Security in Client-Server Android Apps

Mike Pappas, Raluca Ifrim, Ben Weissman, Chris Tam

May 12, 2014

MIT, Cambridge, MA

Abstract

We explore security vulnerabilities in a series of Android apps, in particular with regard to information revealed to a passive eavesdropper. We additionally suggest a number of simple mechanisms to protect against similar attacks in the future.

1 Introduction

There are many different apps on the Android app store that claim to provide various secure benefits. These apps purport to provide total anonymity to the user, or else wish to restrict a service they've created to only be accessed through the app itself. Our project sought to break these guarantees by simply analyzing the data packets gathered through a passive man-in-the-middle attack using a network proxy. We targeted four different apps. Whisper and Ask.fm both claim to provide anonymity to the users, although we will show that Whisper in particular fails to meet this guarantee. Additionally, Shazam and Tinder both restrict the freedom of the user to fully exploit the power of their back-end services, and, again, we find that we are able to bypass these restrictions, especially in the case of Tinder. These apps give us strong examples of both well designed and poorly designed security systems for Android apps, and allow us to highlight the critical elements that any app designer needs to consider when attempting to build a secure app.

2 Procedure

Our procedure for analyzing these apps was very similar across the board, although a few app-specific techniques that were used will be explained in the corresponding sections. However, we were able to use consistent attack strategies mostly because we wanted to see how much information we could get using the most passive attacks. Our basic strategy was to listen to the packets being sent to our device and attempt to uncover any information that we should not have had access to.

We used the *Genymotion* Android emulator[1] running on our computer to download and run our apps, and the web debugging proxy *Charles*[2] to listen to the traffic coming from the apps. In order to allow Charles to record requests and responses, we configured our emulator to use our computer as a proxy. Charles listened to the packets posted from and received by the app, allowing

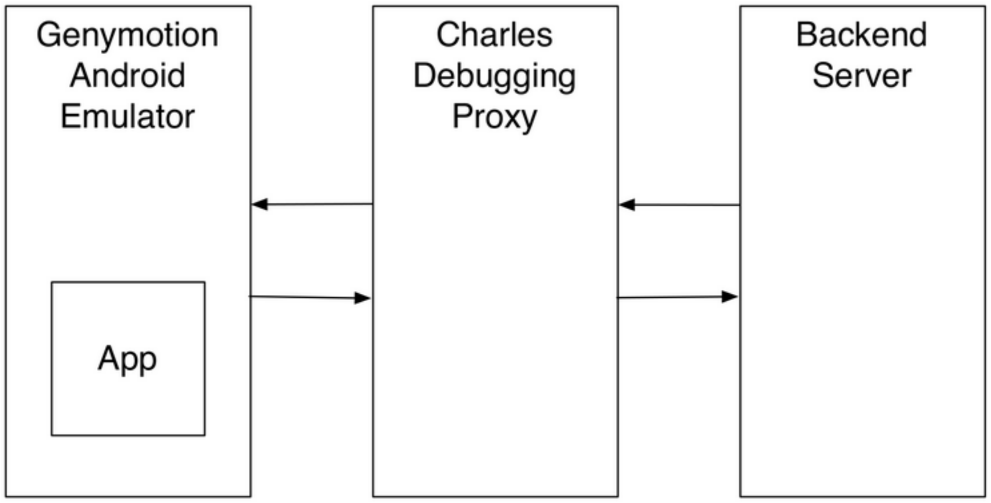

Figure 1: Charles records communication between the app and the backend.

us to look into the packets directly, rather than being restricted to seeing only the data from the packets that the respective apps chose to display. (See Figure 1.)

In order to be able to observe HTTPS traffic, we installed the Charles root CA certificate on the emulator. This allows the app to verify certificates that were signed by Charles, allowing the TLS handshake to happen between the app and Charles. Charles then forwards the app's request to the backend using normal HTTPS protocols.

We were thus able to get the details of every request our apps made. We filtered out as much irrelevant information as we could, such as ads, and then scanned the requests for information that would break the guarantees our applications made.

3 Tinder

Tinder is a dating app that allows users to find other nearby Tinder users. The app shows the user a picture of another nearby Tinder user, and the user can then choose whether they want to skip this user, or if they're interested in the user. If they're interested in the user, it sends a connection request to the other user, who has to confirm to establish mutual interest, at which point the app facilitates communication between the users so they can meet in person. However, if a user rejects a match suggested by Tinder, that match will never be shown again, and there is no way to go back and view a rejected match again. This crucial restriction is what has made Tinder so successful: users are more likely to accept a match because of the pressure of the decision being final, as opposed to traditional dating sites, where users typically browse hundred of profiles before selecting just a few.

While Tinder bills itself as an app that facilitates all sorts of interactions, it has primarily become used for casual romance and hook-ups. Because of this, many users are interested in ways to game the system and give themselves a competitive advantage, hoping to find better matches.

We set out to determine whether the key constraint of Tinder (the inability to reject a match and then go back to it) could be compromised.

3.1 Client-Server Communication Protocol

First, the client application communicates with Facebook to authenticate the user. It then submits the Facebook authentication server to `https://api.gotinder.com/auth` and receives a Tinder authentication token in return, which is included in the HTTP headers of all subsequent requests.

The client application then uses the device GPS to locate the user and submits this location to `https://api.gotinder.com/user/ping`. This location is used in tandem with social graph data from Facebook to match the user with other Tinder users.

Finally, the client application submits a request to `https://api.gotinder.com/user/recs` to get a list of matches from Tinder. Here is a sample request made by the application:

```
POST /user/recs HTTP/1.1
app_version: 631
platform: android
If-Modified-Since: Wed, 02 Apr 2014 01:48:20 GMT+00:00
User-Agent: Tinder Android Version 2.2.1
X-Auth-Token: 880b7db7-54fc-42f7-86a0-530a5f54b247
os_version: 19
Content-Type: application/json; charset=utf-8
Host: api.gotinder.com
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 12

{
  "limit": 40
}
```

Here is the response to the above request. Note that it has been abbreviated to remove redundant information by replacing similar records with `[... x20]` to indicate how many similar records were present, and all identifying information has been replaced with `REDACTED`.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 07 Apr 2014 15:40:39 GMT
ETag: "-1899104168"
Server: nginx/1.1.19
X-Powered-By: Express
Content-Length: 15345
Connection: keep-alive
```

```
{
  "status": 200,
  "results": [
     {
       "distance_mi": 24,
       "common_like_count": 0,
       "common_friend_count": 0,
       "common_likes": [],
       "common_friends": [],
       "_id": "REDACTED",
       "bio": "",
       "birth_date": "1978-02-19T00:00:00.000Z",
       "gender": 1,
       "name": "REDACTED",
       "ping_time": "2014-04-07T15:14:28.250Z",
       "photos": [
         {
           "id": "REDACTED",
           "main": "main",
           "shape": "center_square",
           "fileName": "REDACTED.jpg",
           "extension": "jpg",
           "processedFiles": [
             {
               "width": 640,
               "height": 640,
               "url": "REDACTED.jpg"
             },
             [... x3]
           ],
           "url": "REDACTED"
         },
         [... x4]
       ],
       "birth_date_info": "fuzzy birthdate active, not displaying real birth_date"
    },
    [... x40]
}
```

Looking at this request, we can observe some choices Tinder makes that are good for its security, and others that compromise its security.

First of all, Tinder uses HTTPS for all communication to prevent eavesdroppers from compromising communication between the app and the server. In addition, we can see that Tinder fuzzes birth dates so it can reveal the approximate age or a user without revealing the exact birth date. Finally, the server computes the distance between the two users and returns only the distance, keeping the exact location of the other user private.

However, there is a key flaw in this communication protocol: the Tinder server returns a large number of matches (up to 40) at once! This fundamentally compromises Tinder's ability to enforce the rule that users can only see one match at a time. A dishonest client could show all of the returned matches at once and allow the user to browse through the results, similar to a traditional dating site. This would allow the user to make a more informed decision about the matches, granting a competitive advantage.

3.2 Suggestions for Improvement

Overall, Tinder has made good security decisions. It authenticates via a trusted third-party (Facebook), reasonably avoids leaking irrelevant personal information, and communicates over a widely-used, secure protocol (HTTPS).

However, it unnecessarily allows dishonest clients to circumvent its restriction on browsing multiple users. This could easily be avoided by simply sending a single match at a time, and requiring that a user accept or reject the match before sending another match. Assumedly, Tinder chose not to use this technique because it would require many more requests to the server, increasing latency. However, drastically reducing the number of matches shown at once could limit the efficacy of a dishonest client. For example, Tinder could initially send two matches, and then send one more each time a match is accepted or rejected, so the app always has information on two potential matches, one being shown and one held on reserve. The next match would be stored in memory so it could be shown immediately, and the following one could be loaded from the server while the user is considering their current match. The app already submits a request to `https://api.gotinder.com/updates` every few seconds, so while this would increase traffic to the server somewhat, it would not be a unreasonable increase.

4 Shazam

Shazam is a music identification application. When users start the app, the app will listen for a few seconds to any songs that are playing in the background, process the information, and connect to the Shazam backend, which will attempt to identify the song.[3] Like Tinder, Shazam's security model includes the a heavy restriction on how the user can utilize its back-end service. In this case, even though Shazam has published a paper on how its song identification algorithm works[4], it does not allow anything but the app to communicate with the back end, nor does it wish to identify songs by any means other than listening to short snippets of them. We first attempted to see if we could get around their primary guarantee and find a way to build a third party app that would perform the same service using the Shazam backend. Failing that, we also wanted to see if we could trick the app into behaving in unexpected ways; for example, by getting it to identify an mp3 instead of live audio.

4.1 Song Identification

Shazam uses a specific algorithm to identify each song. In brief, the app doesn't actually record music, but rather creates a spectrogram from the audio input, graphing time, frequency, and intensity. From this, the algorithm identifies "peaks," points of high intensity, and uses these to create a "fingerprint" for the selection. The backend stores previously generated fingerprints in a hash table mapping fingerprints to identifying information such as song title and artist. The app sends the Shazam backend the fingerprint that it has created and the backend responds if a match is found.

These fingerprints as well as additional information can be seen by observing Shazam traffic. The POST request from the application is formatted as follows.

```
POST /orbit/DoAmpTag1/android HTTP/1.1
Connection: Close
Content-Language: en_US
X-Shazam-AMPKey: 657780AVsvPvP337FMWYfKpO6XyGd+h22xz+bXEA4wrwQummw
       ElxCRiEFoiLIh3mall6a8wr7EiaYVeyHzYAitoA7YvIjLp5lRuXG1dXfOcLMBTM/ckVhR
                 7NzuLudqefGAGOOWTzO8Z8q/ULsDZXIc+DFS184UlG2HwAAQqHdoVZH33wo
                         /dO9ru+tF+jCTTqQbaO4q
content-type: multipart/form-data;boundary=A3r_ISAAC_eQeY2Bh
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.3; Custom Phone 7 - 4.3 - API 18 -
          1024x600 Build/JLS36G)
Host: spike.shazamid.com
Accept-Encoding: gzip
Content-Length: 2747
```

This POST request contains several hex-encoded fields, most relevantly: *deviceId, cryptToken, sampleBytes*, and *tagId*.

The backend sends a response formatted similarly to the example message below.

```
<shazamResponse xmlns="http://orbit.shazam.com/v1/response">
<doRecognition1>
<requestId>1398370618192</requestId>
<tracks />
<retry>5000</retry>
</doRecognition1>
</shazamResponse>
```

In this case, the backend failed to identify the request; in successful cases, the response contains many more descriptive fields such as song title, cover art, artist, and information about add-ons such as a Youtube video for the song.

4.2 Security

We reasoned that it would be very difficult to directly build a third party song identification app without having the same implementation of Shazam's algorithm. However, if we were able to generate valid fingerprints, we would be able to use their service by sending the Shazam backend our valid fingerprints from our own third-party app. Most of the other identifying information we would need in the POST request, such as deviceId and cryptToken, remain constant over multiple requests, and could therefore have been collected by simply analyzing a few other Shazam POST requests. The sampleBytes field, which contains the fingerprint, is the one for which we would need to replicate Shazam's proprietary algorithm. Finally, the tagId field just sets an ID for each tag that the user attempts to create and should be straightforward to forge, although we would have to perform additional experiments to see if there are any restrictions. However, overall with the current setup, we have found that having a unique algorithm for identification provides a good defense against replication.

We also attempted to request identification of a song in a different format, such as a recording, from the app. Since apps running on an emulator use audio input from the computer as a source, we modified the source to use internal audio while playing an mp3. We were not yet able to achieve a result with this approach, but as the real challenge of this task is to convert the audio input before giving it to the Shazam app, we know that the construction of such an intermediate reformatting tool would allow us to trick the Shazam app into identifying mp3's for us instead of normal audio, thus violating Shazam's security model.

5 Whisper

Whisper is an app which allows users to anonymously post their feelings or thoughts on any manner of subject, as well as anonymously reply to others's posts. Unlike the previous two apps, Whisper's security is less concerned with access to any back-end algorithms, but is instead centered on the anonymity guarantee. The success of Whisper's interaction model heavily depends on the willingness of users to post thoughts that they would never be willing to reveal if they were tied to their identities. Thus, we chose to see if we could use any information in the packets sent by Whisper to break this anonymity guarantee.

Let us begin by analyzing an example packet, corresponding to one of the most popular Whisper posts.

```
{"wid":"04f64da5e74443154243d3dc8f165ffedfbbb3",
"puid":"aee10f91-b485-4823-8806-96e7db783d8a",
"nickname":"Mr. Know","ts":1396713263160,
"url":"http://wac.a89c.edgecastcdn.net/80A89C/whisper.sh/whisper.sh/04f64da5e744431542
43d3dc8f165ffedfbbb3.jpg",
"text":"Every time I have to spell \"bananas\", I sing \"Hollaback Girl.\"",
"geo_title":"Louisiana",
"in_reply_to":"undefined",
"me2":2936,
"replies":176,
"popularity":368.2818915926223}
```

Figure 2: The packet received by the app for a popular Whisper post.

This packet contains all of the relevant data corresponding to the post content, but it also includes a great deal of identifying information about the one who made the post. A malicious user, then, might be tempted to listen to the data packets, rather than using Whisper's UI, to attempt to break the anonymity guarantee. To stop these types of abusers, we must analyze the actual data being included in these data packets, and ensure that none of this information exceeds the strength of information that can be garnered using the Whisper app's UI.

5.1 WID

The wid (presumably, Whisper ID) varies for every individual whisper post or reply. Thus, we can consider this to simply be a unique identifier for each post to allow easier access to specific posts if Whisper ever needs to find them. (Whisper is used quite frequently, so timestamps are not in general unique enough identifiers.)

Despite each wid being different, we at first hypothesized that there might be some correlation between the wid's of two posts made by the same person. We did not find any such correlation, although we did find another field that made such a theory effectively worthless - the puid.

5.2 PUID

The puid, unlike the wid, does not vary for every individual post. Instead, we discovered that it remains exactly the same for all posts made by a single user. (See Figures 3 and 4.) This immediately gives a passive listener the ability to use any single post made by a user and detect if any other post was made by them as well. Not only does this allow the passive attacker to correlate these posts (and then prefer or dislike posts based on the particular user that posted them rather than the individual post), but this also means that, if the attacker could verify that Alice had made any single post, they could also confirm whether or not Alice had posted anything else on Whisper, completely shattering the anonymity guarantee.

```
{"wid":"04f651bfc0c0357068694672f5694d9385f55e",
"puid":"a935b191-4cd2-44a8-bc73-eaaed3256aab",
"nickname":"b0xing_Steel","ts":1396730876713,
"url":"http://wac.a89c.edgecastcdn.net/80A89C/whisper.sh/whisper.sh/04f651bfc0c0357068694
672f5694d9385f55e.jpg",
"text":"I haven't slept in a week.",
"geo_title":"None",
"in_reply_to":"undefined",
"me2":0,
"replies":0,
"popularity":0.0}
```

Figure 3: The packet corresponding to a post we made on Whisper under the randomly generated nickname b0xing_Steel.

```
{"wid":"04f6f4d00ef6299371933f852decfa8fe64d47",
"puid":"a935b191-4cd2-44a8-bc73-eaaed3256aab",
"nickname":"b0xing_Steel","ts":1397431229948,
"url":"http://wac.a89c.edgecastcdn.net/80A89C/whisper.sh/whisper.sh/04f6f4d00ef6299371933f8
52decfa8fe64d47.jpg",
"text":"I pretend to hate my favorite music because it doesn't click with peoples perceptions
"geo_title":"None",
"in_reply_to":"undefined",
"me2":0,
"replies":0,
"popularity":0.0}
```

Figure 4: The packet corresponding to a Whisper post we made on a different day using the same nickname.

5.3 Nickname

When a user initially registers on Whisper in order to make posts, Whisper requires the user to choose a nickname, which can be however random the user desires. Whisper also provides a random nickname generator so as to prevent information about the user from being conveyed through their nickname. The user can freely change this nickname at any time, making it slightly weaker than the puid, but it also could be used to attempt to correlate the users who made various posts.

It is also worth noting that changing ones nickname does not actually change the puid. In order to confirm this, we actually changed our nickname, hoping that at least the puid was a pseudorandom function of the nickname, but it actually remained the same, meaning that a Whisper user aware of this lapse of security has no way to improve their anonymity, as the only action they could take (changing their nicknames) will not change their ID information.

```
{"wid":"04f71e3686549269531406a2b322dbf3232993",
"puid":"a935b191-4cd2-44a8-bc73-eaaed3256aab",
"nickname":"newNickname","ts":1397609042702,
"url":"http://wac.a89c.edgecastcdn.net/80A89C/whisper.sh/whisper.sh/04f71e3686549269531
406a2b322dbf3232993.jpg",
"text":"I can't get off of the computer no matter how much I know I should.",
"geo_title":"None",
"in_reply_to":"undefined",
"me2":0,
"replies":0,
"popularity":0.0}
```

Figure 5: The packet corresponding to a Whisper post we made using a new nickname. Notice that the puid label is the same as in Figures 3 and 4.

5.4 Geo_Title

The geo_title field is used by the Whisper app in conjunction with location data on each users phone to alert users to Whispers made by users in their rough geographic location. This is a reasonable service to desire, thus it is a slightly more valid field than, for instance, the nickname or puid that only serve to break anonymity. However, this geographic data, while reasonably vague, could still be used to help, for instance, distinguish between two users with the same nickname, or just give some small extra piece of information about a user. Thus, this is a less egregious break than the other two fields, but still should be considered to be a potential security flaw.

It is also worth noting that the geo_title field security hole is more easily fixed by a savvy user than the earlier puid flaw. The app will default to looking for GPS information about the phone's location, but a user could choose to turn off this feature on their phone, preventing Whisper from grabbing it, which results in the "geo_title":"None" entry seen in the posts we made. However, Whisper should still think about securing this information, especially since the app defaults to grabbing it, and since many users don't considering that such information could be used to break their anonymity.

5.5 Security Suggestions

Whisper's main security issues come from the inclusion of identifying information in their data packets. The most important change, then, would be to eliminate the nickname and puid fields from a typical GET request packet. This actually would not require any real overhead. It is understandable that Whisper might internally want something like the puid to ensure that no user was repeatedly violating their terms of service or otherwise making ill-advised posts. However, Whisper could do this by simply checking the puid of the user making the post (i.e. requiring the puid in the original POST request for the Whisper post.) Whisper could then internally store a map between wid's and puid's, such that it could internally correlate any posts as necessary. But other, ordinary users, would then only get the wid from their GET requests, with Whisper withholding both the nickname and puid as potentially sensitive data. Since the user-end app doesn't display either field anyways, this should not cause Whisper any loss in functionality or efficiency.

6 Ask.fm

In a format similar to Whisper, Ask.fm is an online service that allows users to directly ask questions to other users, with the option of sending such messages anonymously. The question is initially private to the recipient, but once a question is answered, both the question and answer become public on the profile of the recipient who answered the question. The key feature is supposedly knowing the identity of the recipient who answers, but not the user who poses the question. The goal of such a format is ostensibly so that people who know a user's account could post anonymous questions relevant to that user's life.

The simplicity of the design seems to mean that there are no obvious flaws in the way it transmits its data, as it does not seem to associate anonymous information with anyone but the desired recipient.

Here is a specific packet received when asking for a question on another users wall:

```
"type": "question",
"data": {
    "qid": 111608330588,
    "type": "anonymous",
    "body": "Can you ask a question for me?",
    "author": null,
    "authorName": null,
    "answer": {
        "author": "testing6857",
        "authorName": "6857",
        "type": "text",
        "body": "Sure. What?",
        "createdAt": 1399242036,
        "likeCount": 0,
        "sharing": [],
        "liked": false
    },
    "createdAt": 1399240686,
    "updatedAt": 1399242036
},
"ts": 1399242036
```

This request was made from user test6857 to another user testing6857. There are a number of potentially identifying fields in the packet data.

6.1 qid

The qid appears to be a unique identifier for every individual question. By itself, it does not appear to reveal any information about who submitted the question, and there does not appear to be any association with the qid and the questioners user profile.

6.2 author and authorName

The author and authorName fields contain the username and assigned name of the relevant party. The username is a unique identifier for every user, while the assigned name can be changed by the user. They occur in both the sender and receiver data, to associate a question or answer with a sender. When a question is marked anonymous, this information does not appear in the packet data. Instead, a placeholder value of null is inserted into the fields, anonymizing the questioner. The recipient does not appear to have this ability, due to the presence of the question on the recipients wall. When an answered question is deleted, the associated answer is disassociated and removed, leaving the question available to be answered again, if the recipient so wishes.

6.3 createdAt and updatedAt

These values contain timestamps for an action on a question or answer. Information like this is potentially exploitable, though highly unlikely. Knowing the time of when someone posts only potentially reveals a specific message, if one can correlate that informations with a question, and it does not reveal any other information about a poster.

6.4 Security Results and Suggestions

There are a number of things Ask.fm does more successfully than the other applications weve looked at. First, it authenticates any API requests under HTTPS, requiring an authentication token provided by the server at https://api.ask.fm/authorize to proceed. Second, because Ask.fm necessarily associates a question only with its intended recipient, no identifying information associates an asker with the question being asked. Only the recipient of the question is associated with the question, unlike in Whisper, where a specific ID can be associated with an anonymous persona.

Ask.fm's largest problem is probably that it does not obfuscate its API, despite not making it publicly available. Any GET request on `https://api.ask.fm/` could authenticate and make calls to the API to access the servers information. This sort of private API could be used to create a new ask.fm client, independent of the intents of the application creators. For example, if the creators of Ask.fm wanted to enforce a specific UI, like how Tinder limits a user to one profile at a time, an independent client could bypass that.

A suggestion for improvement is to secure the use of a private API by performing public/private key protection embedded in the application itself, requiring a thorough decompilation of the application to reverse engineer the API.

7 Conclusion

Android apps have a great deal of variety, and contain many different kinds of security guarantees. Two of the most critical to an app's infrastructure, though, are constraints corresponding to identities of users and use of the back-end API. With Whisper, we saw an example of the identity of users being betrayed by the app, specifically because of unnecessary identifying information being included in data packets. However, Ask.fm gave us an example of an app which correctly disassociates this sort of identifying information, showing that this problem is largely solvable. Much harder, however, is the task of securing a back-end service to ensure that it is only used as desired. In the case of Tinder, we saw that a dishonest client could bypass the app's UI, while Shazam gave us an example of a way to violate the principles of access to the back-end by giving unexpected inputs to the application. There exist a variety of strategies to enhance security. Shazam's proprietary algorithm made their client very difficult to replicate. Another possibility is the existence deeply encoded encryption in communication. Fundamentally, though, it is important that Android application developers recognize that there is a limited number of security guarantees, especially about the use of their APIs, that they will be able to enforce in practice, and they should accordingly plan their applications to have a realistic and attainable security model.

References

[1] *Genymotion*, `http://www.genymotion.com/meet-the-team/`, Accessed March 18, 2014

[2] Karl von Randow and Matthew Buchanan, *Charles Web Debugging Proxy*, `www.charlesproxy.com`, Accessed March 18, 2014

[3] Bryan Jacobs, *How Shazam Works to Identify (Nearly) Every Song You Throw At It*, `http://gizmodo.com/5647458/how-shazam-works-to-identify-nearly-every-song-you-throw-at-it`, Accessed March 29, 2014

[4] Avery Li-Chun Wany, *An Industrial-Strength Audio Search Algorithm*, `http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf`, Accessed March 31, 2014