Security Analysis of GrubHub

“There is such a thing as a free lunch!”

Cynthia Jing, Rebecca Krosnick, Shiyang Liu, Kimberly Toy

13 May 2015

1. Abstract

We have performed a security analysis on GrubHub, an online food ordering service. GrubHub has a large user base and access to sensitive information such as home addresses, phone numbers, and credit card information, thus any security flaws could cause a real impact on users. In this paper, we will give an overview of the GrubHub environment, how orders and users are verified, and possible vulnerabilities we found. We split our analysis between the web app and the Android app during which we found many things GrubHub did correctly, as well as a few critical issues that allowed us to refund a previously completed order.

2. Introduction

GrubHub is a rapidly growing service that provides online and mobile food ordering platforms to connect over 3.4 million active users [1] with 35,000 take-out restaurants. Users can place orders with nearby restaurants for either take-out or delivery and pay by credit card or with PayPal. GrubHub stores basic user account information, including full name and email, during the account registration process and includes the option to store delivery addresses and payment information for convenience during checkout.

For a sense of scale, in the first quarter of 2015, GrubHub’s average number of daily orders totalled 234,700, and profits were projected at $84 million. The application itself processed a total of $590 million dollars from food sales.

Due to the nature of this application, there is enough information stored about each user to pose a threat to the private payment information and online identity of a user if attackers gained access to any of this information. Managing the security of GrubHub’s platforms is critical in order to protect the confidentiality of its many users and to maintain the success of their business operations.

3. Responsible Disclosure

The research in this paper was performed with the permission of the company under a responsible disclosure policy. Our analysis audits GrubHub’s new web application, https://labs.grubhub.com, and its Android mobile application, version 5.0 R17.

Some of the issues discussed below disclose sensitive security issues, some of which may adversely affect GrubHub users or the business itself. Before attempting to receive a refund (see Section 5.2) we asked the GrubHub security operations team for additional permission to perform this test. Upon the completion of the Spring 2015 semester we will contact GrubHub to disclose the remainder of the security vulnerabilities that we have discovered.

4. Architecture Overview

GrubHub API

GrubHub does not advertise a public API that is openly available for use, but it does have a public API for restaurant information so that developers can easily access menu items and restaurant information for their own applications. We were able to recreate much of GrubHub's private API which is documented in Appendix A.

GrubHub Web Application

The site supports HTTPS, ensuring secure transport of private information such as passwords and credit card details. The site allows users to create and modify account information, view restaurant information, and make orders by using the GrubHub API. Credit card and PayPal transactions are performed on the server after orders are placed.

While a user is logged in to their GrubHub account, much of their account information and order history is copied into the browser's local storage. Thus, one can easily obtain this information by accessing local storage, which is as simple as running this line of code: `localStorage.getItem(localStorage.key(10))` in the browser console.

GrubHub Android Application

The mobile app provides much of the same functionality as the web interface, such as ordering and storing personal payment and address information. Most of these operations are performed by interfacing with the GrubHub API. Exceptions include adding credit card numbers, which are kept on the Android device's local SQLite storage, and making PayPal transactions, which is done directly from the Android app.

5. Vulnerabilities

5.1 Summary of Methodology

For our analysis on the GrubHub web application, we logged network traffic while using the site. We were able to recreate a general summary of the private API that serves the web app and audit it for potential security flaws. We also tested the website for general security issues, as outlined by the OWASP top 10 [4].

For the GrubHub Android application, we downloaded the app, which is freely available on the Google play store. We were able to audit the app by using an APK decompiler[3] in order to retrieve the application's source code. In the source, we checked to see how payments were made, how sensitive information was handled, and looked for areas where encryption schemes were being used.

5.2 Hardcoded PayPal Keys

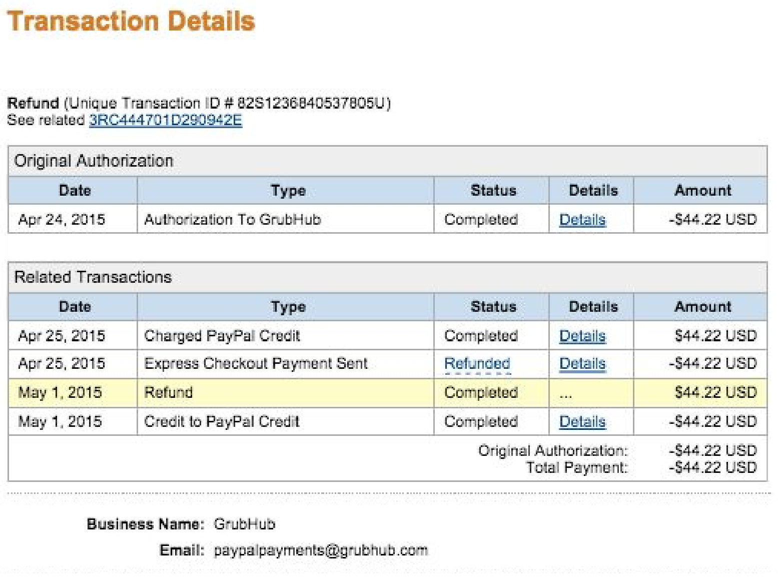
In the decompiled Android application, we found that GrubHub had hard coded their private PayPal API keys into the application's `strings.xml`, located in /res/values/. This set of API credentials: a username, password, and signature, uniquely identifies each PayPal business user and is linked to the user's PayPal account [5].

Using these credentials, we were able to gain access to the PayPal Express Checkouts API [6], including methods such as:

- Checking the balance of the GrubHub corporate PayPal account
- Searching through previous transactions, which yields customer PayPal emails, transaction amounts, and unique transaction IDs.
- Obtaining transaction details per ID, including the payer's full name and address.
- Refunding transactions given an ID.

The ability to refund transactions lent itself to the possibility of obtaining a free lunch, so we set out to exploit this vulnerability with the permission of the GrubHub security team. We placed an order on the GrubHub website and completed the checkout process with a PayPal payment, allowing us to look up the transaction ID via curl calls to the express checkout API.

Once the transaction was marked as complete, we made subsequent calls to refund the order, and the money was returned back to our account immediately, as seen in Figure 1. The related curl calls can be seen in Appendix C.



Transaction Details

Refund (Unique Transaction ID # 82S12368405378O5U)
See related 3RC444701D290942E

Original Authorization

Date	Type	Status	Details	Amount
Apr 24, 2015	Authorization To GrubHub	Completed	Details	-$44.22 USD

Related Transactions

Date	Type	Status	Details	Amount
Apr 25, 2015	Charged PayPal Credit	Completed	Details	$44.22 USD
Apr 25, 2015	Express Checkout Payment Sent	Refunded	Details	-$44.22 USD
May 1, 2015	Refund	Completed	...	$44.22 USD
May 1, 2015	Credit to PayPal Credit	Completed	Details	-$44.22 USD
			Original Authorization:	-$44.22 USD
			Total Payment:	-$44.22 USD

Business Name: GrubHub
Email: paypalpayments@grubhub.com

Figure 1. Successful PayPal refund of Grubhub order.

Though it would be clear after performing an audit which PayPal account money is being diverted into, a malicious user could create GrubHub and PayPal accounts with fake user information in order to abuse this vulnerability without directly implicating himself. In order to severely threaten GrubHub's service, the attacker could also choose to refund an enormous number of transactions, which would create a substantial impact on GrubHub's finances and not directly implicate the attacker.

5.3 Permanent Tokens

When a new user creates an account on GrubHub, a unique user token is generated. The token is a permanent 16-byte hexadecimal string. It does not expire after a session nor does it change when the user changes email or password. With this token, anyone can access all of the user account information by making the relevant GrubHub API call and passing the token as a URL parameter.

If an adversary has access to a user's token, they could look up the user's email address, physical address, and internal credit card id numbers, amongst other information. Furthermore, they could add a new address, order food from GrubHub, and pay with the victim's saved credit card. To confirm that an adversary can place an order knowing only the victim's permanent token, we successfully used purely API calls with the token to place an order from one of our accounts and pay using a saved credit card. Even if the victim changes his password to protect his account, the adversary will still have access to the account since the token does not change or expire.

This permanent token is stored, unencrypted, in a user's local storage. Persistent local storage is a feature of HTML5 that allows for the storage of named key/value pairs locally, within the client web browser [7]. The storage is domain specific, so only pages that fall within a certain domain or its subdomains has access to the items stored in their domain. One benefit to local storage is that there is no longer a need for sending a cookie every time a user wishes to visit a new page. However, by storing all relevant user information unencrypted in local storage, this opens the user up to several potential attack vectors.

If a computer is public or left unattended and a GrubHub user has not logged out of their account, their account data will be available local storage. An adversary could easily obtain this data, including the user token which could be used later when the user is not logged in or even on a different machine. The success of this vulnerability depends on an attacker's ability to gain control of a user's device.

If there were cross-side scripting vulnerabilities, an attacker could insert JavaScript code to force a user to send his token to another untrusted site. We attempted non-persistent XSS exploits by entering Javascript into the restaurant search tool. However, GrubHub sanitizes their inputs and the resulting redirected URL has all JavaScript escaped. We also looked at persistent XSS vulnerabilities through their restaurant review form. This method of attack failed because reviews comments are processed by hand and are not immediately posted on the site.

Another possible attack vector would include DNS spoofing to divert traffic to a malicious site. Because the user still believes that he is in the GrubHub domain, the malicious site will be able to access the local storage associated with the GrubHub domain.

5.4 Inability to Delete Accounts

[Edit: Grubhub **does** provide the ability to delete an account upon request.]

GrubHub does not allow users to delete accounts. If a user no longer wishes to use GrubHub's service, the best that the user can do is leave the account as inactive. However, the user's order history including delivery address cannot be deleted from the system. The order history along with other user saved information can be exposed to adversary if the account is compromised.

5.5 Failure to Limit Password Retries

The GrubHub website requires an email and password for login, where the password is required to be longer than eight characters and cannot be too simple like 'password' or '12345678'. Many sites prevent brute force access to accounts by displaying a captcha after a number of failed login attempts. However, neither the GrubHub site UI nor the API endpoint for authentication have implemented any mechanism for limiting password retry attempts.

This becomes a significant concern for GrubHub, as their failure to secure their PayPal API credentials allows a malicious user to easily obtain thousands of PayPal emails, which are potentially the same emails used to register for a GrubHub account. The adversary can then use brute force mechanisms to gain access to user accounts, such as by trying a list of the most common passwords for each email.

5.6 Storing Private Information on Local Android Devices

The Android application stores some personal user information in the device's SQLite storage. Some of this data is stored temporarily. For example, user PayPal credentials are only stored in the SQLite database while the user is completing a transaction and are removed afterward.

However, some of the data stored in the SQLite database is stored permanently, and this could pose some risks. For example, when a user saves postal addresses and credit card numbers in the Android app, this information is stored in the SQLite database, remaining there once the user logs out. The credit card numbers are encrypted, but the CVV number and zip code associated with the credit card number are not encrypted. Postal addresses the user saves also are not encrypted.

If an adversary could gain access to the SQLite database on an Android device, then they could read this unencrypted data. They should also be able to decrypt the encrypted data, because the encryption and decryption algorithms the GrubHub Android app uses do not use an encryption key parameter; anyone who knows the encryption scheme can decrypt the ciphertext. The encryption and decryption algorithms are shown in Appendix B.

In order to gain access to this SQLite database other than through the GrubHub app itself, one would need to root the device so that other applications could access the database. This is probably not a high-risk attack, though, since the adversary must have physical access to an Android device whose owner is a user of the Grubhub mobile application, and the device must be rooted. This is not a likely avenue of attack for an adversary, as the payoff of this scheme is not very high. It is a great deal of work to obtain one user's account information with a low probability of success, and a malicious person is more likely to simply sell the stolen phone.

6. Recommendations

6.1 Avoiding Hard Coded Keys

The risk of exposing private API keys could easily be prevented by using a tool, such as ProGuard [8], which obfuscates string names in order not to expose the API credentials. Another solution would be to only store PayPal keys on GrubHub's secure server and implementing a GrubHub API call that in turn calls the PayPal API.

6.2 Better API Authentication Implementation

The current system that Grubhub uses for authenticating users through API calls with permanent tokens appears to follow a claims based architecture (CBA) [9], in which a user is issued a signed access token (the claim) by a trusted authenticator, as a kind of one time sign-on authentication. In the case of Grubhub, the authentication is the regular username and password login, after which the application returns the permanent token or claim which was discussed earlier in section 5.3. Observing the post response returned when a user initially signs up supports the idea that Grubhub is using a form of CBA; the permanent token is called "claim_id", though later references of the same 16 byte string in local storage or in Grubhub's API refer to it as "token".

Overall, using a CBA scheme does not appear to be the most robust method of securing API calls as explained above. The current implementation a permanent token stored in local storage is a single point of failure. We feel that a better method of security would be to use session tokens actually have an expiry time. At least if an adversary obtains an access token, it will eventually expire, and the adversary will need to discover the new token in order to regain access to a user's account.

6.3 Limiting Password Retries

After a certain login rate limit has been surpassed, GrubHub should require login with CAPTCHA, so that to successfully login, human interaction is necessary. This could help prevent adversaries with high computational power from using brute force to break into accounts while still allowing a user who has forgotten their password to give it a few tries.

Login rates could be limited per IP address and per IP address block to limit login attempts over all usernames for an individual client and for a cluster of collaborating clients, respectively. Login rates could also be limited per account username [10].

6.4 Avoiding Storing Insecure Data in Android Device SQLite Database

For the Android app, it is better not to store data permanently in the local SQLite database. Instead, this data should only be stored in the database while the user is logged in. When the user logs in, an API call can be made to the GrubHub server to retrieve necessary data (e.g., saved postal addresses and saved credit card numbers) to store in the database. When the user logs out, the data should be cleared from the database. When a user saves new postal addresses and credit card numbers, this information should be saved on the GrubHub server. Currently, credit card numbers are currently stored only on the Android device and do not appear as saved credit card numbers in the web application.

7. Conclusion

We performed security analysis of GrubHub for its website and the Android app. One major vulnerability that we found was the hard coded PayPal credentials on their Android app. With the credentials, we were able to access GrubHub's corporate PayPal account and obtain information such as account balance, previous transaction details and refund transactions. Through reverse engineering their private web API, we discovered that each user was associated with a long-lived token. If an adversary has access to the token, he could access personal user information and even place and pay for an order on behalf of the user. To take advantage of the other vulnerabilities we found, the adversary would need to have either high computation power or root access to Android devices. Therefore, overall we found GrubHub to be reasonably secure.

8. Acknowledgement

We would like to thank the entire course staff of 6.857, especially Prof. Ron Rivest and Britt Cyr, the TA who advised our project. We would also like to thank the GrubHub security team, especially, Justinian Fortenberry, for giving us permission to do this project.

9. References

[1] Darrell Etherington. *GrubHub publicly files for $100M IPO*. Feb 28, 2014. URL: http://techcrunch.com/2014/02/28/grubhub-publicly-files-for-100m-ipo-saw-1-3b-in-food-sales-in-2013-from-3-4m-active-users/
[2] Tess Stynes. *GrubHub's profit soars, but order growth disappoints.* April 29, 2015. URL: http://www.wsj.com/articles/grubhubs-profit-soars-1430315407
[3] *Compile Android app*. 2015. URL: http://www.decompileandroid.com/
[4] *Owasp Top10*. Aug 26, 2014. URL: https://www.owasp.org/index.php/Top_10_2013-Top_10
[5] Paypal API Credentials. URL: https://developer.paypal.com/docs/classic/api/apiCredentials/

[6] PayPal Express Checkout. 2015. URL:
https://developer.paypal.com/docs/classic/express-checkout/
[7] Mark Pilgrim. *The Past, Present, and Future of Local Storage for Web Applications* URL:
http://diveintohtml5.info/storage.html
[8] Proguard. URL: http://proguard.sourceforge.net/
[9] An Introduction to Claims. URL: https://msdn.microsoft.com/en-us/library/ff359101.aspx
[10] Rate-limiting web application login attempts. URL:
http://timoh6.github.io/2015/05/07/Rate-limiting-web-application-login-attempts.html

Appendix A. API Documentation

Below is a summary of the important endpoints in Grubhub's private API.

If a full url is not specified, the calls are being served from the Grubhub labs API: https://labsapi.grubhub.com/api/v1. Generally, actions where a user is expected to be logged in are served from this API, whereas those not requiring login are served from: https://api-gtm.grubhub.com

A.1 Registration and Authentication

POST https://api-gtm.grubhub.com/credentials Create a new account. Post request body must include an email, first name, last name, and password.

POST https://api-gtm.grubhub.com/auth Log into a user account. Post data must include an email and password.

POST https://api-gtm.grubhub.com/resetPassword Reset account password. Post data must include email.

A.2 Account

The user token is of the form XXXX-XX-XX-XX-XXXXXXX where X is a hexadecimal byte. See section 5.3 for more details.

GET account/[user token] Returns account information for the user associated with this token. This includes the user's first and last name, user ID, permanent account token, email, and a complete list of previous orders.

GET /account/[user token]/creditcards Returns an array of saved credit cards objects, including a credit card ID, the last 4 digits of the card, and its expiration date.

GET /account/[user token]/addresses Returns an array of saved address objects.

POST /account/[user token]/creditcards Submit new credit card information including entire credit card number, expiration date and cvv code to be saved to the account.

POST /account/[user token]/reviews Submit a review for a restaurant. Grubhub only access one review per restaurant that a user has previously ordered from. Not all review comments are posted on the site. Grubhub selectively chooses reviews to post.
Of the format {"restaurantId":[id],"rating":[integer 1-5],"text":[review text]}

A.3 Search

GET https://api-gtm.grubhub.com/restaurants/search/lite/[latitude]/[longitude] Returns search results for cuisines, dishes, and restaurants at the given latitude/longitude.
Query parameters:

- ids: restaurant id numbers, comma-delineated

GET https://api-gtm.grubhub.com/restaurants/search/search_listing See restaurants matching the provided query parameters.
Query parameters:

- queryText: user-entered food query
- orderMethod: delivery, pickup
- locationMode: DELIVERY, PICKUP
- location: POINT([longitude]%20[latitude])
- sorts: default, price, price_desc, restaurant_name, avg_rating, distance, delivery_estimate, relevance, delivery_fee, delivery_minimum
- pageSize: number of restaurants to be shown
- facet: key/value pairs specifying specific cuisines, ratings, price

Required Headers

- Authorization: Bearer [access token id; a new one is created for each session]

A.4 Restaurant Related API

GET /restaurant/[id] Returns information about a particular restaurant, where restaurant ID is an integer. Data includes ratings, delivery fee, address, phone number, and more.

GET /restaurant/[id]/menu Returns menu information for the given restaurant. Each item in the menu has an associated ID.

GET /restaurant/[id]/reviews Returns reviews that were chosen to be displayed on Grubhub. Review information includes review text, date posted, reviewer's first name, reviewer's total orders placed on Grubhub, and a star rating. Accepts additional query parameter, reviewCount, to specify how many reviews to return.

GET /restaurant/[id]/coupons Returns coupons specific to a restaurant. These are usually displayed on the restaurant menu page to show promotions to users.

A.5 Checkout

POST /order/[id] Initialize an order cart. Some of the data required in the post includes restaurant ID, user ID, address, and generation date. Returned data includes an order token and cart ID.

POST /order/[id]/items Add items to an order. Post data must include the cart ID, order token, restaurant ID, and item IDs of dishes from the same restaurant.

POST /order/[id]/discount-codes Apply a discount code to an order. Post data must include a valid discount code string and the order token.

POST /order/[id]/checkout Place an order and specify a payment method. Payment can either be through an entered credit card number, a saved credit card, or through Paypal. In the case of Paypal, the user is redirected to an external Paypal page to enter payment details, and is redirected back to Grubhub upon success or cancellation.

Appendix B. Android App Encryption and Decryption Algorithms

Below are the algorithms used in the Android app for encrypting and decrypting credit card numbers and PayPal information that is stored locally on the Android device. They can be found in /src/com/grubhub/android/j5/local/Encryptor.java of the decompiled app.

```
public String decrypt(String s)
{
        return
        Joiner.on("").join(Lists.reverse(Lists.newArrayList(Splitter.on("g31h").split(s
        .substring("TOKEN-".length())))));
}

public String encrypt(String s)
{
        return (new
        StringBuilder()).append("TOKEN-").append(Joiner.on("g31h").join(Lists.reverse(L
        ists.newArrayList(Splitter.fixedLength(3).split(s))))).toString();
}
```

Appendix C. Paypal Express Checkout API

The following curl calls were used to interface with the Paypal Express Checkout API. In order to perform these calls, one must also have a valid username, password, and signature issued by Paypal.

Search for previous transactions. Accepts additional filters, such as payer username and end date.
curl https://api-3t.paypal.com/nvp -d
"USER=[credentials]&PWD=[credentials]&SIGNATURE=[credentials]&METHOD=TransactionS earch&VERSION=119&STARTDATE=[URL encoded date]"

Refund a transaction. The transaction status must already be complete.

```
curl https://api-3t.paypal.com/nvp -d
"USER=[credentials]&PWD=[credentials]&SIGNATURE=[credentials]&METHOD=RefundTransa
ction&VERSION=119&TRANSACTIONID=[Transaction ID]&REFUNDTYPE=Full"
```