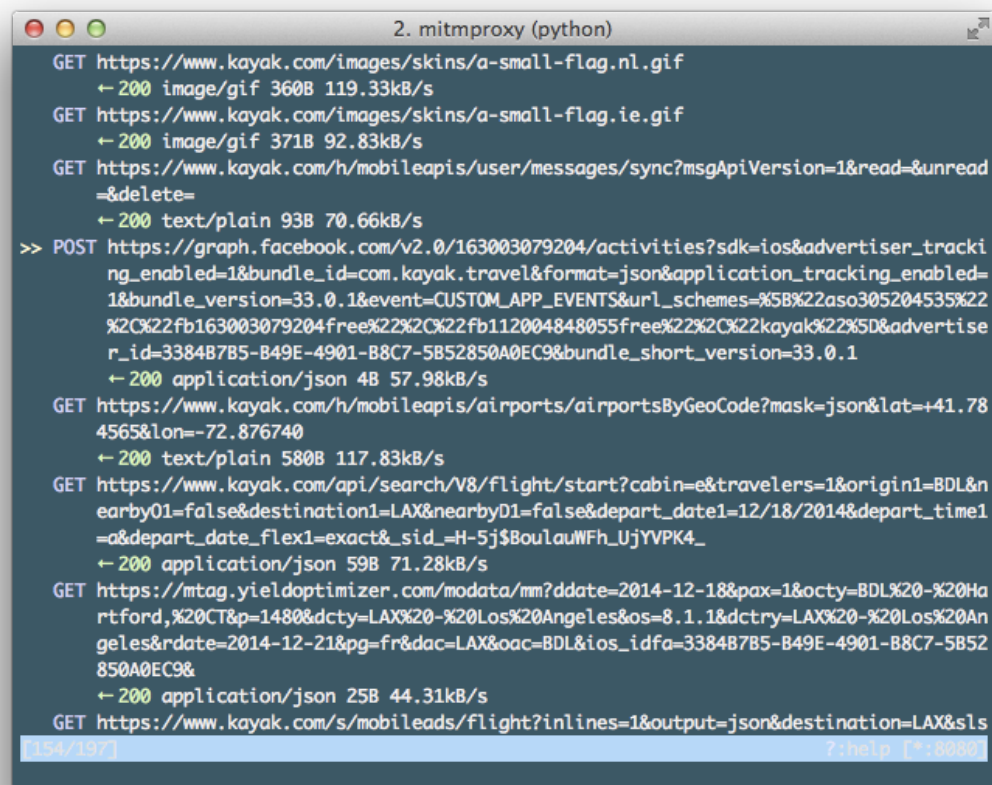
I configured my iPhone to use my Mac as the proxy by tapping Settings > Wi-Fi > [Network Name], and set HTTP PROXY to 'Manual'. Then I set the computer's IP address as the server and 8080 as the port.

## Recording Kayak network activity

We're now able to observe and save all of the iPhone's network activity, including those occurring over HTTPS. Let's record activity for the Kayak iPhone app by running `mitmdump -w kayak_flows.out` and tapping through the Kayak app as usual. In particular, navigate to Flights > From > Current location > Find Flights. After getting to the search results page, close the Kayak app and exit mitmdump by pressing `ctrl+c`. We know that, during this usage, the Kayak app must have communicated with the server to learn airports and prices.

## Browsing saved network activity

Having saved the network activity to a file, we can browse the results in a beautiful [curses](#) interface with `mitmproxy -r kayak_flows.out`.



```
2. mitmproxy (python)
GET https://www.kayak.com/images/skins/a-small-flag.nl.gif
← 200 image/gif 360B 119.33kB/s
GET https://www.kayak.com/images/skins/a-small-flag.ie.gif
← 200 image/gif 371B 92.83kB/s
GET https://www.kayak.com/h/mobileapis/user/messages/sync?msgApiVersion=1&read=&unread=&delete=
← 200 text/plain 93B 70.66kB/s
>> POST https://graph.facebook.com/v2.0/163003079204/activities?sdk=ios&advertiser_tracking_enabled=1&bundle_id=com.kayak.travel&format=json&application_tracking_enabled=1&bundle_version=33.0.1&event=CUSTOM_APP_EVENTS&url_schemes=%5B%22aso305204535%22%2C%22fb163003079204free%22%2C%22fb112004848055free%22%2C%22kayak%22%5D&advertiser_id=338487B5-B49E-4901-B8C7-5B52850A0EC9&bundle_short_version=33.0.1
← 200 application/json 4B 57.98kB/s
GET https://www.kayak.com/h/mobileapis/airports/airportsByGeoCode?mask=json&lat=+41.784565&lon=-72.876740
← 200 text/plain 580B 117.83kB/s
GET https://www.kayak.com/api/search/V8/flight/start?cabin=e&travelers=1&origin1=BDL&nearby01=false&destination1=LAX&nearby01=false&depart_date1=12/18/2014&depart_time1=a&depart_date_flex1=exact&_sid=H-5j$8oulauWFh-UjYVPK4_
← 200 application/json 59B 71.28kB/s
GET https://mtag.yieldoptimizer.com/modata/mm?ddate=2014-12-18&pax=1&octy=BDL%20-%20Hartford,%20CT&p=1480&dcty=LAX%20-%20Los%20Angeles&os=8.1.1&dctry=LAX%20-%20Los%20Angeles&rdate=2014-12-21&pg=fr&dac=LAX&oac=BDL&ios_idfa=338487B5-B49E-4901-B8C7-5B52850A0EC9&
← 200 application/json 25B 44.31kB/s
GET https://www.kayak.com/s/mobileads/flight?inlines=1&output=json&destination=LAX&sls
[154/197] 7:help [*:8080]
```

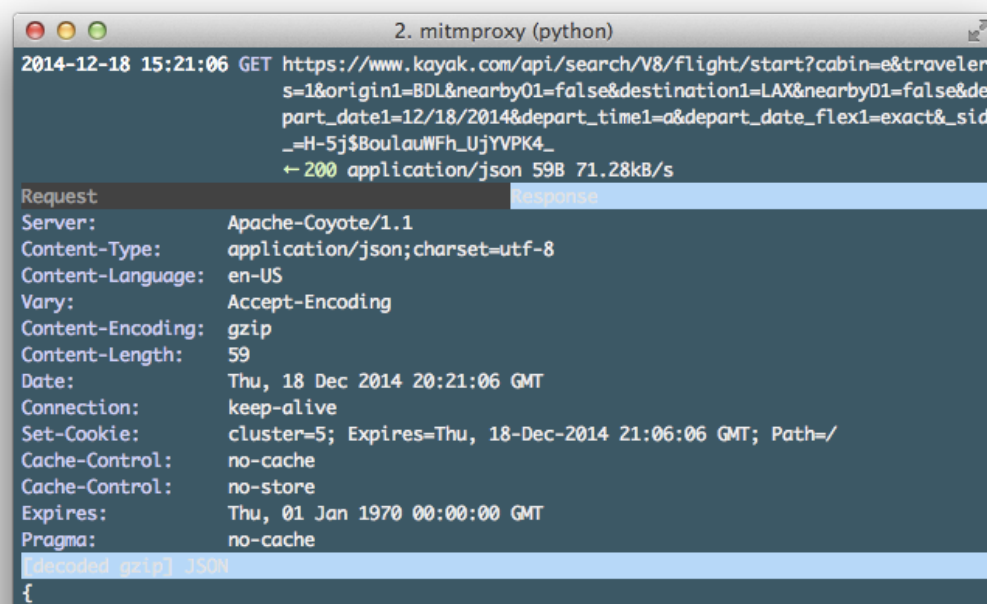
As we flip through the requests with the arrow keys, the first to jump out is:

```
GET https://www.kayak.com/k/authajax
/?action=registermobile&uuid=[UNIQUE
IDENTIFIER]&hash=[A HASH]&model=iPhone4,1&
appid=kayakfree&os=8.1.1&msgApiVersion=1&as=0&
appdist=adhoc&prefix=`
```

I hit <Enter> then <Tab> to view the server response. We discover fields for status, uid, token, sid, and bogus.

Great! Let's make a note of the uid, token, and sid fields because they might be used in later requests. Back in the request list, we continue to look for interesting URLs. Here's one:

```
GET https://www.kayak.com/api/search/V8/flight
/start?cabin=e&travelers=1&origin1=BDL&
nearby01=false&destination1=LAX&nearbyD1=false&
depart_date1=12/18/2014&depart_time1=a&
depart_date_flex1=exact&_sid_=[SID VALUE]
```



```

    "error": false,
    "searchid": "keECDKuMsc"
  }

```

[156/197] ?:help q:back (\*:0000)

This request generates a `searchid` that will probably be used shortly to uniquely identify our choice of airports, dates, and other preferences.

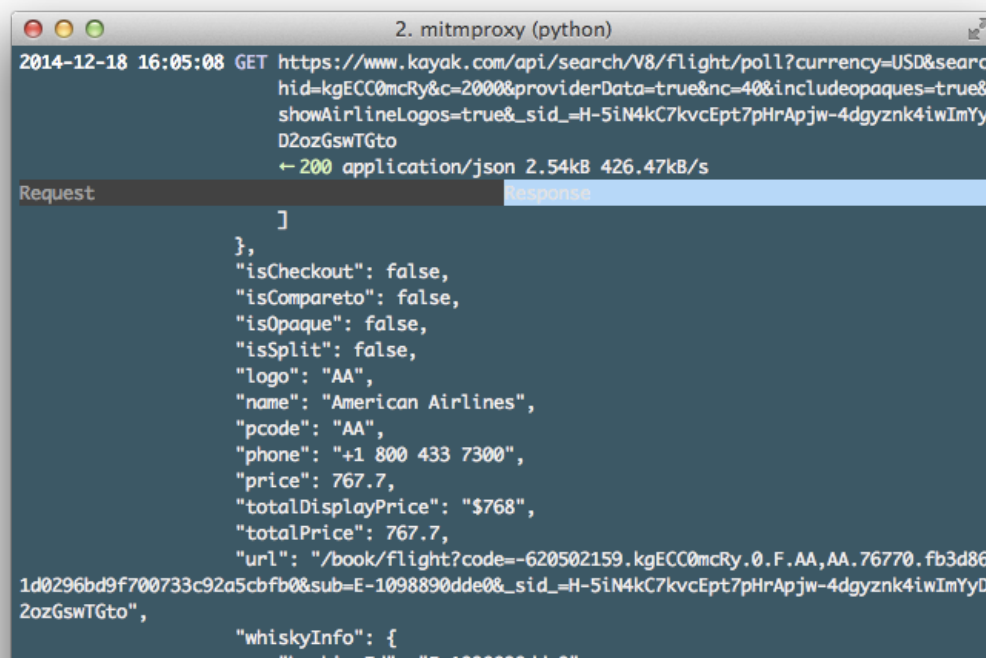
Indeed, that's the case! Our attention is drawn to:

```

GET https://www.kayak.com/api/search/V8/flight/poll?currency=USD&searchid=[SEARCH ID]&c=2000&providerData=true&nc=40&includeopaques=true&showAirlineLogos=true&_sid_[SID VALUE]

```

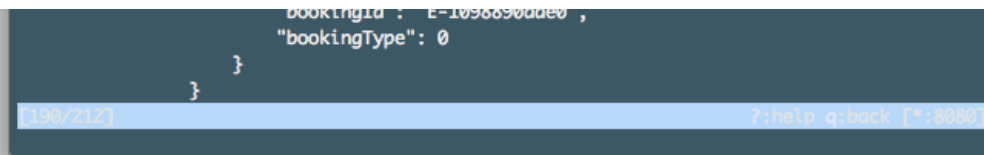
This request includes our `searchid` and `sid` from earlier, and the response is a large JSON object that includes airline names, prices, contact, booking URLs, and more. This is exactly the data we're looking for.



2. mitmproxy (python)

2014-12-18 16:05:08 GET https://www.kayak.com/api/search/V8/flight/poll?currency=USD&searchid=kgECC0mcRy&c=2000&providerData=true&nc=40&includeopaques=true&showAirlineLogos=true&\_sid\_=H-5iN4kC7kvcEpt7pHrApjw-4dgyznk4iwImYyD2ozGswTGto  
← 200 application/json 2.54kB 426.47kB/s

Request	Response
	<pre> ] }, "isCheckout": false, "isCompareto": false, "isOpaque": false, "isSplit": false, "logo": "AA", "name": "American Airlines", "pcode": "AA", "phone": "+1 800 433 7300", "price": 767.7, "totalDisplayPrice": "\$768", "totalPrice": 767.7, "url": "/book/flight?code=-620502159.kgECC0mcRy.0.F.AA.AA.76770.fb3d861d0296bd9f700733c92a5cbfb0&amp;sub=E-1098890dde0&amp;_sid_=H-5iN4kC7kvcEpt7pHrApjw-4dgyznk4iwImYyD2ozGswTGto", "whiskyInfo": {   "bookingId": "E-1098890dde0" } </pre>



```
    "bookingId": "E-10988900000",  
    "bookingType": 0  
  }  
}  
[190/212] 7:help q.back [*:0000]
```

## Putting it all together

Let's put these observations together to discern a basic API:

- Think of the `uuid=[UNIQUE IDENTIFIER]&hash=[A HASH]` fields to be like an API key and secret generated when running the mobile app.
- Generate an `sid` at: `https://www.kayak.com/k/authajax/`
- Generate a `searchid` at: `https://www.kayak.com/api/search/V8/flight/start`
- Retrieve prices at: `https://www.kayak.com/api/search/V8/flight/poll`

You can check out a complete basic client on [GitHub](#).

The results presented here are far from a complete API, but I hope this tutorial demonstrates the power of reverse-engineering mobile apps. Tools like mitmproxy helps us obtain a level of understanding previously prohibited by the locked-down nature of mobile operating systems. Given how young the mobile space is, there's never been a better time to do some exploring.

To read more about researching mobile apps with proxies, check out [Extracting My Data from the Microsoft Band](#), [Yik Hak](#), and [What They Know Mobile](#).

December 2014

[Hacker News](#) · [Reddit](#) · [Hackaday](#)