ECS 153: Computer Security Winter 2014

Mobile Ad Library Requests and Vulnerabilities

Prof. Hao Chen *Authors: William Keller, Henry Walton*

1 Introduction

In-app ads have rapidly become one of the most important revenue streams in the mobile world. As these ads have become more common, so too has their sophistication risen. With advanced finger printing techniques, ads can increasingly identify users and target their content with ever improving precision. This has raised the issue of information privacy, and how much users are sharing with these ad services. In an attempt to better answer this, we captured the ad library requests of Android and iOS and explored them to see just how much data, and what data, was being collected.

2 Setup

2.1 Android

To analyze the Android ad library requests, we used a combination of two applications to capture the HTTP packets and examine them respectively. To capture the packets we used tPacketCapture on our Android device, which works by routing packets through a virtual VPN on the device, and then transferred them to a computer where we could further examine them with Wireshark. Though we were able to set up a proxy much like we did for iOS, tPacketCapture proved useful as it could filter the captured packets to a single Android app.

2.2 iOS

iAd is a response from Apple to security concerns regarding standard mobile ad tracking as we found on Android. Before iOS 4, ads were unregulated on apps in the App Store. Apple found advertisers/developers were lifting the UUID assigned to the device, and using that to (very effectively) fingerprint users. With an ideal fingerprint like a UUID, advertisers found it easy to aggregate data from multiple sources and generate a more complete profile of the user. Instead of introducing stringent and hard to enforce rules on fingerprinting, Apple instead purposely streamlined the fingerprinting process after making concessions to protect user privacy. iAd allows opting out of tracking, but more importantly, it allows users to generate a new random tracking id at will. This way, users can retain the benefit of having relevant ads while allowing themselves to disassociate from their tracking id whenever privacy concerns arise.

While the unregulated ad networks on android gace up their data to simple packet capture, iAd's message exchange is encrypted with HTTPS. This required us to construct a man-in-the-middle "attack" to perform deep packet inspection. We began by attempting to write an SSL re-signing socks proxy in python, but quickly decided that was too hard, and instead used the Fiddler debugging proxy published by Telerik. Fiddler2 was installed on a Windows 8 desktop, and and then used the Fiddler Certificate Maker plugin to generate a new X.509 root certificate. Then, we transferred the certificate to a jailbroken iPhone running iOS 7.0.6 (restored as closely as possible to

its un-jailbroken state). After adding our root certificate to the iPhone's trusted keystore, we configured Fiddler to use the same root certificate to decrypt incoming HTTPS connections using the key, then resign them itself before forwarding the packets to the internet at large. In this way, we were able to inspect the contents of SSL packets while still keeping an active connection (through fiddler) to the destination.

We hit immediate trouble inspecting the captured iAd data. First off, the packets were encoded in a fashion unfamiliar to Fiddler. The iAd packets are SSL encrypted gzipped POST requests. Gzipped POST requests, at least in the situations Fiddler was designed for, are uncommon. We attempted a fix by modifying Fiddler's internal request handling scripts:

```
static function OnBeforeRequest(oSession: Session) {
    if (oSession.requestBodyBytes != null \&\& oSession.requestBodyBytes.Length>0
        && oSession.oRequest["Accept-Encoding"] == "gzip, deflate"){
    oSession.requestBodyBytes = Utilities.GzipExpand(oSession.requestBodyBytes);
    oSession["Content-Length"] = oSession.requestBodyBytes.Length.ToString();
    oSession["Content-Encoding"] = "text/ascii";
    }
}
```

Unfortunately, while this handily decoded test gzip POST request bodies, it couldn't crack the iAd packets. An example of the unencrypted data follows (these lines were extracted from binary data and not all characters rendered correctly):

```
iPhone5,2iPhone OS 7.0.6 O
adsheet.client.session-durationcom.jminteractive.Doge-Wallet  6 A
adsheet.launched <
adsheet.client.connectedcom.jminteractive.Doge-Wallet
```

The binary blobs suggest the data is encoded further, but the occurrences of plaintext strings suggest it isn't compressed heavily or encrypted again. After many, many failed attempts writing our own GZIP headers and checksums, we gave up on that endeavor, and decided to work with what we had of the data (but not after accidentally pushing a java IOException to the iButt key value store by modifying packets on the fly). While writing the end of this report, we concluded that this was very probably a binary .plist file, but that was not tested.

3 Observations

3.1 Andoird

In line with its open nature, there is no centralized ad service for Android. Instead, there are numerous different services which compete for ad space in apps. These services vary in methodology, structure, and trustworthiness leaving the choice of which to use to the developers' volition. By far the most popular are the Google services AdMob and Analytics. A 2012 paper on the security risks of in-app ads found that over half of the apps they surveyed used Google's ad services and this was reflected in our own experiences. [1] It proved difficult to find even a handful of apps that didn't use AdMob or another Google service. However, because of the huge presence of Google's services, and the fact that this presence means they are held to high standards of behavior, many

[1]Book, Theodore, Adam Pridgen, and Dan Wallach. "Longitudinal Analysis of Android Ad Library Permissions." (2013): n. page. Web. 22 Mar. 2014. <http://arxiv.org/pdf/1303.0857.pdf>.

of the smaller services can practice less sanctioned behavior. For our research, we examined the packets from several apps using a variety of ad services.

One of the first apps we examined, a game called Trainyard Express, uses Google's AdMob service. Upon examining the requests we found a number of metrics passed as parameters that could be used to fingerprint the device for cross app targeted advertising. In addition to basic device information such as OS version, hardware model, app version, and carrier, we also found a device id parameter. Further examination revealed this to be a unique id generated for each Android device the first time it connects to AdMob. This device id is shared across apps and can be used to target advertising through the user's usage habits.

Another notable app we examined was the restaurant reservation app, OpenTable. What made OpenTable stand out wasn't any particular security risk or shared information, but the multitude of ad requests we found. We were at first convinced we were accidentally capturing packets from multiple apps, but as it turns out, OpenTable simultaneously uses three different ad services. We found packets from MoPub, Flurry, and AdMob among OpenTable's traffic. Though MoPub and Flurry proved trustworthy, each passing information similar to AdMob and employing similar device id systems, this still struck us as a somewhat questionable practice. Because these ad libraries automatically receive the same permissions as their host apps, a user never has the opportunity to opt out of having their information shared, nor are they made aware of to whom their information is sent. While this sharing has become largely accepted, doing so with three different services is definitely not the norm. Probably the most significant app we explored was a recipe navigator, BigOven. Though BigOven's ad library, admarvel, didn't directly share more information than any of the libraries we examined, it revealed potential vulnerabilities in the Android ad system. Among the http requests, we found a suspicious looking request to *http://uac.advertising.com/wrapper/aceUAC.js*. Though this request proved to be benign, it revealed some very worrying potential vulnerabilities.

Android ads usually work by essentially creating a miniature web browser in the host app that the ads can be loaded into. In order to gain an advantage on its competitors, Mobclix provides additional functionality to its ads by binding several Android API calls to javascript functions available to the ads. These calls give ads access to most of the device's sensors, such as gps and camera, as well as user information such as phone and contact data.[2]. These calls can be combined with Javascript or a downloaded jar to essentially allow an ad to fetch and run arbitrary code.

3.2 iOS

To decide if iAd is playing as cleanly as it claims, we examined five different iAd use cases and inspected their packets: an app with iAd initializes, iAd preforms metric gathering, the phone resolves a zip code, and the user refreshes their tracking id.

When an app using iAd cold-started, we saw a request that was responded to with a list of assets (extracted from response body that was partially binary data):

```
http://iadctest.qwapi.com/adunits/c0/3c/ed/7297fa93-ee7a-40e2-9fce-82f6ff32b8c6/1.ad/Creatives/
    f3a6570f-df1c-4b5d-bf9f-6e6138c88d88_Creative/HTMLBanner/WebArchive@2x.webarchive
http://iadctest.qwapi.com/adunits/e7/ff/59/9a71b09a-5d69-4e5d-9d1d-37d8bcde089d/1.ad/Creatives/
    7cdcfa75-b33d-493d-bbef-d04a96f31f66_Creative/HTMLBanner/WebArchiveiPhone@2x.webarchive
http://iadctest.qwapi.com/adunits/77/12/55/a14cfd11-ef11-434d-ad3c-b99e95662677/1.ad/Creatives/
    0f01d86c-9374-442b-a7e6-69b2f34e43a9_Creative/HTMLBanner/WebArchive@2x.webarchive
```

[2]Michael Grace, Wu Zhou, Xuxian Jiang and Ahmad-Reza Sadeghi, "Unsafe Exposure Analysis of Mobile In-App Advertisements," Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2012), Tucson, Arizona, April 2012.

After confirming the domain is administrated by apple, we concluded that this was iAd pinging different CDNs to find a fast connection.

Next, we examined what metrics iAd tracks for ad service. After sending a request:

```
POST https://iadsdk.apple.com/adserver/1.7/metrics/aggregate HTTP/1.1
User-Agent: iPhone5,2; iPhone OS 7.0.6; com.apple.AdSheet; 143441-1,21
storefront: 143441-1,21
```

The phone recieved a response with the following keys:

```
iPhone5,2;iPhone OS 7.0.6;0
network.WiFi.resource.bandwidth
network.WiFi.resource.latency
network.WiFi.web.bandwidth
network.WiFi.web.latency
adsheet.launched
resource-cache.miss
resource-cache.stored
url-cache.disk.hit
url-cache.disk.lookup
url-cache.disk.miss
url-cache.memory.add
url-cache.memory.hit
url-cache.memory.lookup
url-cache.memory.miss
adsheet.client.connected;com.jminteractive.Doge-Wallet
```

This gives us an idea of what iAd is tracking. While it seems to be mainly data related to efficient distribution of assets to users, it still leaks some information that can be used to fingerprint and potentially locate users, even if all other Apple tracking is disabled. The WiFi bandwidth/latency and the URL-cache entries could allow Apple to place you between home and work, if you had different connections speeds (though this seems expectionally unlikely).

Before it's displayed, the ad is fetched from iadsdk.apple.com:

```
GET /adunits/c0/3c/ed/7297fa93-ee7a-40e2-9fce-82f6ff32b8c6/1.ad/Creatives/
   f3a6570f-df1c-4b5d-bf9f-6e6138c88d88_Creative/HTMLBanner/WebArchive@2x.webarchive HTTP/1.1
```

The asset is served as an RDF directly from apple. There is little chance for information leakage here.

One of the more interesting things we saw was iAd's (or core location) attempts to resolve a ZIP code for targeting local ads:

```
POST https://gsp10-ssl.apple.com/hcy/pbcwloc HTTP/1.1
Host: gsp10-ssl.apple.com
User-Agent: locationd/1613.5.1 CFNetwork/672.0.8 Darwin/14.0.0
```

The response returned carries information about mapped MAC addresses:

```
98:fc:11:69:1a:a0 [snip data]
58:6d:8f:5b:29:99 [snip data]
...
```

As this information never leaves apple, and is sent over SSL, this also appears to be okay.

Then, we changed tracking id's several times and inspected the packets:

```
POST https://iadsdk.apple.com/adserver/1.7/optout/optout_optin HTTP/1.1
Host: iadsdk.apple.com
User-Agent: iPhone5,2; iPhone OS 7.0.6; com.apple.AdSheet; 143441-1,21
```

Response:

```
 5 <l fW8  k { F  |
[snip]
```

While we couldn't inspect this data directly, we saw several parts of the returned data changing every request. As there is no encryption avalance effect, this suggests that Apple is serving us a new tracking id (or, at least purporting to).

Opting out responded with a plaintext XML plist:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>identities</key>
  <array>
  </array>
  <key>status</key><integer>0</integer>
</dict>
</plist>
```

Before hitting init.itunes.apple.com to (presumably) generate a new id.

Finally, for comparison, we tracked an ad in the jailbroken Cydia app store. As expected, we see standard cookie based fingerprinting from online ad providers:

```
Set-Cookie: demdex=46258853564644915241320700872135223423;Path=/;Domain=
    .demdex.net;Expires=Thu, 10-Mar-2016 17:52:21 GMT
```

While this particular example is pretty behaved, this request could also include info like the iPhone's UUID. The jailbroken software has zero protection beyond its reputation.

4 Conclusions

Due to their fundamentally different environments and developments, iOS and Android handle mobile ads in very different fashions. Android, being an open platform, is at the mercy of its developers. The huge number of different ad libraries have varying levels of stability and privacy and due to the fact that the ad libraries are granted the same permissions as their host apps, the user has to trust the ad services as much as the app developer. This is partially remedied by Google's massive presence and the prevelance of AdMob, but a better solution to ad library permissions will hopefully arise in the near future.

On the other hand, Apple's iAd service is the single option for ad services on iOS. Apple has total control over the permissions of in-app ads as well as the fingerprinting techniques involved. Our examinations revealed a reliable and responsible system, though ultimately one that can only be as trusted as much as one trusts Apple.