Tinder Security Audit
SSN Project 2013

Joris Claassen Leendert v. Duijn Mick Pouw Esan Wit
{Joris.Claassen,Leendert.vanDuijn,Mick.Pouw,Esan.Wit}@os3.nl

December 22, 2013

Abstract

Tinder is a dating app that is being used by many users on a global scale. The Tinder application requires a Facebook profile as well as GPS location to function properly. Tinder claims not to share this information with other users. Since personal information like sexual preferences or your location is highly sensitive data it should be properly secured. This audit focuses on testing whether or not Tinder uses available technologies to withstand common attacks. Whilst most known leaks in Tinder appear to no longer be around there are still some points of concern. It turns out that there is an insecure connection which sends user preferences and GPS data to an analytics company.

Contents

1 Introduction

Dating apps are common place in society. Many dating apps include the option to contact matched users and send text messages or images. Due to the amount of personal information potentially involved in the matchmaking process this a prime target for identity theft. If an app or service deals with personal information, it has to ensure the security of this private information. Since the Tinder application has a lot of users, it would be interesting to check this security.

The user has to supply some information before using the app. This includes a Facebook profile API credentials and eventually even GPS information. The information that can be collected from the Facebook profile could be valuable for eavesdroppers or third parties. Information like GPS, the actual location of the end user.

In this paper we will look in to the risks of using Tinder, what information is sent where and how securely this information is transported over the internet. We also look into whether information is stored at a third party or even Tinder itself.

We will first discuss the materials and methods we used to research the Tinder application. We will then discuss the analysis we did on the Tinder application eg. the offline and online analysis. We will discuss certain attacks and possible weaknesses of the Tinder implementation. We will then draw a conclusion based on our results. We divided the research questions into several sub questions. These will be handled in the analysis. The sub questions we are going to answer are:

- Does Tinder implement HTTPS during communication?
- Does Tinder send privacy sensitive data to systems outside of Tinders control (e.g. client devices)?
- Is the undocumented, proprietary Tinder API prone to leaking data?
- Is it possible to eavesdrop on conversations between users?

2 Related Work

We found some interesting articles and papers online, which inspired us to do this research.

This research is loosely based on similar research into the Grindr application[1]. The researchers examined the security of the Grindr dating app and found a serious error in the implementation of Grindr. The Tinder app is not based on the Grindr application.

The first article on Tinder we found was an article on Quartz[2], describing a possible leak of data Tinder used to have. Tinder claimed these data leaks ".. happen[s] as you're developing products", and was going to be patched the same day. The issue was fixed only a week later. This leads us to believe the data privacy of Tinder users is not one of their top priorities.

Recently security researcher Shaked Klein Orbach did an audit of the security in the Tinder iOS app[3]. His findings included the coupling of Tinder Photos and Facebook ID's, due to excess data sent by Tinder. The Tinder iOS app has a matching mechanism, which allows two users to "match" each other. Orbach used the earlier found Facebook ID's to match people to other people who where not friends, thus bypassing the matching mechanism.

Our research will look into the practical security of the Tinder Android app. This will include, for example, the implementation of SSL, the possibilities of man in the middle attacks, eavesdropping on chat messages, identity spoofing or setting up fake matches. When gathering the results of these tests, we can get insight in the security of the Tinder implementation.

3 Materials and Methods

Our research focuses on several particular attack vectors against the platform. We will analyze several aspects of the application. User authentication, authentication of API server(s), possible user-base enumeration, and leaking of information via side channels. We will investigate this by using a combination of static offline analysis of the program code and analysis of live network traffic.

3.1 Static analysis

The static analysis of the Tinder application was by decompiling the source files from a retrieved APK file using APK to Java[1]. This APK file has been retrieved from a rooted android device. The decompiler used was the freely available jd-gui[4].

The purpose of the static analysis is to extract some information on how Tinder is implemented. We focus on which libraries are used, how communication is handled and if there are any obvious flaws in the implementation. We will also attempt to extract the structure of their API for later use.

3.2 Network traffic

The network traffic was analyzed by using a Man in the Middle(MitM) attack. To do this we setup an access-point using a laptop and airbase-ng[2]. The access point would then forward all requests via iptables to our proxy.

3.2.1 HTTP(S) Proxies

To complete our MitM setup we needed to proxy HTTP(S) requests and fake certificates for the various requested web resources. Because we wanted to analyze all requests we chose to set our proxy in transparent setup as opposed to defining the proxy in the WiFi settings. Reason for this is that applications on Android don't use the proxy as configured in the WiFi settings. As such we required a proxy which could generate SSL certificates for the various end-points requested by the application.

First we used the Burp suite[5]. This program sat as a proxy between client and server but allowed the user to change the data in the middle. This would allow us to change request and response data without having to create a placeholder server. However it turned out that Burp did not have the option to run completely transparent and serve proper SSL certificates.

After this we had a brief look into using MitM proxy[3], as this was previously used for exactly this purpose in another analysis[1]. However we didn't manage to configure it to run transparently with our setup.

Finally we settled on an implementation using SSLsplit[4]. This allowed us to use our own certificate 'authority' to generate proper SSL certificates for any endpoint connected to during our tests. By importing this authority we

[1] `http://forum.xda-developers.com/showthread.php?t=1910873`
[2] `http://www.aircrack-ng.org/doku.php?id=airbase-ng`
[3] `http://mitmproxy.org`
[4] `http://www.roe.ch/SSLsplit`

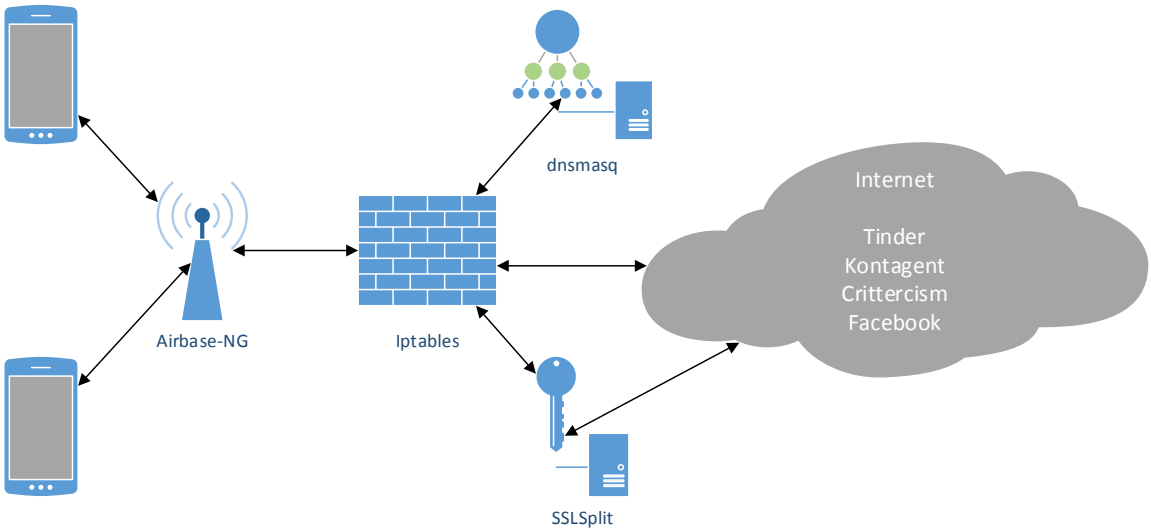

Figure 1: Network layout

where able to run the Tinder application while logging all traffic to plaintext files for later inspection. However for a casual attacker this would be highly impractical as we had to manually install the certificate authority and accept severe warnings to get it to work.

It should be noted that the app refused to function when serving HTTPS resources with invalid certificates. This included certificates which were signed but for a invalid domain, unsigned for another domain, or unsigned for the correct domain were all rejected by the application.

3.2.2 Realization

3.3 User authentication

Tinder requires its users to register and log in using Facebook. Facebook log in uses a token which, once generated, can be used by any interested party. As such it might be possible to log a user in with a third party Facebook token. To check this we will attempt a log in request to the API of Tinder and replace their Facebook token with one of our own. If Tinder does not verify whether or not the supplied token is created using their own Facebook app this will allow any Facebook app owner to attempt to log in to Tinder for a user.

If it is possible to log in to the Tinder API using a random access token from another application this may enable third parties to profile users of Tinder or manipulate existing contacts, current recommendations, and chat conversations

4 Analysis

4.1 Offline analysis

Decompilation of the Tinder APK file provided some clear insights for further research. It provided us with a list of libraries used by Tinder, API endpoints, and some clarification on the authentication scheme used.

4.1.1 Libraries

Tinder uses several libraries which appear to have code enabling network connectivity. These libraries may provide a channel from which information may be obtained. The libraries found are:

Apphance[5] Usage analysis and user feedback syndication.

Crittercism[6] Analytics and trend development.

Facebook[7] User authentication.

Kontagent[8] Analytics and trend development.

Volley[9] Android networking library.

Apphance, Crittercism and Kontagent are all products related to app usage analysis. Apphance seems to be dedicated to analysis of app crashes. Since the app did not crash during testing we have not intercepted any request of Apphance. Crittercism and Kontagent are used for usage analytics. Crittercism seems to be focused on application usage and device statistics whilst Kontagent seems focused on user details and interactions.

Facebook is used for user registration and log in. The app retrieves an authorized token from the library which is used to log in with the Tinder backend.

The Volley library is a library provided by Google to make it easier for app developers to use HTTP requests effectively. The native Tinder API usage is build upon this library. As such we don't expect many developer bugs or faults which would enable us to retrieve data.

4.1.2 API resources

Due to incomplete obfuscation of the APK[10] file certain resources could easily be identified. The base URL of the Tinder API was also revealed to be 'https://api.gotinder.com'.

```
URL_MEDIA_BASE = "/media";
URL_MEDIA = "https://content.gotinder.com" + URL_MEDIA_BASE;
URL_USER_BASE = URL_BASE + "/user/";
```

[5] http://www.utest.com/apphance

[6] https://www.crittercism.com/

[7] https://developers.facebook.com/docs/android/

[8] http://www.kontagent.com/

[9] https://android.googlesource.com/platform/frameworks/volley

[10] It should be noted that after the Tinder 2.0 update for Android this file was properly obfuscated.

```
URL_AUTH = URL_BASE + "/auth";
URL_MATCHES = URL_USER_BASE + "matches/";
URL_PING = URL_USER_BASE + "ping";
URL_PROFILE = URL_USER_BASE + "profile";
URL_RECS = URL_USER_BASE + "recs";
URL_UPDATES = URL_BASE + "/updates";
URL_REGISTER_PUSH = URL_BASE + "/device/android";
URL_LIKE_ADJUNCT = "/like";
URL_PASS_ADJUNCT = "/pass";
URL_DESTINATIONS_LIST = URL_BASE + "/destinations";
URL_DESTINATION = URL_BASE + "/destinations/";
URL_REPORT = URL_BASE + "/report/";
```

We also found that the API is adding an X-Auth-Token header to each request with a token retrieved from the `/auth` endpoint.

4.1.3 Local storage

Analyzing files left behind by the app on the system opened up some new locations where data could be leaked. Most noteworthy is the 'tinder.db' file which is an unencrypted SQLite3 database. This database houses information on past and future gradings as well as a complete chat history. Since this files is protected by Android to only be accessible by the app itself this is generally okay. However on rooted devices or with the possibility of a new exploit encryption might be preferable over the performance loss.

4.2 Dynamic analysis

After having concluded our offline analysis we started our online analysis, using our test setup as described in section 3.2.1. During this process we had two test users run the app on several occasions, both instructions to generate a representative stream of requests for a complete analysis of application behavior and possible weaknesses. We inspected the logs collected to investigate where certain data was sent, whether it was secured in transit, and whether there were other feature worth investigating.

4.2.1 Secured traffic

We inspected the hosts the device running the application connected to, after eliminating traffic from non Tinder related hosts we were left with the following hosts using HTTPS:

Number of requests	Hostname
66	api.crittercism.com
3635	api.gotinder.com

Seeing encrypted traffic to Crittercism, a company which is summarized on their website [6]:

> We collect everything We group it so it's easy to see and prioritize for impact analysis We enable the identification of key trends that are happening with your app in real-time

This prompted us to make an inventory of what data they received, during our test this included the following:

- Hashed device id, equivalent to the UUID on the device making the request.
- App id, an identifier which is the same for our two inspected accounts.
- Device name, Platform name
- Library version, the Crittercism code version.
- Meta-data with Username, UID(Tinder user ID) and Name

This information will most likely be stored indefinitely and used for legitimate purposed as advertised and permitted by the terms, conditions and privacy statements of Tinder.

The API endpoint `api.gotinder.com` is only available of a secure HTTPS connection and the application refused to connect using anything but a properly signed certificate from a trusted CA (see section 3.2.1).

With this we can conclude an answer to `Does Tinder correctly implement HTTPS during communication?`, namely that the Tinder API uses HTTPS to secure traffic to their service. Although we were able to circumvent this (see section 3.2.1) we had to install a custom root certificate authority on our client devices, which required full administrative control of the devices. Should this be a concern for Tinder they could consider certificate pinning [7].

4.2.2 Insecure traffic

We also we saw a large number of requests going to non-HTTPS endpoints, closer inspection revealed the following endpoints to receive non encrypted requests:

Number of requests	Hostname
469	images.gotinder.com
860	api.geo.kontagent.net

Of these the most interesting seems to be `api.geo.kontagent.net`, information we noticed being sent to this host:

- Age of the local user
- Gender of the local user
- User ID(Tinder ID) of the local user
- Location, in 7 decimals precision
- User number
- Selected sexual interest
- User ID(Tinder ID) of the 'matched' user
- Target user ID(Tinder ID) of another action

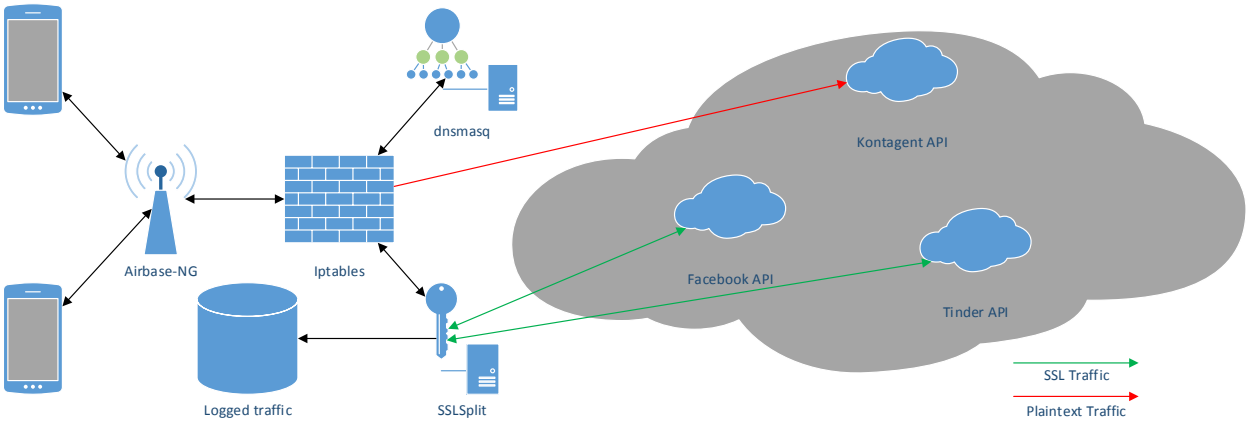

Figure 2: Data flow

A user of Tinder will should be aware that most of this information is transmitted, the privacy statement made by Tinder specifies that this data can/will be shared with service providers, Kontagent seems to be one of these parties specializing in data mining, analytics services and visualization.

What users might not realize is the insecure channel used for this, enabling a motivated third party to passively inspect and log it.

This answers `Does Tinder send privacy sensitive data to systems outside of Tinder's control?` as they do send sensitive data (as illustrated in figure 2) over an unsecured channel, one which they cannot control access to. However most data being sent to Tinder's partners is communicated over encrypted channels, and all to reputable companies.

We could not gather much information about other users which the application did not already show.

4.2.3 Targeting authentication

After getting a feeling for the structure and general requirements for an API call we started to look into a more active approach. One of the first attacks tried on the Tinder API was to check if we could exchange a third party Facebook token for a Tinder session token.

We send a fully authorized token, created from the Facebook Graph API Explorer[11]. However it appears that Tinder is validating the origin of the supplied token. As such we conclude a token created by the Tinder app is required for proper authentication with the Tinder backend.

4.2.4 Replay attacks

Inspecting our logged traffic we located some legitimate chat traffic in order to test whether we could automate the chat functionality. For this experiment we had three parties

- The user

[11] `https://developers.facebook.com/tools/explorer`

- The chat partner
- The third party

With the user having sent a test message to the chat partner[12] our third party took the network logs and crafted another request, this crafted request was then sent to the Tinder API server which accepted, processed and relayed it to the original chat partner. The conversation can be found in appendix B.1. During this experiment we noticed that while our chat partner received the message, the user on whose behalf we sent it did not get any notification in their chat log. We suspect the latter to be due to the response to the actual message being sent containing the information the application would normally add to the chat log.

Execution of this attack requires the following resources:

- The template API request (see appendix A.1)
- The identifier of a match with the target user
- A (currently) valid authentication token for the sending user
- The application version identifier i.e. the user agent for the request
- A properly encoded JSON payload specifying a message

This attack shows that it is possible to write a simple chat bot, even if you would have to resort to someone manually creating matches with other users. There are strong indicators that spam bots are active on Tinder [8], we suspect that automation like we demonstrated is used for these purposes.

From the initial replay we wondered how the system would respond to replacing the Match-ID with a User-ID, To test this we used two accounts who had never had contact and where in fact both Male, set to look for female matches. By tweaking the request to simply replace the match identifier with our test user's identifier we hoped to forcibly open a chat dialog on the targeted users phone, however the execution revealed that the Tinder API makes a distinction between a match and a user, simply refusing to send our message, as can be seen in appendix B.2.

4.2.5 Data retrieval

We discovered the application retrieve history from the server by making a call to `api.gotinder.com`, by adjusting the requested time period we obtained an entire history for our test accounts, on one of the accounts we got results covering the entire history of the account, more than a month of collected chats.

This call requires the following information to be executed:

- The template API request (see appendix A.2)
- A (currently) valid authentication token for the sending user
- A properly encoded JSON payload specifying a date

[12] who was told of our experiment

It returned the following information:

- Chat history with timestamps
- User profiles related to the chat history
- The last time a chat partner was online, precise up to milliseconds
- A list of identifiers what we strongly suspect are Tinder matches rejected by the user

This answers `Is the undocumented, proprietary Tinder API prone to leaking data?` As we have seen that the API will divulge a users own history, though it is limited in that you require a valid authentication token for this user, at which point you are likely to have enough other methods to come by the same data. We have not found any effective ways to harvest data from unknown users.

Previously Tinder send out more than the required information to the application as shown by Chintan Parikh[2], however currently no data is sent to the application other than that which is relevant to actions being performed by the user.

With this we can also conclude `Is it possible to eavesdrop on conversations between users?` Since encryption is properly implemented in communicating with the API endpoint there is no straightforward methods of eavesdropping on a users chats, the only methods we found require root access to the device (see section 4.1.3), a stolen user token (see section 3.3) for history retrieval, or fully transparent SSL decryption (see section 3.2.1).

4.3 Legal analysis

Since Tinder is dealing with privacy sensitive information, we also need to check whether the information that is transferred, is transferred according to laws and regulations. You can look at these laws from different countries, but we decide to look into it for the Netherlands. This also implies to the European law.

4.3.1 Dutch law

The Netherlands has a common law in order to help the protection of personal data. This law is called 'Wet bescherming persoonsgegevens'[9]. This law states that the user, which data is collected, must be asked for explicit permission. The application however asks for this permission, thus the gathering of the personal data is legit.

4.3.2 European law

The European law is, however, more strict of securing the information in transit. This is described in the 95/46/EC[10] regulation. It says that the developer of the application must take care of security measurements to ensure the safe transit of data. This data must be protected against:

- Accidental or unlawful destruction

- Loss of data
- Alteration of data
- Unauthorized disclosure or access
- Other forms of unlawful processing

The interesting part of these criteria is that the developer must also ensure that the data is not accessible. However the API that Tinder uses (Kontagent) does not make use of any encryption but merely uses a base64 encoding. This is not considered to be good encryption.

This tells us that Tinder is not implementing the application according to the European Regulations. There are no clear penalties against these kind of implementation flaws[11], but it is highly recommended to fix these issues. The usage of a post implementation in combination with a SSL secured connection would already suffice to ensure the safe passing of this information and thus preventing unauthorized access from sniffing the HTTP network.

4.3.3 Privacy concerns

Tinder sends highly private information over the network. The information that is transferred to Kontagent contains the GPS location, gender, age, sexual preference, user number and Tinder ID. This can be intercepted by an eavesdropper, since the traffic is not encrypted. The information retrieved by the eavesdropper can be used for different purposes.

Tinder says in it's privacy statement that it does not store this privacy sensitive information, but it sends this information to the Kontagent API. This is a third party. It is possible that Tinder lets this third party store the data as this is allowed according to the Tinder privacy statement. The user agrees with this when installing and using Tinder for the first time. Tinder says in it's privacy statement that it is allowed to collect the following data next to the data listed above:

- Technical details, device, browser, OS
- Preferences, timezone language,privacy settings, product preferences
- URL of last visited age, referrer.
- Clicked buttons, controls and ads
- Usage tracking, Online/Offline status

Data gathered by third party API's can be gathered during the use of Tinder. When the Tinder account is deleted, this information will remain stored on with the third parties. The stored information cannot be deleted. Third parties can sell or use this information for targeted add. All of this is according to Tinders privacy statement the user agreed upon.

5 Conclusion

We set out in our research to test whether the Tinder application is resistant to casual attack after recent publications have proven that it was not so in the past. As previously discussed many of the old holes have been sufficiently patched and are no longer exploitable.

Currently the Tinder API is not leaking any data and is properly using SSL. It is not possible to eavesdrop on a user's conversations without first obtaining an authentication token for either side of the conversation or installing a fake certificate on the user's device. However we still found some minor issues, including an analytics company being sent sensitive data over an insecure connection. We do believe Tinder to now be secure for everyday use as they use proper authentication and apply encryption for data sent to their API.

Our primary advice for Tinder would be to investigate whether the communication with `Kontagent` can be secured. Furthermore we would also like to point out some other points where Tinder could improve their platform and its security: They could pin certain certificates, implement encryption on their image resources and perhaps even try to harden or rate limit their API to deter the bots exploiting their application.

Lastly it would be prudent to stay up to date on recent developments. During the course of this research another developer has shown that there was a feature in the Tinder API which could be abused. Although this had no direct or lasting consequences it illustrates that application security is an ever changing field where companies must remain on the lookout for new attacks.

References

[1] W. Katz T. Fiebig. Grindr application security evaluation report. https://www.os3.nl/_media/reports/grindr.pdf, 2013.

[2] Zachary M. Seward. Dating app tinder briefly exposed the physical location of its users. http://qz.com/106731.

[3] Shaked Klein Orbach. Tinder privacy issues. http://www.shakedos.com/2013/Nov/23/tinder-privacy-issues.html.

[4] JD-GUI. A graphical decompiler of java bytecode to readable code. http://jd.benow.ca.

[5] Burp suite. An intercepting proxy, which lets you inspect and modify traffic between your browser and the target application. http://portswigger.net/burp/.

[6] Crittercism.com. Crittercism developers summary. https://www.crittercism.com/for-developers/.

[7] Moxie Marlinspike. Your app shouldn't suffer ssl's problems. http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/.

[8] Symantec Employee Satnam Narang. Tinder: Spammers flirt with popular mobile dating app. http://www.symantec.com/connect/blogs/tinder-spammers-flirt-popular-mobile-dating-app.

[9] Dutch government. Wet bescherming persoonsgegevens(dutch). http://wetten.overheid.nl/BWBR0011468/geldigheidsdatum_21-12-2013#Hoofdstuk2.

[10] European Union. 95/46/ec on the protection of individuals with regard to the processing of personal data and on the free movement of such data. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML.

[11] Dave Shackleford. Regulations and standards: Where encryption applies. Sans, 2007. https://www.sans.org/reading-room/analysts-program/encryption-Nov07.

A Scripts

A.1 Sending a Tinder message

We used the following template to send chat messages to existing matches, using information gathered by our test user.

```
#!/bin/sh

### Replace : With
### 888888888888888888888888 : Match ID
### 11111111-2222-3333-4444-555555555555 : Currently valid token
### Tinder testbericht 4 : A message aproved by the tokens owner

curl -v "https://api.gotinder.com/user/matches/888888888888888888888888" \
-H "X-Auth-Token: 11111111-2222-3333-4444-555555555555" \
-A "Tinder Android Version 2.0.1" \
-H "Content-Type: application/json; charset=utf-8" \
-H "Accept-Encoding: gzip" \
-d '{"message":"Tinder testbericht"}'
```

A.2 Retrieving user history

We used the following template to request a history of the authorized user, using the date parameter to specify the period we were interested in.

```
#!/bin/sh

### Replace : With
### 11111111-2222-3333-4444-555555555555 : Currently valid token
### 2000-12-09T14:55:30.356Z : Earliest time the history includes

curl -v "https://api.gotinder.com/updates" \
-H "X-Auth-Token: 11111111-2222-3333-4444-555555555555" \
-A "Tinder Android Version 2.0.1" \
-H "Content-Type: application/json; charset=utf-8" \
-d '{"last_activity_date":"2000-12-09T14:55:30.356Z"}'
```

B Traces

B.1 Replay, chat message to friend

```
* Connected to api.gotinder.com (54.236.172.128) port 443 (#0)
* SSL connection using DHE-RSA-AES256-SHA
* Server certificate:
*   subject: OU=Domain Control Validated; CN=api.gotinder.com
*   start date: 2013-07-17 01:34:03 GMT
*   expire date: 2018-07-17 01:34:03 GMT
*   subjectAltName: api.gotinder.com matched
*   issuer: C=US; ST=Arizona; L=Scottsdale; O=GoDaddy.com, Inc.; OU=http://certifica
*   SSL certificate verify ok.
```

```
> POST /user/matches/333333333333333333333333 HTTP/1.1
> User-Agent: Tinder Android Version 2.0.1
> Host: api.gotinder.com
> Accept: */*
> X-Auth-Token: 99999999-8888-7777-6666-555555555555
> Content-Type: application/json; charset=utf-8
> Accept-Encoding: gzip
> Content-Length: 34
* upload completely sent off: 34 out of 34 bytes

< HTTP/1.1 200 OK
< Content-Type: application/json; charset=utf-8
< Date: Mon, 09 Dec 2013 10:00:00 GMT
< Server: nginx/1.1.19
< X-Powered-By: Express
< Content-Length: 142
< Connection: keep-alive

{
  "created_date":"2013-12-09T10:00:00.000Z",
  "from":"aaaaaaaaaaaaaaaaaaaaaaaa",
  "to":"bbbbbbbbbbbbbbbbbbbbbbbb",
  "message":"Tinder testbericht 3"
}
```

B.2 Replay, chat message to stranger

```
* Connected to api.gotinder.com (107.21.49.214) port 443 (#0)
* SSL connection using DHE-RSA-AES256-SHA
* Server certificate:
*   subject: OU=Domain Control Validated; CN=api.gotinder.com
*   start date: 2013-07-17 01:34:03 GMT
*   expire date: 2018-07-17 01:34:03 GMT
*   subjectAltName: api.gotinder.com matched
*   issuer: C=US; ST=Arizona; L=Scottsdale; O=GoDaddy.com, Inc.; OU=http://certifica
*   SSL certificate verify ok.

> POST /user/matches/111111111111111111111111 HTTP/1.1
> User-Agent: Tinder Android Version 2.0.1
> Host: api.gotinder.com
> Accept: */*
> X-Auth-Token: 11111111-2222-3333-4444-555555555555
> Content-Length: 55
> Content-Type: application/x-www-form-urlencoded
* upload completely sent off: 55 out of 55 bytes

< HTTP/1.1 401 Unauthorized
< Content-Type: application/json; charset=utf-8
< Date: Mon, 09 Dec 2013 14:00:00 GMT
< Server: nginx/1.1.19
```

```
< X-Powered-By: Express
< Content-Length: 93
< Connection: keep-alive

{
  "status":"401",
  "error":"X-Auth-Token, match id, and message are required to send a message"
}
```

C Contributions

	Esan	Joris	Leendert	Mick
Global				
Decompilation	✓			
Static analysis	✓	✓	✓	✓
MitM Setup		✓	✓	✓
Traffic Generation		✓		✓
Traffic Analysis	✓	✓	✓	✓
Replay Analysis	✓			
Presentation		✓		✓
Report				
Introduction	Initial	Expanded		Expanded
Related Works		Initial		Expanded
Approach	Expanded	Expanded	Expanded	Initial
Static Analysis	Initial		Expanded	Expanded
Dynamic Analysis	Expanded		Initial	
Legal Analysis	Adjusted			Initial
Conclusion	Expanded		Initial	