

Security Analysis of Snapchat

Madeleine Dawson, Heeyoon Kim, Wei-En Lee, Kelly Shen
6.857, Spring 2016

Abstract

Snapchat is a popular app, especially among millennials. The app is unique compared to other apps in that photos, videos, and messages are ephemeral, an interesting property that we attempted to break by looking at the data dump on a rooted Android device. We also analyzed Snapchat's web and iOS apps. Many of our exploits on the iOS app required using a jailbroken iPhone. By attempting network attacks, we determined that the persistent adversary would be able to remotely intercept a packet containing a user's username and password as the user logs in. We also found SQL tables containing information that should be encrypted. We conclude with a list of recommendations to address the issues discovered during our tests.

1. Permission and Responsible Disclosure

Before beginning our analysis of Snapchat's mobile and web apps, we received explicit permission from Snapchat under a responsible disclosure policy. On completion of our evaluation, we will send this paper to the Snapchat team and inform them of any other relevant findings. Additionally, we will provide the team with recommendations to improve the security of their apps. This paper should be posted on the 6.857 page only after we email and receive approval from Snapchat. We will notify the staff once this is done.

2. Overview

We begin this paper by introducing Snapchat and our project goals (section 2), describing previous related work (section 3), defining Snapchat's security policy (section 4), and formulating an attack model (section 5). We then describe the methodology of all attacks and analyses in section 6 and detail the corresponding results in section 7.

2.1. Snapchat

Snapchat has quickly become a popular app, with 100 million daily active users and 6 billion videos being sent per day¹. It is a social media app that allows users to send photos, write messages, and transfer cash to other users. The Snapchat app is available for download on Android and iOS devices. Users can also login through a web portal² to view and change account settings.

One feature that sets Snapchat apart from other messaging platforms is that all content is transient--after receipt, messages and photos can only be viewed for a limited time. These restrictions not only make Snapchat unique but also protect the visibility of users' content.

2.2. Project Goals

The main goal of this project was to analyze how effectively Snapchat enforces its security policies (defined in section 4) for their Android, iOS, and web apps. We also wanted to explore ways users can circumvent content-viewing restrictions as well as ways an adversary could find a user's login credentials or personal information. Furthermore, another goal was to analyze the security of Snapcash transactions.

3. Related Work

3.1. Phantom

Phantom for Snapchat³ is a popular app modification that runs on jailbroken iOS devices. Phantom adds additional features to the original app, such as the ability to keep snaps open for longer than allowed, to save received media to a designated destination, and to spoof your location. The latter method tricks the Snapchat app into thinking your location is different from your physical location and allows users to use Geofilters from any location in the world.

3.2. Major Security Threats in 2014

Concerns over Snapchat's security were widely discussed at the beginning of 2014 after Gibson Security released a full disclosure of Snapchat's private API⁴. 4.6 million users' username and phone numbers were leaked because of this event⁵. The main vulnerability was that Snapchat had hard-coded the encryption key used by *all* Snapchat accounts. Using the disclosed hard-coded key, third-party applications offered features not supported by Snapchat, such as decrypting Snapchat images on Android phones⁶. Snapchat has since greatly improved their security scheme, although certain hard-coded information can still be found in the Android app source code, which is discussed in section 7.5.2.

3.3. Previous Student Work

Last year, a group of MIT students focused their 6.857 project on the security analysis of iOS apps.⁷ In their project writeup, they documented a few attacks they performed on the Snapchat iOS app. We performed similar attacks on a jailbroken iOS 8 device and found that the method is rather trivial. With Cycrypt code injection, the command `choose(LoginViewController)[0].passwordTextView.text` simply shows what the user has typed into the password field. The adversary has to be on the app's login screen, and even then the command only shows the adversary what they have just typed in. This information is not very helpful if the adversary does not already know the password. In addition, the input is not stored, so if the adversary logs out and logs back in, the field would be blank. We completed additional investigations with Cycrypt, as detailed in section 6.4.2.

4. Security Policy

To identify Snapchat's security vulnerabilities, we must first explicitly define a security policy. There are four data types: stories, snaps, chats and cash. Each data type involves at least two principals. A commonality among all four datatypes is time-constrained availability.

4.1. Snaps

Snaps are pictures or video messages taken and shared with at least one chosen Snapchat friend.

- Sender:
 - Can only send to friends
 - Cannot view the snap after sending
 - Can choose how long (up to 10 seconds) receivers would be able to view the snap
 - Can see who viewed, screenshotted, or replayed
 - The image (snap) must be taken with camera at the time of the snap

- Receiver:
 - By default, the receiver has to be a friend of the creator to view; however, the receiver could change his/her settings to allow anyone to send snaps directly
 - Can view once, and replay at most once, which must be within 24 hours of receiving
 - Can take screenshots of the snap

4.2. Stories

Stories are compilations of snaps that can be seen by a chosen audience, which is all of the creator's friends by default, or everyone on Snapchat or a customized group. It is a powerful tool to reach out to Snapchat users; for example, all remaining presidential candidates have official Snapchat accounts and upload stories regularly.

- Creator:
 - Can view as many times as desired within 24 hours of posting
 - Can see who viewed or screenshotted
 - Can delete or save story at any time

- All friends of the creator:
 - Can view as many times as desired within 24 hours of posting
 - Can take screenshots of the story

4.3. Chats

Chats are text exchanges between two friends. The message is cleared after the receiver views it. In addition, both the sender and receiver can take screenshots and see if the other has taken a screenshot. Similar to snaps, the receiver cannot view chats sent by non-friends by default but could change their settings to allow anyone to send chats directly.

4.4. Snapcash

Snapcash is created in partnership with Square. It allows two friends to exchange money by simply typing a dollar sign and an amount in a chat. Cash is sent immediately, so the only way to issue a refund is for the receiver to send the amount back to the sender. All payments are processed by Square, so the security of Snapcash is highly dependent on the security of Square Cash. To complete the transaction, both the sender and receiver have to have their U.S. issued Visa or MasterCard debit cards linked to their accounts. In addition, the sender can only send up to \$250 in a week unless they provide more personal information to Snapchat.

5. Attack Model

In general, an adversary of a system tries to disrupt the activity of honest users. In the case of Snapchat, an attacker might try to undermine the security of others by stealing user credentials or information. Additionally, an attacker might try to get around Snapchat's various content-viewing restrictions, thus violating other users' expectations of Snapchat's content security promises.

5.1. Expose User Information

Many users often reuse login credentials between apps. Therefore, if an attacker Evan knows Alice's email address, discovering her Snapchat password may make it much easier for him to gain unauthorized access to Alice's other accounts. Additionally, if Evan is Snapchat friends with Alice, our adversary could gain access to Alice's Snapchat account and send all of her money to himself. Knowing Alice's Snapchat credentials could allow Evan to pull off many nefarious anti-Alice attacks in a short period of time.

Snapcash also requires that users enter their debit card information for payment purposes. If Evan were to discover Alice's payment information, he could easily pretend to be Alice and spend her money online.

Snapchat allows users to protect identifying information such as full name and birthday. If Evan had unauthorized access to other users' personal information, this breach of privacy could have real-life repercussions for the victims.

5.2. Bypass Content-Viewing Restrictions

As described in section 4, Snapchat places heavy restrictions on when and for how long users can view content. These restrictions not only define Snapchat's brand but also give users a sense of security about the availability of their content. If an adversary could bypass Snapchat's restrictions, they could gain access to content not intended for them as well as undermine honest users' security expectations.

6. Methods

6.1. Social Attacks

Social attacks on systems or apps generally involve an adversary deceiving users using psychological manipulation or other trickery. For example, in the case of Venmo, an adversary Evan could send money to Alice but then ask her to repay him on the premise that the first payment was an accident. Meanwhile, he could simultaneously submit a report to Venmo's support team asking for the money back. Evan could then get his payment back twice from Alice. While researching Snapcash, we investigated whether this type of attack would be possible using Snapcash's support services.

Another simple trick involves Snapchat's Geofilters. Usually, Snapchat chooses which Geofilters to display based on the user's current location. But if a user Alice uses a location spoofer (such as the one provided by Phantom, mentioned in section 3.1), she could set her apparent location to be anywhere in the world to access different Geofilters. Alice could use this trick to deceive her friends of her current location.

Since Snapchat's philosophy is based on ephemeral media, the app notifies users if someone saves their content via screenshot. If Alice sends a chat, snap, or story, she will receive a notification if anyone takes a screenshot while viewing her content. One simple way to bypass this notification is to take a picture of the phone screen with another camera. However, we explored ways to bypass the notification when actually taking a screenshot. We experimented with changing network connectivity settings and screenshot timing when carrying out this attack.

6.2. Database Dump

Like many other applications on the platform, the Android version of Snapchat stores structured data within a SQL database. While all of the information is also probably stored on Snapchat servers, the local databases presumably reduce loading times. Normally, the directory in which the database is stored is inaccessible to the user. However, using a phone with root access, users can easily explore all of the files found in any system folders.

This attack requires the adversary to have access to the victim's Android device and also requires the device to have root access. Depending on the device, it may be possible to obtain root access without

losing data. To execute this attack, we used a file explorer on a rooted tablet running Android 5.1.1 to find a database at `/data/data/com.snapchat.android/databases/tcspahn.db`. We exported this to a laptop, where we used DB Browser for SQLite⁸ to view the contents. We found that the database included some sensitive information that is normally not visible even to authenticated users. The results of the attack are documented in section 7.2.

6.3. Network Attacks

The following network attacks were done on the Snapchat webapp. Similar procedures were carried out on the iOS app--methods and results are described in sections 6.4.2 and 7.4.2, respectively.

6.3.1. Man in the Middle

Charles Proxy is a proxy server that can intercept and record packets. Once installed, the proxy automatically reconfigures the network settings so that it will trick the client to thinking that the proxy is the server, and similarly trick the Snapchat server to thinking it's the client.

Our team installed Charles Proxy on a Mac, and inserted breakpoints on `snapchat.com` and `accounts.snapchat.com`. This enabled us to automatically stop any packets that are sent to and from those URLs.

We also used Charles Proxy for attempting a Man in the Middle attack on the iOS app, which is described further in section 6.4.2. Details of successes with MitM on the webapp are documented in section 7.3.1.

6.3.2. Decrypting Packets

Because Snapchat uses HTTPS and TLS, the packets we were able to intercept (see section 6.3.1) were encrypted. We then attempted to intercept and decrypt packets using Charles Proxy, Fiddler, and Wireshark. Our goal was to be able to decrypt packets sent to and from the device running Charles Proxy as well as from other devices. Below, we outline the procedure we used for the former case. The latter case, decrypting packets sent to and from a device not running Charles Proxy, is described in section 6.4.2.

We first installed Charles Proxy on a Mac. We inserted breakpoints on `snapchat.com` and `accounts.snapchat.com`. Thereafter, we were able to intercept all packets sent to and from those pages. We were able to detect the type of device and browser the packets were sent from by analyzing the packet header. We also analyzed the packet request bodies, and while almost all of it was encrypted, we noticed that small parts of the request body were still readable. For example, it was apparent that Snapchat uses digicert SHA2 for certification.

In order to decrypt packet bodies, we installed the Charles Root Certificate, a "malicious" certificate that would be used for authentication, rather than the computer's public key. We then enabled Charles Proxy's SSL proxying feature. This feature uses the malicious certificate to decrypt packet content. We were able

to decrypt packets sent from the laptop running Charles Proxy (see section 7.3.2). We were also able to decrypt packets sent from a remote iOS device under very specific conditions (see section 6.4.2).

We were also able to replicate these results using Fiddler NET 4 installed on a Windows computer. Similar to Charles Proxy, we installed a malicious certificate to be able to decrypt packets.

Using Charles Proxy and Fiddler, we were unable to intercept all packets that were sent by nearby devices on the network. When we tried the same process with WireShark, we were able to intercept packets but were unsuccessful in decrypting them.

6.3.3. Replay Packets

Fiddler is an application that can intercept HTTPS packets for websites. We downloaded the NET 4 version of Fiddler on Windows. Once downloaded, the application automatically reconfigures network settings so that all packets are redirected through the Fiddler proxy. Our team used Fiddler for `accounts.snapchat.com` and `snapchat.com`. In particular, we saved packets using Fiddler and attempted to replay them using Fiddler's Replay feature.

We also attempted to use Charles Proxy to conduct a replay attack. Charles Proxy has a feature called Repeat, which we can set to replay a request for a certain number of times every certain number of seconds. Because the Charles Proxy free version only allows usage of the app 30 minutes at a time, we were only able to replay attacks for 30 minutes. We attempted to replay both POST and GET requests. In particular, we replayed the welcome page which we land on after logging in, the My Data page, and the login page (before logging in). The results of the replay attacks are detailed in section 7.3.3.

6.4. Analysis of iOS App

The latest iOS version that can be completely jailbroken is 9.1, which was released on October 21, 2015. Since then, Apple has released iOS 9.2, 9.2.1, 9.3 and 9.3.1. Currently, there is no publicly released method to gain complete root access on devices running iOS 9.2+, which is the first step in reverse engineering iOS apps running on those systems. Snapchat takes advantage of the increasing security of the iOS system and its own iOS app by not allowing an old version of the app to run on an iOS version below 7. However, we did investigate Snapchat on a jailbroken iOS 8 device by analyzing its source code and bypassing SSL pinning to detect traffic.

6.4.1. Analyze and decrypt the executables

We SSHed into the jailbroken device and used `dumpdecrypted`⁹ to decrypt the Snapchat app. We then used `class-dump`¹⁰, which worked on the resulting unencrypted binary and gave us the interface files for Objective-C. These files include all the public and private methods that are defined by an executable.

6.4.2. SSL Killswitch to bypass SSL pinning

As discussed in Section 6.3.2, we used Charles Proxy to try to detect network traffic and decrypt packets going through the iOS app. However, Snapchat pins SSL certificates, meaning that the app saves the X.509 certificate unique to your device, and hence is able to detect when a user attempts to access the app through a different certificate. Thus, we were not able to decrypt requests or packages going through an un-jailbroken iPhone.

We then attempted to use Cypript--an application that allows one to unpack Snapchat's mobile code and alter configuration files. Cypript only works on jailbroken iPhones, so we tried using it on a jailbroken iPhone 6 running iOS 8. We were able to find several variables in the configuration of interest. In particular: `_allowsInvalidSSLCertificate`, which was originally set to false, and `_defaultSSLPinningMode`, which was set to 0. We attempted to change `_allowsInvalidSSLCertificate` to true, since the app's SSL pinning normally disallows unauthorized certificates. Our hope was that since Charles Proxy uses an unauthorized certificate, setting this boolean to true would allow Charles Proxy to bypass the SSL pinning. We also tried setting `_defaultSSLPinningMode` to 1.

Unable to use Cypript to bypass SSL pinning, we then attempted to use the iOS SSL Kill Switch, which is also only usable on jailbroken iPhones. Activating the Kill Switch for Snapchat allowed us to bypass SSL pinning. The details of our success are documented in section 7.4.2.

6.5. Analysis of Android App

We tested and analyzed version 9.28.1.0 (released on April 14, 2016) of Snapchat's Android app on devices running Android 5.0.1, 5.1.1, and 6.0. One device, a Galaxy Note 8 tablet running Android 5.1.1, was rooted and allowed us to carry out attacks that required root permissions.

6.5.1. Brute-Force Login

To attempt brute force login on an Android device, we used the app Bot Maker¹¹, which requires root access and allows users to write and execute automated scripts. Using accessibility mode to obtain the proper coordinates, we wrote a script to automate the login process with various input passwords. The attack was ultimately unsuccessful. Our results are documented in section 7.5.1.

6.5.2. Code Analysis

After downloading Snapchat from the Google Play Store, we first exported the APK to our laptops using Braveheart's Apk Extractor¹². Next, we decompiled the APK using two different methods: (1) using Android APK Decompiler¹³, a free online APK decompiler, and (2) following instructions¹⁴ to use the command line tool Apktool.¹⁵

The class files extracted using the first method were written in Java, but the class files produced by the second method were Smali files--assembled .dex bytecode used by the Android Runtime. Therefore, to more easily read the class files produced by Apktool, we followed instructions¹⁶ to use dex2jar¹⁷, a tool that can convert .dex code to a jar containing the APK's Java classes.

After decompiling the APK, we combed through the code, specifically looking for security- and encryption-related code. We also searched for common vulnerabilities such as hard-coded security information or access tokens. Our findings are described in section 7.5.2.

6.5.3. Reverse-Engineering

We also attempted to discover encryption keys and other sensitive information by modifying the decompiled APK and recompiling it. First, we modified the Java files produced by method 1 in section 6.5.2 and tried rebuilding the APK using the Android Studio IDE. We also tried modifying the Java files produced by the dex2jar tool and tried rebuilding the APK. Finally, we tried directly modifying the Smali files produced by method 2 in section 6.5.2 and recompiling using Apktool. We were unsuccessful; our results are documented in section 7.5.3.

6.6. Analysis of Web App

While Snapchat is primarily a mobile application, there exists a web application through which users can perform limited tasks related to their account settings. The login page for the web application is not accessible from Snapchat's home page; instead, users must navigate to `accounts.snapchat.com/accounts/login` and enter credentials. Once logged in, users may perform the following tasks:

- View or change Snapcode (QR code identifying user)
- Request user data, which includes account history, purchase history, and snap count
- Change password
- Unlock or delete account
- Set up Geofilters

To analyze the security of the web application, we looked for a few common vulnerabilities which are listed below. In addition, we explored the HTML using Chrome's inspect element feature to find hidden elements.

6.6.1. Common Web Attacks

First, we used an open source tool called Nikto to scan the website for matches with the open source vulnerability database. Finding none, we launched a variety of common web attacks against the Snapchat web application. These included cross-site scripting attacks, SQL injection, cross-site request forgery, and brute force login. We also checked against a database of sites that are known to be vulnerable to the DROWN attack.¹⁸ After all of this testing, we concluded that the web application is secure against these common attacks. Further details about the results are included in section 7.6.1.

6.6.2. Inspecting Elements

We used Inspect Element to check if any content that users should not be able to see were simply hidden away on the front end, in other words, marked as “hidden” on the HTML. By simply removing the hidden label, we were able to access some parts of the site that we shouldn’t be able to see (described further in section 7.6.2). We also looked for any buttons that were marked as unclickable, and attempted to activate them by changing the HTML.

We first looked at `geofilters.snapchat.com`, where one can create filters. Users must be logged in with an email-verified account to access this page. For this, we considered two Snapchat accounts, one that was verified and one that was not. We logged into each account and compared the resulting HTML.

We then looked at the My Data page, which can be accessed through `accounts.snapchat.com`. Similarly, this page can only be accessed by logged-in users with an email-verified account. We analyzed this page in a similar way to the Geofilters page.

7. Results and Vulnerabilities

7.1. Social Attacks

Snapchat does not host its own support service for Snapcash. Snapcash uses Square’s Square Cash API, so Snapcash support requests are directed to Square’s support system. Since we didn’t receive permission from Square to probe their system, we held off on any attacks that involved interacting with their system. Therefore, we did not thoroughly investigate the viability of the refund attack described in section 6.1.

However, we did try the location spoofing trick on an Android device. Many location-spoofing apps are available for download on the Google Play store; we used Fake Location Spoofer Free¹⁹. Apps such as these are easy to use and trick Snapchat into using different geofilters. The iOS app Phantom, described in section 3.1, provides location spoofing for iOS devices.

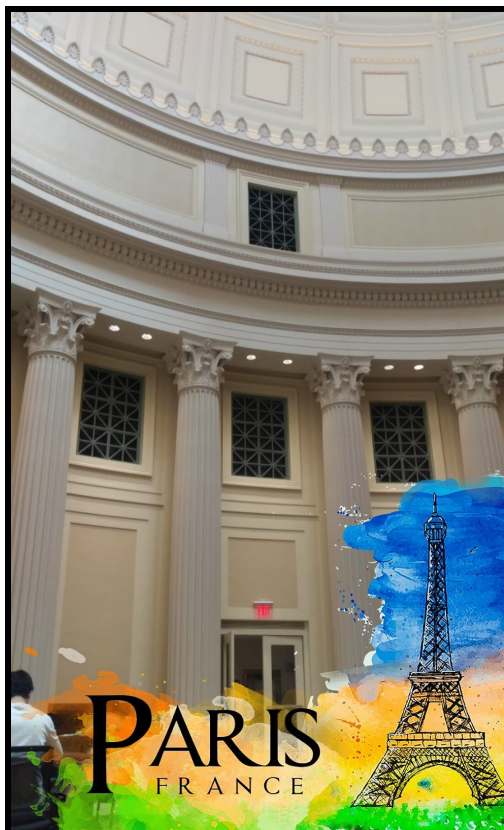


Figure 1: Has MIT's Barker Library moved to Paris?

We also attempted the screenshot attack described in 6.1 to try to bypass Snapchat's screenshot notifications. We found that an adversary can carry out this attack by following these steps: (1) receive a snap or message, (2) close all network connections, (3) view the content and screenshot it, (4) reopen network connections. In this case, the sender can see that the adversary viewed the content but does not see a notification that a screenshot was taken.

7.2. Database Dump

By exploring the database that was extracted from a rooted Android device, we found various tables that contained sensitive information. Suppose that an adversary Evan wants to view information about Alice and her Snapchat friends. Since the attack requires that Evan already has access to the phone, being able to carry out this attack means that he can view any information that Alice can view through the Snapchat app. Therefore, we focused on searching the database for information that would not normally be accessible even to authenticated users through the application.

The dumped database contains 74 tables, most of which are empty. The application likely uses most tables as a cache and regularly clears them out. However, we did find several tables with sensitive data. These are listed below in the following sections.

7.2.1. Friends Table

The friends table includes a list of all the user's friends, along with their username and display name. Most of the columns include data that is usually accessible through the application itself. However, the table also includes birthdays in plain text for users who have entered that information into their account. While the application does display an emoji on the friend's actual birthday, there is normally no way to manually obtain a list of birthdays.

The table also includes timestamps that indicate when the friendship was formed. This information is also not accessible via the application, and is sensitive data that should not be exposed to adversaries.

7.2.2. FriendsWhoAddedMe Table

The FriendsWhoAddedMe table contains information about the method of friendship formation. In particular, the table indicates for each friendship whether it was formed through Snapcode or username. This gives the adversary insight into the relationship between victims and all their friends. For example, a friendship formed using a Snapcode reveals that the two users most likely met in person. The table also indicates which of the two participants initiated the friendship, information not usually accessible to a third party.

7.2.3. ReceivedSnaps Table

The ReceivedSnaps table contains timestamps and usernames of senders for recently received snaps. This is usually not accessible from the app once the snaps have been opened.

7.2.4. SentSnaps Table

The SentSnaps table contains timestamps and usernames of recipients for recently sent snaps. This is usually not accessible from the app once the snaps have been opened.

7.2.5. MediaCache Table

The MediaCacheTable contains the file location and CBC encryption keys of cached files. However, without an initialization vector, we were unable to successfully decrypt the files.

7.3. Network Attacks

7.3.1. Man in the Middle

The Man in the Middle attack was successful on both the iOS app and the web app using the procedures outlined in section 6.3.1. In this section, we focus on what we were able to accomplish when attacking the web application.

After installing Charles and inserting breakpoints on the Snapchat site, we were able to intercept any packets sent to and from our device, including SSL packets. After pausing a packet's transmission, we could do the following:

- 1) Simply read, and then send the packet along as if nothing happened. This means that we can conduct a passive MitM attack, where we read the user's packets without the user ever realizing. This is arguably only useful if we're able to decrypt packets, which we were able to do (see section 7.3.2).
- 2) Abort (i.e. drop) the packet (Figure 2 below). This means that we can prevent a user's packets from ever reaching any Snapchat endpoint.

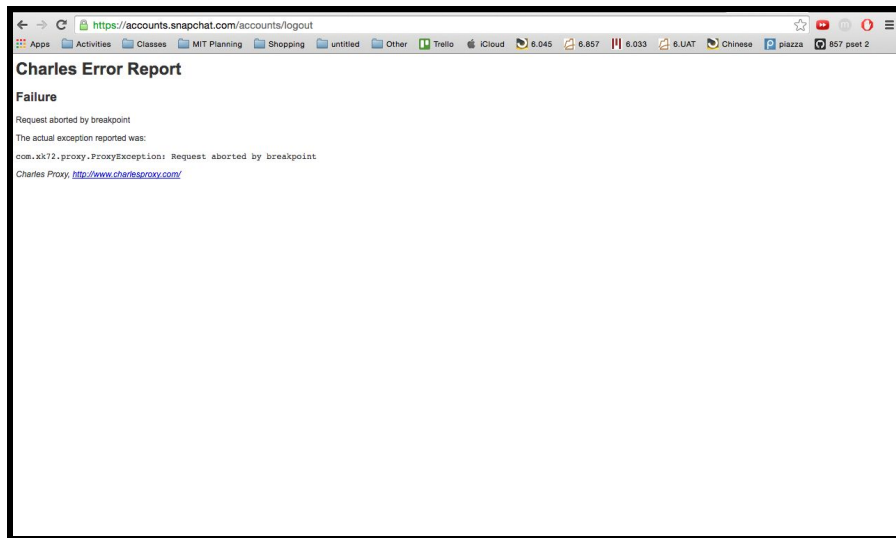


Figure 2: We are able to block users from accessing pages by dropping packets.

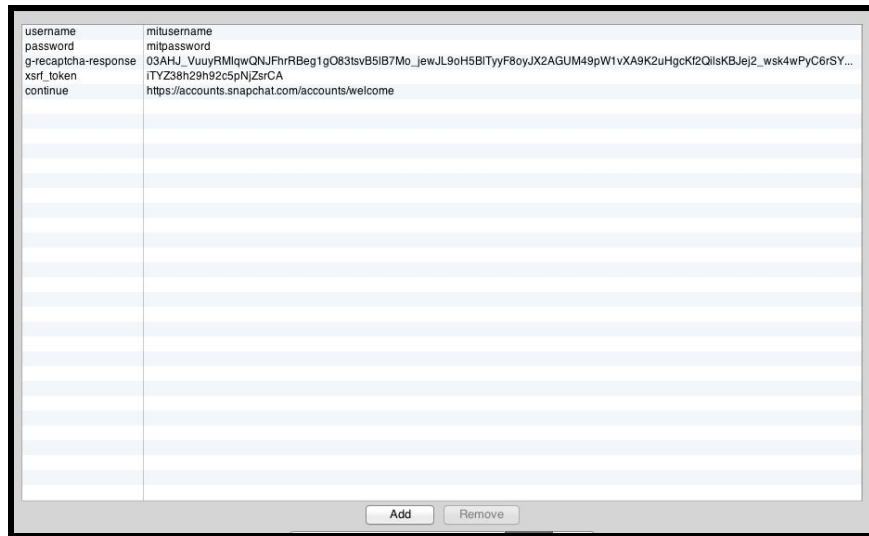
- 3) Edit the packet header and continue sending, meaning we could give false information on what browser or iOS device the packet was sent from.
- 4) Change the HTTP request URL and request type (GET, POST, etc.) and continue sending.
- 5) Edit packet request/response body and continue sending. If we delete the entire response body, the user would not be able to receive any content. After decrypting packets, we are also able to send incorrect content (described in 7.3.2).

7.3.2. Decrypting Packets

Our team used the methods outlined in section 6.3.2 to decrypt packets encrypted by TLS. We applied SSL proxying to both the website and iOS apps. While we were first met with an obstacle for the iOS app (SSL pinning), we were able to bypass SSL pinning and hence were able to obtain the same information as in the web app. Here we describe what information we were able to obtain and under what circumstances.

When a user logs in and sends a packet, we were able to stop the packet and decrypt it using Charles Proxy. Hence, we were able to see the user's username and password in plaintext. This worked for a

request sent by the same device that is running Charles Proxy, whether the request was sent by Safari, Chrome, or Chrome in incognito mode.



username	mitusername
password	mitpassword
g-recaptcha-response	03AHJ_VuuyrMIqwQNJFhrRBeg1gO83tsvB5IB7Mo_jewJL9oH5BITyyF8oyJX2AGUM49pW1vXA9K2uHgcKf2QilsKBJeJ2_wsk4wPyC6rSY...
xsrf_token	ITYZ38h29h92c5pNjZsrCA
continue	https://accounts.snapchat.com/accounts/welcome

Figure 3: Obtaining the Username and Password from a packet sent by Mac. Note that the username and passwords were changed to fake ones for this paper.

This attack requires that the user and adversary use the same device. However, this scenario isn't impossible. If a user unknowingly logs into Snapchat on an adversary's device running Charles Proxy, the adversary could record all packets that were sent to and from the device, and later check those packets for sensitive information. Alternatively, the adversary could send malware to their victim that downloads and activates Charles Proxy and then forwards the packets to a remote server.

For the iOS app, we were able to obtain a user's login credentials remotely. However, this attack requires certain conditions: the iPhone must be jailbroken, have downloaded iOS SSL Kill Switch, and have changed the WLAN server to point to the adversary's IP address.

Besides the username and password, we were also able to get the sender's Snapcode. We intercepted snaps and chats, but could only see the sender's username and the type of device the request had been sent from. We were unable to find any information about the geofilter, chat content, snap image, or snap caption. When we intercepted a snap, there was a huge block of ciphertext that we were unable to decode.

We finally tried to decrypt packets for the My Data page on the Snapchat Accounts site. This page is accessible to anyone who has logged in and whose account had been previously verified by email. This page contains a link to a zip file containing the user's private data. We were unable to access any information about the file besides the file's name, which appeared to be a random integer.

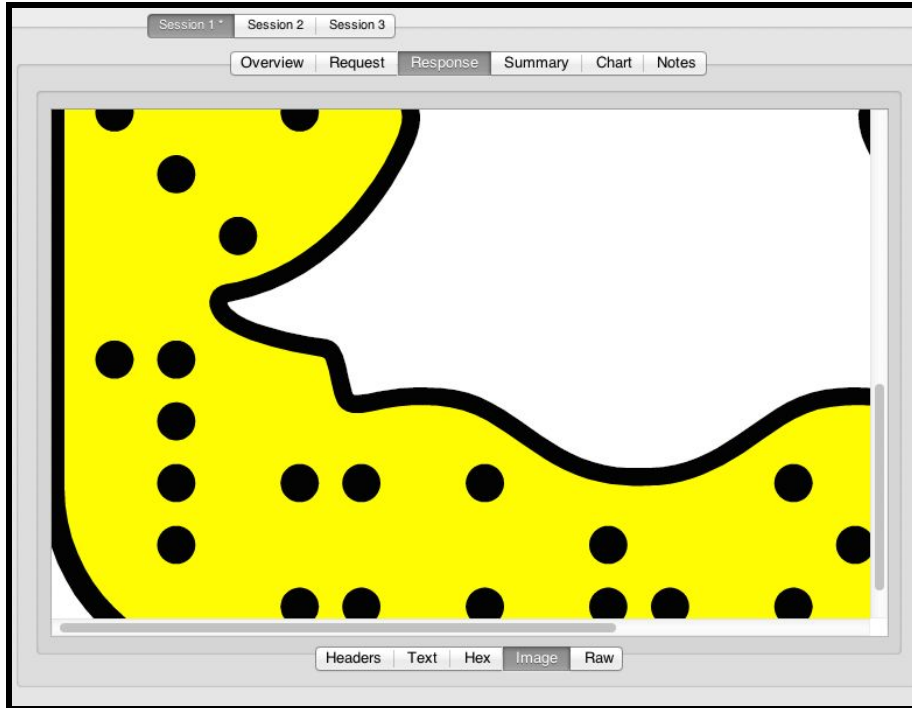


Figure 4: We were able to obtain the Snapcode of a user.

7.3.3. Replay Packets

Using the method described in 6.3.3, we were able to successfully record packets and replay them as many times as desired, but only within the current session. In other words, if a page requires the user to be logged in, we can only replay the request before the user logs out. Whenever we replayed a request successfully, we were able to receive the HTML response, and hence were able to receive the contents of the page. Below is a figure of the replay of the user's welcome page.

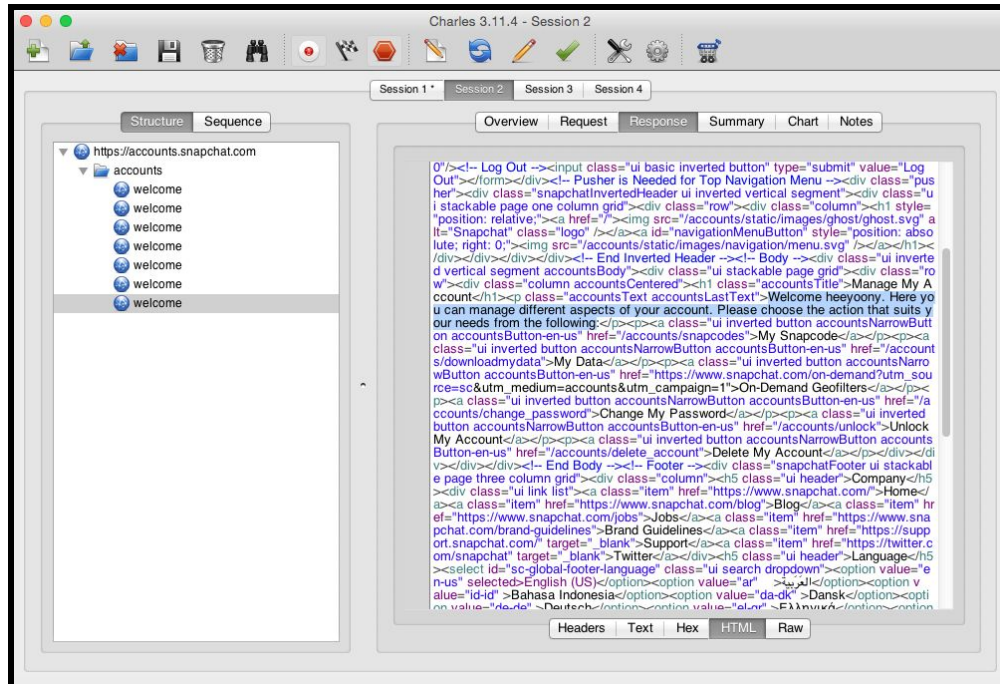


Figure 5: Replay Attack on Welcome page. Able to receive the HTML content again.

7.4. Analysis of iOS App

7.4.1. Analyze and decrypt the executables

Our investigation into the disassembled Snapchat iOS framework showed that the app is in fact very secure. It does not contain any hard-coded sensitive information, has methods to detect if the device is jailbroken and if the property list is safe, and uses elliptic curve cryptography to manage signature and encryption.

Snapchat does not publish an official API, but we looked into publicly available third-party Snapchat APIs²⁰. We noted that only one of them could potentially work, and it explicitly requires Snapchat accounts created via an Android client because Snapchat handles iOS accounts separately and an account will most likely be disabled or flagged as suspicious if using third-party application²¹. We credited this to the high security of Snapchat’s iOS app.

7.4.2. SSL Killswitch to bypass SSL pinning

As written in the methodology, we found two variables after unpacking the iOS app using Cypcript: `_allowsInvalidSSLCertificate`, which was originally set to false, and `_defaultSSLPinningMode`, which was set to 0. We changed `_allowsInvalidSSLCertificate` to true, and `_defaultSSLPinningMode` to 1.

When attempting to reset those variables in Cypcript, we were still unable to send packets on the Snapchat iOS app under the Charles SSL Proxying feature. However, when we used the iOS Kill Switch, we were able to use SSL proxying. This means that we were able to remotely intercept packets sent by an iPhone.

After intercepting the packet, we were able to successfully decrypt the packet and view its contents before encrypting again and delivering the packet to its intended recipient. We were also able to drop or modify packets as we wished.

We were able to acquire the following information:

1. As a user logs in on an iPhone, we were able to obtain a packet with the decrypted username and password. In other words, we were able to see both items in plaintext after Charles Proxy fully decrypted the packets. (Figure 6 below)
2. For any other request, we were able to see the sender's username in plaintext.

When we attempted to send a chat under SSL proxying, we were unable to see the message body. Similarly, we were unable to see neither the message nor the image when sending snaps. Instead, we saw a block of what appears to be ciphertext, which could be the encrypted snap.

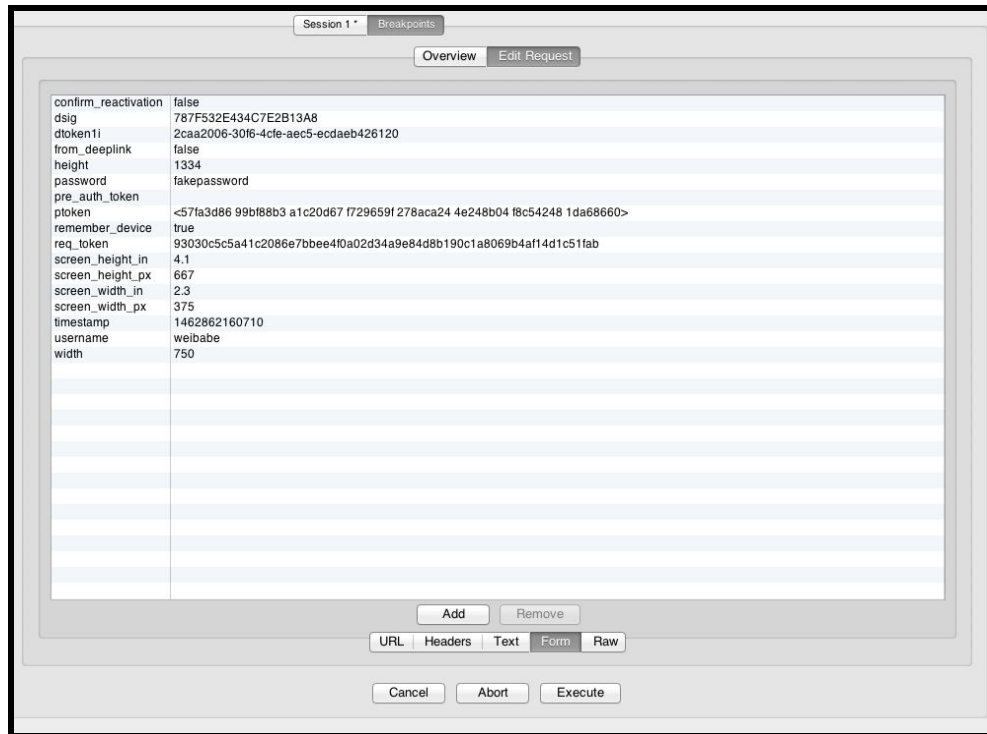


Figure 6: Intercepting a decrypted, login packet. Note the username and password fields.

7.5. Analysis of Android App

When analyzing the Android app, we investigated not only what vulnerabilities an adversary could exploit but also what they could glean from the app's source code.

7.5.1. Brute-Force Login

While we were able to automate the login process using Bot Maker on a rooted Android device, the brute-force attack was ultimately unsuccessful. After a few unsuccessful attempts (around 10), the application displays the following pop-up message:

“Oh no! Your login temporarily failed, so please try again later. If your login continues to fail, please visit `support.snapchat.com/a/failed-login` :)”

We dismissed the pop-up modal and continued to run the automated login script. After several more attempts, the app responded with a different message:

“Due to repeated failed login attempts or other suspicious activity, login for your account is temporarily disabled. Please try again later.”

After this, the account needed to be unlocked via the web application. From this, we concluded that the Android application is indeed secure against brute-force login attacks.

7.5.2. Code Analysis

Snapchat’s class files are obfuscated using a service like ProGuard--classes are written in Java, but class and variable names are changed to make the code harder to understand. This obfuscation made understanding the interaction of the app’s modules somewhat difficult. Nevertheless, we were still able to understand the source code and look through it for vulnerabilities.

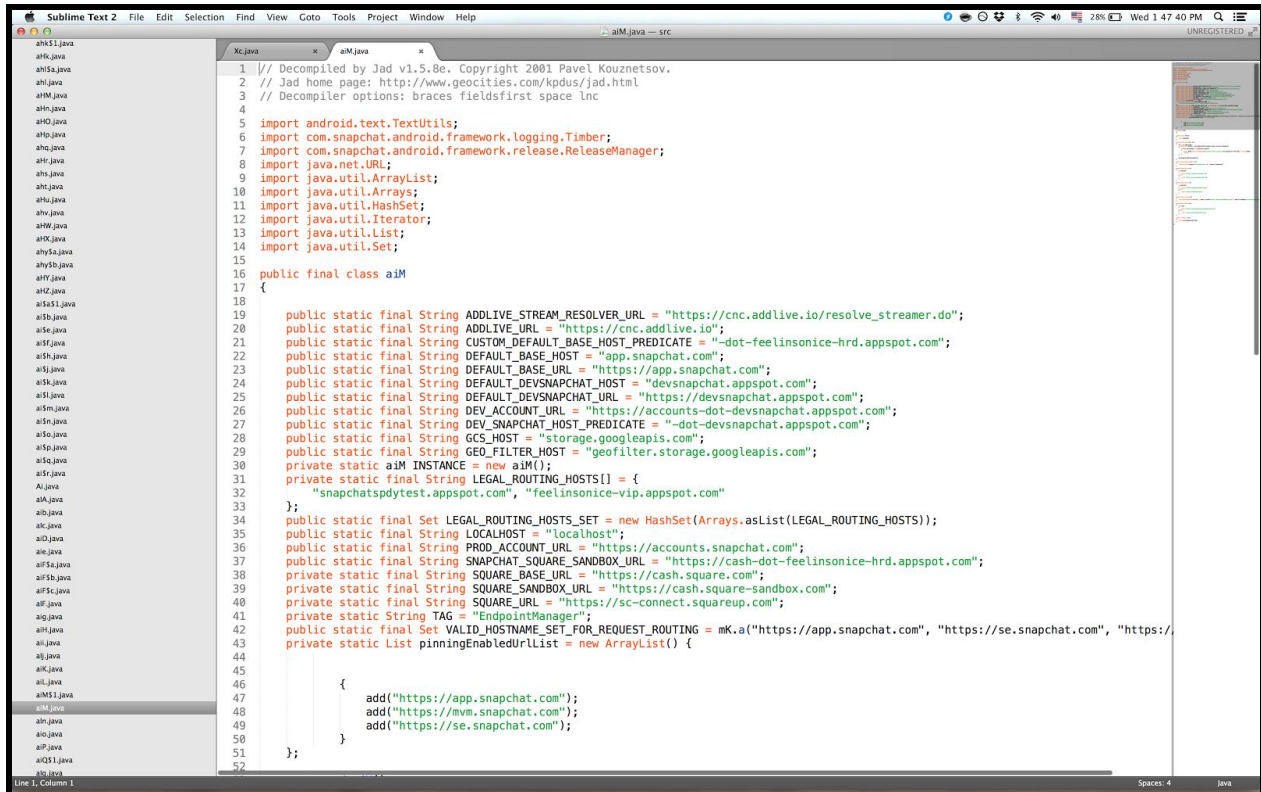
We found that Snapchat’s Android app uses HTTPS/TLS to secure its communications. The app appears to use both custom and third-party modules for encryption. To make sure users cannot access media in their phone caches, Snapchat CBC-encrypts all media before caching it. While we found CBC encryption keys in Snapchat’s SQL database (section 7.2), we could not find the IV and hence could not decrypt the content in our phones’ caches.

We found that Snapchat hard-codes X.509 certificates used for the TLS protocol. If an adversary Evan discovered these certificates, it is possible that he could gain information about Snapchat’s digital signature and subsequently use this signature to impersonate Snapchat when communicating with honest users.

```
11 public final class Xc
12 {
13
14
15     private static KeyStore a;
16
17     public static KeyStore a()
18     {
19         Xc;
20         JVM INSTR monitorenter ;
21         KeyStore keystore;
22         if (a != null)
23         {
24             break MISSING_BLOCK_LABEL_213;
25         }
26         keystore = KeyStore.getInstance(KeyStore.getDefaultType());
27         CertificateFactory certificatefactory;
28         int i;
29         try
30         {
31             keystore.load(null);
32         }
33         catch (Exception exception1) { }
34         certificatefactory = CertificateFactory.getInstance("X.509");
35         i = 0;
36
37     _L2:
38         if (i >= 21)
39         {
40             break; /* Loop/switch isn't completed */
41         }
42         ByteArrayInputStream byteArrayinputstream = new ByteArrayInputStream((new String[] {
43             "-----BEGIN CERTIFICATE-----\nMIIDtzCAPqgAwIBAgI0DDfg5RfYRv6P5ND8G/AwOTANBgkqhkiG9w0BAQUFADB\l\nM0swCQYDV00EwJVUzEVMBMGA1UECH
44             "-----BEGIN CERTIFICATE-----\nMII CPCCCAAUCEDyRmcsf9LAbDpq40ES/Er4wDQYJKoZIhvcNAQEFBQAwXzELMAKG\nA1UEBHMVVMxZzAVBGNVBAOTDzL1cm
45             "-----BEGIN CERTIFICATE-----\nMII E2CCBEgAwIBAgIEEN0rS0ZANBgkqhkiG9w0BAQUFADCBwzELMAKG\nA1UEBHMVVMxZzAVBGNVBAOTDzL1cm
46         })[i].getBytes());
47         X509Certificate x509certificate = (X509Certificate)certificatefactory.generateCertificate(bytearrayinputstream);
48         byteArrayinputstream.close();
49         keystore.setCertificateEntry(UUID.randomUUID().toString(), x509certificate);
50         i++;
51         if (true) goto _L2; else goto _L1
52     _L1:
53         a = keystore;
54         keystore = a;
55         Xc;
56         JVM INSTR monitorexit ;
57         return keystore;
58     Exception exception;
59     exception;
60     throw exception;
61 }
62
```

Figure 7: We found hard-coded certificates presumably used for the TLS protocol in the Android source code.

We also found a file with many hard-coded developer endpoints. These endpoints are not available to the public and can only be accessed using an API token. Although we were unable to find any hard-coded tokens, hard-coding internal endpoints may make it somewhat easier for an attacker to target their respective systems.



```
1 // Decompiled by Jad v1.5.5.Be. Copyright 2001 Pavel Kouznetsov.
2 // Jad home page: http://www.geocities.com/kpdus/jad.html
3 // Decompiler options: braces fieldsfirst space lnc
4
5 import android.text.TextUtils;
6 import com.snapchat.android.framework.logging.Timber;
7 import com.snapchat.android.framework.release.ReleaseManager;
8 import java.net.URL;
9 import java.util.ArrayList;
10 import java.util.Arrays;
11 import java.util.HashSet;
12 import java.util.Iterator;
13 import java.util.List;
14 import java.util.Set;
15
16 public final class aiM
17 {
18
19     public static final String ADDLIVE_STREAM_RESOLVER_URL = "https://cnc.addlive.io/resolve_streamer.do";
20     public static final String ADDLIVE_URL = "https://cnc.addlive.io";
21     public static final String CUSTOM_DEFAULT_BASE_HOST_PREDICATE = "-dot-feelinsonice-hrd.appspot.com";
22     public static final String DEFAULT_BASE_HOST = "app.snapchat.com";
23     public static final String DEFAULT_BASE_URL = "https://app.snapchat.com";
24     public static final String DEFAULT_DEVSNAPCHAT_HOST = "devsnapchat.appspot.com";
25     public static final String DEFAULT_DEVSNAPCHAT_URL = "https://devsnapchat.appspot.com";
26     public static final String DEV_ACCOUNT_URL = "https://accounts-dot-devsnapchat.appspot.com";
27     public static final String DEV_SNAPCHAT_HOST_PREDICATE = "-dot-devsnapchat.appspot.com";
28     public static final String GCS_HOST = "storage.googleapis.com";
29     public static final String GEO_FILTER_HOST = "geofilter.storage.googleapis.com";
30     private static aiM INSTANCE = new aiM();
31     private static final String LEGAL_ROUTING_HOSTS[] = {
32         "snapchatspytest.appspot.com", "feelinsonice-vip.appspot.com"
33     };
34     public static final Set LEGAL_ROUTING_HOSTS_SET = new HashSet(Arrays.asList(LEGAL_ROUTING_HOSTS));
35     public static final String LOCALHOST = "localhost";
36     public static final String PROD_ACCOUNT_URL = "https://accounts.snapchat.com";
37     public static final String SNAPCHAT_SQUARE_SANDBOX_URL = "https://cash-dot-feelinsonice-hrd.appspot.com";
38     private static final String SQUARE_BASE_URL = "https://cash.square.com";
39     private static final String SQUARE_SANDBOX_URL = "https://cash.square-sandbox.com";
40     private static final String SQUARE_URL = "https://sc-connect.squareup.com";
41     private static String TAG = "EndpointManager";
42     public static final Set VALID_HOSTNAME_SET_FOR_REQUEST_ROUTING = mk.a("https://app.snapchat.com", "https://se.snapchat.com", "https://
43
44
45
46
47     {
48         add("https://app.snapchat.com");
49         add("https://mvn.snapchat.com");
50         add("https://se.snapchat.com");
51
52     }
53
54 }
```

Figure 8: We found many hard-coded developer endpoints in the Android source code.

7.5.3. Reverse-Engineering

Our attempts to recompile and run Snapchat after editing its source code were unsuccessful. Using the first two methods described in 6.5.3, we were unable to rebuild the project using Android Studio's compiler. Using the third method, we successfully recompiled the source code both with and without edits.

However, we were only able to run the version recompiled without edits--the edited app crashed on login. Nevertheless, we suspect that we could tweak the Smali code in such a way so that the app would print out sensitive data such as encryption keys or user information.

Additionally, an adversary could use this technique to create a malicious version of the Snapchat app that records login credentials, completes unauthorized Snapcash transactions, sends spam, etc.

7.6. Analysis of Web App

In our testing, we found that the web application was highly secure. Initial scans using open source tools found no known vulnerabilities, and our own tests did not yield any substantial results. The results of these tests are described in the following sections. We also note that the website follows sound security patterns. The single point of entry is well secured against common web attacks, and users are asked to

re-authenticate to perform any serious task. Users are also notified via e-mail when they request data, thereby enabling the detection of compromised accounts.

7.6.1. Common Web Attacks

Our initial scan with the nikto tool found no matches with the open source vulnerability database. We attempted to perform cross site scripting and SQL injection by inserting code snippets in forms throughout the website, but all inputs were sanitized properly. The pages contained `xsrftoken` elements that prevented our cross-site request forgery attacks. Finally, the login page uses Google reCAPTCHA to thwart any brute force login attacks.

7.6.2. Inspecting Elements

Even without an email-verified account, we were able to see parts of the page that the unverified user shouldn't be able to access. For example, we were able to see the progress bar for creating a geofilter and a dropdown menu (albeit empty) that belongs to the geofilter form. Unverified users should not be able to access these elements. However, we were unable to submit a geofilter without verifying an email address. Below is the original page, and the page after un hiding elements.

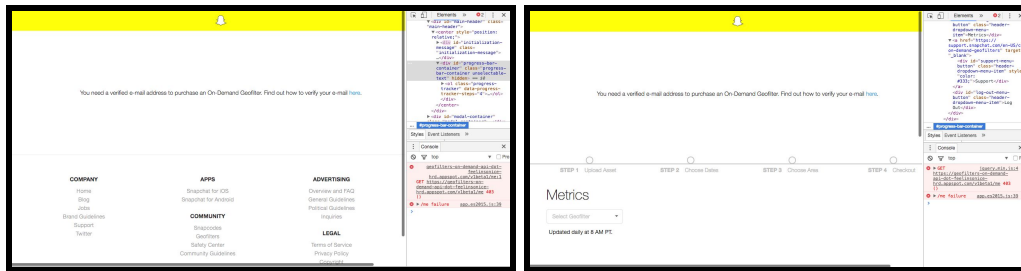


Figure 9: Unhiding hidden elements. Before hiding (left) and after (right).

The goal for analyzing the My Data page was to be able to obtain the zip file containing personal data (in the hope that this was simply marked as hidden in the HTML). This was unsuccessful. It appears that the app fetches the file only for users who have email-verified accounts.

Using the email-verified account, we attempted to create an invalid Geofilter and submit it. However, we couldn't enable the Submit button until all fields (submitting the filter image, setting the geofilter dates and location, and filling out credit card information) were completed, even after modifying the HTML.

8. Recommendations

8.1. Avoid Hard-coding

It is important not to underestimate the determination of an adversary. Although combing through decompiled source code for vulnerabilities is tedious, the discovery of any sensitive information hard-coded in plain text would make it worth it. For example, if an adversary Evan uncovered Snapchat's

CBC encryption IV's and keys, he could decrypt and view cached Snapchat media on his phone, thereby bypassing Snapchat's content-viewing restrictions.

As described in section 7.5.2, we found certain hard-coded information that could possibly allow an attacker to impersonate Snapchat or attack systems that interact with Snapchat. One way to protect against such attacks is to not hard-code this sensitive information. Instead, the app could access the information via a call to Snapchat's internal API. If hard-coding is absolutely necessary, the information could be encrypted first so that it is not hard-coded in plain text.

8.2. Encrypt Information Stored in SQL Database

While the SQL database on Android devices is not normally accessible to users, it is not safe to assume that information stored there is always secure. In section 7.2, we showed that rooted users can access these tables and view information that is not usually accessible even to authenticated users. We recommend that this data be encrypted so that additional information is not leaked even if the adversary is able to gain control of the victim's phone.

This may come at a cost in performance, since encryption will take extra computation. A good balance might be to only encrypt information that is not accessible via the application, since an adversary will be able to open the application if given control of the device. Section 7.2 includes a list of tables in the database where we found normally inaccessible information.

8.3. Network Attack Protection

By decrypting packets using Charles Proxy, we were able to obtain the username and password as someone logs into the iOS Snapchat app. There are a lot of caveats to making this attack work. For example, the phone must be jailbroken, and the user must have manually reconfigured WLAN settings and have iOS SSL Kill Switch installed. However, the scenario is not impossible. Jailbreaking is growing more popular because one can access apps or features that are normally unavailable.²² A persistent adversary may then release an app that also forces the iOS Kill Switch to be downloaded, and then perhaps change network settings of the target.

Therefore, since Charles Proxy can decrypt packets encrypted by HTTPS/TLS under the above conditions, we recommend that Snapchat use additional means to encrypt login credentials before they are sent over the network.

8.4. Hidden Elements

In section 7.6.2, we show that an unverified user can uncover some elements of Snapchat's web app that should only be visible to verified users. We recommend that the app only retrieve information that the current user is allowed to see. For example, in the case of the Geofilters page, an unverified user should not be able to see any part of the Geofilters form. Therefore, the page should only load this form if the user is verified.

8.5. User Authentication

In November 2015, Snapchat added an “official story” feature to verify celebrity accounts. However, adding a celebrity by username or official story still seems confusing--it is possible to add a fake celebrity account by mistake, especially when the celebrity does not have an official story. Our recommendation is to display both the username and official story fields when people search for users. This way, if a user searches for a celebrity who does not have a Snapchat-verified account, the empty row under the “Official Stories” section will inform people that the celebrity does not use Snapchat.

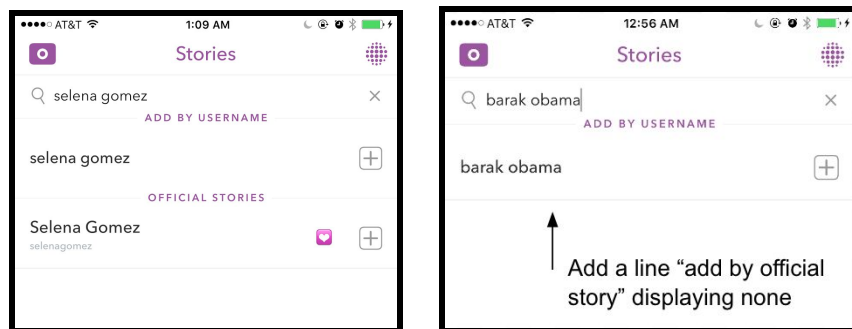


Figure 10: Official story feature. We recommend having an empty official story row for celebrity accounts with no official story posted.

9. Conclusion

The scope of this security analysis covered Snapchat’s Android, iOS, and web applications with a particular focus on jailbroken iPhones and rooted Android devices. While the Snapchat app seems relatively secure, we determined that under very specific scenarios, a persistent adversary could remotely obtain a user’s login credentials as they log in. Specifically, the adversary could gather this information when the user logs in on a jailbroken iPhone with particular network configurations and the iOS Kill Switch installed or when the user logs in on the adversary’s laptop running Charles Proxy. Additionally, we were able to open the Android app’s SQL database and thereby obtain information that a user should not normally be able to access. By protecting against these attacks, Snapchat can further enforce user privacy and content-viewing restrictions.

10. Acknowledgements

Our team would like to give special thanks to Professor Ron Rivest, who connected us with Dr. Moti Yung who is part of the Snapchat Security Team. We would also like to thank the Snapchat Security team for giving us permission to analyze the app, in particular Dr. Moti Yung whom we initially contacted and Jad Boutros, the Chief Security Officer at Snapchat.

We would like to thank the entire 6.857 staff, especially Cheng Chen who took the time to meet with us, answered our questions, and pointed us in helpful directions. Finally, we thank Lynda Tang who allowed us to use her Windows laptop for installing Fiddler.

Reference

1. Robert Hackett. *Snapchat's Video Traffic Is Catching Facebook*. Jan 11, 2016. URL: <http://fortune.com/2016/01/12/snapchat-facebook-video-views>
2. Snapchat account page. 2016. URL: <https://accounts.snapchat.com>
3. Jeff Benjamin. *Jailbreak tip: spoof your location in Snapchat to use any geofilter*. Jul 14, 2015. URL: <http://www.idownloadblog.com/2015/07/14/snapchat-spoof-location-filters-phantom>
4. Gibson Security. *Snapchat - GibSec Full Disclosure*. Dec 25, 2016. URL: <http://gibsonsec.org/snapchat/fulldisclosure>
5. Violet Blue. *Predictably, Snapchat user database maliciously exposed*. Jan 1, 2014. URL: <http://www.zdnet.com/article/predictably-snapchat-user-database-maliciously-exposed>
6. Decrypting Android Snapchat images. URL: <https://github.com/programa-stic/snapchat-decrypt>
7. David Uskhin, Samuel Edson, Santhosh Narayan, Akshit Dua. *Assessing Security of iOS Apps*. May 2015. URL: <http://courses.csail.mit.edu/6.857/2016/files/sukhin-edson-narayan-dua.pdf>
8. DB Browser for SQLite. URL: <http://sqlitebrowser.org>
9. Dumpdecrypted. URL: <https://github.com/conradev/dumpdecrypted>
10. Class-dump. URL: <https://github.com/nygard/class-dump>
11. [ROOT] Bot Maker for Android. URL: <https://play.google.com/store/apps/details?id=com.frapeti.androidbotmaker&hl=en>
12. Apk Extractor. URL: <https://play.google.com/store/apps/details?id=braveheart.apps.apkextract&hl=en>
13. Online Android decompiling tool. URL: <http://www.decompileandroid.com>
14. *How to Decompile and Compile Android APK's on A Mac Using Apktool*. URL: <http://ilikekillnerds.com/2014/09/how-to-decompile-and-compile-android-apks-on-a-mac-using-apktool>
15. Apktool: a tool for reverse engineering Android apk files. URL: <http://ibotpeaches.github.io/Apktool>
16. *Stackoverflow: Is there a way to get the source code from an APK file?* URL: <http://stackoverflow.com/questions/3593420/is-there-a-way-to-get-the-source-code-from-an-apk-file>

17. Tools to work with android .dex and java .class files. URL:
<https://github.com/pxbl988/dex2jar>
18. *The DROWN Attack*. URL: <https://drownattack.com>
19. Fake Location Spoofer Free. URL:
<https://play.google.com/store/apps/details?id=com.incorporateapps.fakegps.fre&hl=en>
20. Snapchat Official Blog. *Third-Party Applications and the Snapchat API*. URL:
<http://snapchat-blog.com/post/99998266095/third-party-applications-and-the-snapchat-api>
21. Nodejs client for the unofficial Snapchat API. URL:
<https://github.com/fisch0920/snapchat>
22. John Paul Titlow. *Nearly 1 Million People Jailbroke Their iPhone or iPad Over the Weekend*.
Jan 24, 2012.
http://readwrite.com/2012/01/24/1_million_jailbreak_iphone_4s_ipad_2_ios_5