Security Analysis of Tapicnic, A Food Ordering App

Grace Liang, Mark Sayson, Radu Nesiu, and Siyuan He
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
graceliangg@gmail.com, masayson@gmail.com, radunesiu@yahoo.com, hesiyuan48@gmail.com

***Abstract*—This analysis project evaluates the security of Tapicnic, a mobile food ordering application, according to the security principles of confidentiality, integrity, and availability. By intercepting and analyzing traffic between the Tapicnic iOS application and the Tapicnic server, we discovered vulnerabilities related to outdated infrastructure, information leakage, and weak password policies. We have included recommendations to fix these vulnerabilities, as well as an estimation of their cost and impact.**

I. Introduction

In this analysis project we study the security of Tapicnic [1], a food ordering application for iOS. We demonstrate current vulnerabilities in the system, and suggest adjustments that can be implemented to mitigate these issues and improve the overall security of the system. Tapicnic interacts with payment information as well as personal user information such as emails and home addresses. As such, the system owner can expect some risk in managing this information, and securing the assets in transit and storage is a priority.

Related work in the field has produced a number of articles and blog posts that gave insight into the types of vulnerabilities frequent in similar applications, and how to identify them. We encountered difficulties in finding research papers, however, specific articles on software and mobile security were very fruitful. [3].

We focused on the security of the mobile application and its communications with the server, and identified server vulnerabilities that could be determined by an external third party. We used an Internet proxy from an iOS device to a laptop to analyze both encrypted and decrypted communications between the mobile application and the server and investigate potential security issues.

At the conclusion of the project we compiled a list of discovered security issues and specific modifications that can be made to resolve them. This will assist the system owner in mitigating current vulnerabilities and improving the overall security of their system, and draw attention to high-value improvements that can be made to similar systems. The system owner may also use the results of their efforts to demonstrate their focus on security and grow trust with users as they expand their user base.

II. Analyzed System

Tapicnic is a Vancouver-based iOS application for ordering food from local restaurants. Users can view restaurants by neighborhood, make orders, and view the progress of their orders.

The main components in the system are the iOS mobile application, a REST API server endpoint, and a secondary administrative server endpoint. The technologies used for implementing the web API and administrative endpoint are PHP, JavaScript, HTML and CSS. The server endpoints are hosted on an Ubuntu server using Nginx for content handling.

The primary stakeholders in the Tapicnic system are customers of restaurants, restaurant owners, and system administrators. Users interact with the mobile application to request information about local restaurants and submit orders to the application server. Tapicnic staff act as a proxy between customers and restaurants and have the ability to manually filter out orders before they reach the restaurant. Customers of restaurants and restaurant staff both interact directly with the application server, and restaurants are also able to provide updates on pending orders, which are forwarded to the customer.

III. Assets, Threats, Risks

For customers, the primary assets at risk are their payment information and any personal information that may be registered with the application. Unauthorized access to payment information may result in an third party being able to commit credit card fraud, causing significant financial damage to customers. Forms of identity theft and fraud may also be possible if a customer's personal information, such as name, phone number and mailing address, are inadvertently revealed.

From the restaurant's perspective, their resources and reputation are assets that may be at risk. By accepting an illegitimate order, the restaurant prepares an order of food for which no corresponding customer exists. This is especially a problem for take-out orders where the customer can choose to pay upon picking up their food. The restaurant will also want to ensure that they can reliably collect payments, and that customers cannot pick up or receive ordered food without paying.

Another possible scenario is that a malicious user may attempt to deny orders on behalf of the restaurant. In the same

way, a malicious user may also attempt to intercept and modify a customer's order. This would be a major inconvenience to the customer, who is expecting a certain food order, and the business who may lose clients.

IV. Related Work

A blog post on 'Medium' outlines how using Wireshark to capture packets to and from ZopNow, a web application for ordering groceries, is sufficient to draft out an illegitimate transaction to the ZopNow server [5]. The author placed a legitimate order and discovered a 'zoppay_reference' variable in packets. Using Postman, he created a duplicate packet, and incremented the value of 'zoppay_reference'. He sent the modified packet, and to his surprise, an order confirmation was returned.

There has been general work on penetration testing of iOS applications that has been published through information security websites such as InfoSec Institute [2] that we will be able to build from for our security analysis, particularly related to application traffic analysis and testing of local data storage.

For instance, guides on re-routing HTTP and SSL iPhone traffic to external devices, analyzing storage of application data and performing runtime analysis of iOS applications are available through the 'Penetration testing of iPhone Applications' series published on the security testing blog SecurityLearn [3]. We will be able to directly apply this to our analysis of Tapicnic's iOS application.

A recurring idea we observed was that it can be helpful to test how apps react to user timeouts. As we saw in the FoodPanda exploit [4], timeouts may sometimes be handled incorrectly on payment pages.

V. Methodology

A. Overview

We tested the security of the Tapicnic iOS application from both the iPhone application's user interface and from directly inspecting, initiating and intercepting network calls between the iPhone application and the Tapicnic server.

The iPhone's proxy settings were configured to forward traffic to another computer [3], which allowed us to use a wider range of network tools for our security analysis.

We used publicly available network tools such as Burp Suite [6] and Wireshark [7] for inspecting, recording and modifying network calls. Browser plugins such as Postman [8] was used to interact with the server independently of the iPhone application.

Once we identified routes used by the iOS application, we modified API parameters to obtain additional food tokens, modify food prices, and query other other users' account and order information. At each step, we verified whether the server was authenticating requests from unauthenticated and unauthorized users.

We attempted to order food without completing payments by manipulating the normal processing flow of network requests between the phone and application server, and evaluated the security of Tapicnic based on the following Confidentiality-Integrity-Availability principles.

B. Confidentiality

1) Inspect traffic between the server and the phone application, listen to incoming and outgoing calls.
2) Inspect network traffic for unencrypted credentials and confidential data.
3) Probe the server to determine whether we can obtain credentials or confidential data without authentication.
4) Inspect application logs for sensitive data (unencrypted credentials or other confidential information).
5) Investigate whether we can impersonate a third-party such as a restaurant to prompt the server to deliver information to us.
6) Investigate whether an authenticated customer can request confidential account information for accounts other than their own.

C. Integrity

1) Initiate calls to the server to see if we can prompt the creation of user accounts or modify restaurant listings and reviews without authentication.
2) Attempt to send invalid data to the server.
3) Attempt to initiate restaurant orders and financial transactions without authentication.
4) Attempt to replay legitimate requests between the phone application and the server.
5) Attempt to intercept and modify legitimate network calls between the phone application and the server.
6) Attempt to cancel legitimate restaurant orders and financial transactions, without authentication and while authenticated as another user who should not have access to those orders.
7) Attempt to modify restaurant or customer information as an authenticated customer who should not have access to that account.

D. Availability

1) Investigate whether we can lock out legitimate users by impersonating them or by sending invalid credentials under their user names

E. Ethical Considerations

The analysis done by the team follows the three ethical principles. Society as a whole will benefit from this analysis since it sheds light on potential issues with storing and accessing a customer's personal information on the Tapicnic system. As we conclude our system analysis we will prepare a set of concrete recommendations for the system owner, which will assist them in improving their system and increase protection of user information. We will give the system owner enough time to make enhancements before releasing our findings to the public. This balances fairness to both customers and the system owner, as well as encourage increased security for all parties involved. During the analysis project, we will not share our findings with any other persons.

Since one of the goals of this analysis is to determine security flaws in the Tapicnic system with minimal impact to live users, our team will focus on analysis methods that demonstrate the existence of vulnerabilities without disrupting active service. If we find that a system component uses protocols or library versions that are associated with a number of well-known vulnerabilities, we will study the potential risk as well as known countermeasures and report our findings to the system owner without applying the exploits.

We will also only attempt to access information from user accounts that we have created specifically for the system analysis. That is, if our goal is to demonstrate the difficulty of determining a given user's password, we will only use accounts belonging to our team members for the demonstration. This way, we maintain fairness to the customers as no one is unfairly targeted and no vulnerable information is accidentally revealed.

F. Risk Management

Our project team was authorized by the system owner to perform our outlined analysis of the Tapicnic system. As we followed the scope of work agreed to in our authorization form, our project team should be protected from legal risks. We also received permission to share our findings with teaching staff and conference participants who have signed a non-disclosure agreement. After six months have passed from the time that our findings have been reported to the system owner, we are authorized to share our findings with the public.

In terms of analysis completion, we managed some risks regarding available resources. Tapicnic advertises both iOS and Android mobile applications for their customers. We initially planned to test the Android-based mobile application, since most of our team members are non-iOS users and we anticipated that there would be more security testing tools readily available for Android. However, the Android application is no longer offered, so we shared two iPads between our members and spent more time investigating testing methods accessible from iOS.

By proxying Internet communications from an iPad to another IP/port address, we were able to read and modify both unencrypted and encrypted traffic on both Linux and Windows computers, allowing us to analyze application messages and APIs while using the iOS application.

VI. Results

1) All in-application requests and responses are encrypted using HTTPS
2) Database queries are protected from SQL injections through the use of structured parameterized queries
3) Tapicnic API responses include the server's installed versions of PHP and Nginx.
4) The server reports that it is using PHP 5.5.9 and Nginx 1.4.6, both of which are out-of-date
5) Session tokens are stored in request URLs
6) Session tokens do not expire until the next login
7) Authenticated users can view orders of other users
8) Restaurant map responses include the SQL query used
9) User Search collects "LIKE" matches on first name, last name and email
10) There is no limit on the number of login attempts
11) There is no delay between login attempts
12) Password reset codes are five characters long, and only contain numerical and uppercase alphabetical characters
13) New password reset codes do not invalidate previous reset codes
14) Password reset codes accumulate
15) Password reset codes do not expire

VII. Discussion of the Results

A. Interpretation of the Results

The application uses HTTPS for all requests and responses between the mobile application and server, which protects the confidentiality and integrity of messages in transit.

The server reports that it is using PHP 5.5.9. The PHP 5.5 branch is no longer supported and passed its end-of-life date in mid-2016 [9]. This means that PHP 5.5.9 will no longer receive security updates, which is significant because there are many medium-to-high-risk vulnerabilities reported under categories including code execution, overflows and denial of service [10].

Tapicnic embeds session tokens in request URLs, which is generally discouraged as URLs are often easier for third parties to access than request body contents [11] [12].

While the application's use of HTTPS means that the URL will be encrypted in transit, URLs may be logged and stored on multiple endpoints including on-phone logs, server application logs, and server access logs. Server access logs are often unencrypted, which means that if any event leads to a third party gaining access to the server logs, they will have access to all recent request URLs and session tokens. Because session tokens do not expire until the next successful login, access to server access logs currently implies access to active session tokens for every user.

Any party with an active session token can request any order by its order ID, and order IDs are consecutive integer values. This allows users to iterate through order IDs to request each and every order. Order responses contain delivery addresses and phone numbers in addition to order breakdowns and prices, which may facilitate social engineering attempts.

API responses for restaurant metadata include the Restaurant table's SQL SELECT statement, which isn't needed by the client and leaks information on the internal structure of the database.

Within the Tapicnic application, a User Search feature allows users to look up other people to "Follow" one another. We found that the User Search performs a LIKE search on users' first names, last names, and emails, even though other users' emails are not displayed anywhere in API responses or in the user interface.

Searching on partial email strings allows users to collect the email addresses of all other users in a way that links the emails to people. Moreover, because email addresses are used

as user names during login authentication, email scraping can assist the first step for password attacks.

There is no limit on the number of login attempts and no delay between login attempts. This enables malicious entities to brute force the user password. Commonly there is a delay or even lock-out period to combat this. What we notice is that the application performs a hash on the password, client side. However, it is weak and thus it does not provide enough of a delay.

Password reset codes are five characters long, and only contain numerical and uppercase alphabetical characters. The combination of keyspace and length for the password reset codes may have been sufficient a few years ago, but due to improvements in hardware, we now need to question these assumptions.

Generating new password reset codes do not invalidate previous ones. No matter how difficult the password reset codes are to brute force, anyone with a user's email can simply create more and more password reset codes. This is conceptually similar to a meet-in-the-middle attack, where generating password reset codes and guessing password reset codes can be performed simultaneously until a code is correctly guessed.

If a password reset code was discovered through sniffing traffic, leaking information, or other means, the code should expire. This prevents adversaries from permanently having the ability to change a user's password once they've identified a code. In addition, expiring password reset codes also reduce the threat from accumulating password codes. Since codes will now expire, it is more difficult (if not impossible, depending on expiration policy) to generate a large number of codes before any expire.

B. Adversary Model

1) Objectives:

1) Obtain private ordering information.
2) Obtain access to user accounts.

2) Initial Capabilities:

1) Knowledge of system server architecture and version and its vulnerabilities.
2) Access to client version of Tapicnic (able to register as a new client, access to log in, user search, order viewing functionalities).

3) Capabilities During the Attack:

1) Modifying, repeating, removing, HTTPS requests.
2) Infering User emails from server HTTPS responses to crafted HTTPS requests
3) Sending obtained emails to server to generate accumulative reset codes.

C. Principles of Designing Secure Systems

1) One user, in effect, can access all order details. (Least privilege)
2) User search implicitly allows searching based on emails. (Economy of mechanism and Least common mechanisms)
3) Reset code accumulation. (Complete Mediation)
4) Reset code won't expire. (Fail-Safe Defaults)
5) Reset code too short. (Question Assumptions)

VIII. Recommendations

1) Updating PHP and Nginx on the application server. Keeping software up to date loosely follows the principle of question assumptions by addressing the hidden assumption that vulnerabilities in third-party dependencies will not accumulate or impact the application. Updating PHP and Nginx will immediately remove a number of known vulnerabilities. It is cost-effective since upgrading these packages is simple. However, this should only be done after checking for dependencies or plug-ins specific to earlier versions of PHP. The cost is highly dependent on the project's use of PHP libraries, however, we estimate it is a low-to-medium-cost change. Also, PHP 7 has several performance improvements, which could slightly enhance usability of the application.
2) Removing PHP and Nginx software versions from API response headers.
   This leads from the high-level goal of restricting access to that which is "need-to-know". The underlying software versions should not be transparent to users, since users do not need to know them. Revealing software versions only makes it easier for third parties to look up vulnerabilities. This update follows the least privilege principle and would potentially elimiate threats of being exploited by known vulnerabilities. Additionally, it reduces the amount of traffic per reponse. The cost and implementation on the change is trivial and user experience will not be affected.
3) Storing session tokens in API request bodies instead of in URLs.
   Even though the HTTPS Requests are not exposed to users, they are stored in the server logs database in plaintext form. As such, one should always question if a malicious admin or developer can use them to find private information about a user. Therefore, this change follows the question and assumptions principle. The change will prevent potential internal attacks. Although a number of locations in code may need to be updated to construct or read requests, the changes are straightforward and no new library or tools are required. User experience is not affected.
4) Invalidating session tokens after users log out.
   A session token should only be valid for a single session. After a user logs out, instead of permanently allowing access until a new token is generated, the default action is to deny access. Therefore, the change is closely aligned with the fail-safe default principle. This change will reduce the vulnerabilities of session hijacking attacks since the token will eventually expire. Currently, no communication is made with server when users log out. One possible implementation is to send HTTPS request when user clicks log out. The change is

not disruptive of the normal flow, simply it requires one more check on the server to be performed and normal users would not feel the change.

5) Updating the user search API to search by first and last names only.
   Obvously, the recommendation follows the economy of mechanism principle, but we also argue that least common mechanism is also followed. Given that user search implicitly allows search by emails, the emails can be used as the stepping stones for login attacks. That is, the server should not share the emails since they are the crucial part of user login information. With this change implemented, estimated by a removing of few lines of code, email reconstruction attacks would be mitigated. For normal users, they don't even know this functionaliy exists. Hence, we assume no impact on usabiliity.
6) Improving confidentialy of food orders
   We recommend associating UserID with orders so that each user can only view his/her own orders. This follows the least privilege principle. At every interaction with the server database, the server should be checking whether the active user has the privilege to access the requested information. This update is crucial because many private information could be leaked with an attacker capable of travesing all orders in the server.
7) Introducing a password policy on account creation.
   We recommend implementing client-side password creation guide to explain the password length and strength requirements. Our recommendation is to require a password to be at least 8 characters in length. In addition, it must contain at least one number, symbol, and upper and lower case character. This needs to be validated on the server side as well.
   This update follows the principle of open design, psychological acceptability and question assumptions. The requirements on the user password should make the app both more usable and more secure. Usability is enchanced by asking users to come up with passphrases. This change significantly decreases the password guessing and brute-forcing attacks. Regardless to say, this policy is easy to implement with no effects on use cases.
8) Improving login security
   We recommend a minimum delay on failed login attempt or a lock-out period after a number of login attempts.
   A simple approach to mitigating brute-force attacks is to enforce a minimum delay per attempt. For example, the server could check the timestamp of the last login attempt for the user, and wait until three seconds have elapsed before processing the next login request. Or the server could exponentially increase the time delay for a series of failed login attempts in a less than a second. On the other hand, a lock-out period could be imposed given that the maximum number of attempts has been reached within some period of time.
   The recommendation follows fail-safe default, complete mediation, and questions assumptions principle. If the recommendation is implemented, every attempt to log in would be treated differently by the server. If multiple attempts are unsuccessful one can consider that information has been leaked (malicious user confirmed 5 passwords that do not work) and mechanism should default to a lock. In addition, one should always doubt those who need many attempts to log in.
   Combined with introducing a delay and increasing key space from the password policy, this will mitigate the brute force attack vector. If exponential delay function is used, then it can be proved that an attacker won't be able to brute force any account (assuming average scnerio). There would be some additional software handling for each user. Particularly, the server must remember the frequency of login attempts. However, these changes are cost-effective though users with volatile memory may find slow login and lock-out period annoying.
9) Improving password reset code policy.
   We recommend increasing the number of characters used in password reset codes, invalidating previous password reset codes and placing a lifetime on the password reset codes.
   First of all, password reset codes should be more difficult to guess in order to discourage third parties from targeting them. We recommend 8-character password reset codes that use a wider range of possible characters. Second, password reset codes should be invalidated as soon as they are used, or after a new password reset code is generated. Third, the reset codes should expire after a certain period of time if they are unused. This disallows adversaries from using it to gain unauthorized access to an account.
   The recommendation follows the question assumptions and fail-safe defaults principles. Tapicnic was first released a few years ago and its reset code length may have been lengthy enough at the time. As computers get faster and the capabilities of adversaries improving, whether a password reset code of this length is still effective should be questioned. Also, reset codes should remain effective for the smallest window of time that still allows users enough time to reset their passwords. Ultimately, a reset code would be better prematurely expiring rather than never expiring. It is also interesting to note that reset codes accumulation is completely opposite to the principle of separation of duty because the system grants access to an attacker if only one of the reset code matches.
   By increasing the length of reset codes to 8 characters, entropy is increased by 14 bits which imposes significantly more time for Trudy to brute force. By having an expiring reset code, repeating password resetting of a given user account is meaningless. These changes would achieve both saving the memory for the server and making the user accounts more secure. These properties of reset codes are already implemented in numerous modern software systems. Overall, the usability of the

system is remain unchanged since for users, they only need to type three more characters to reset their password.

IX. Benefits of Analysis

Our analysis will assist in boosting the overall robustness of the company's services by identifying security vulnerabilities and possible methods of patching them, which will directly help to protect affiliated restaurants and users from financial harm.

Once any vulnerabilities we discover are fixed, this will reduce the risk of illegitimate orders being accepted, and maintain the confidentiality and integrity of legitimate orders and payments to restaurants. Our work will ensure that a third party cannot easily obtain a customer's order (and money) without delivering the requested food. Our work will also confirm that feeding invalid data to the application or the restaurant will not take down the entire service. This ensures that the service can remain available at all times of day to customers.

X. Conclusion

Tapicnic provides the opportunity for customers to remotely order food and pay either up-front or upon receiving their order, and as such it may be viewed as both a business opportunity and a potential risk for participating restaurants. Restaurants must be able to trust that the application will be able to complete orders and receive payments without interruption of service. In turn, customers must have confidence that their payment information will be securely transmitted and their orders reliably delivered to the restaurant.

Our security analysis primarily tested the customer-facing iOS Tapicnic application. By inspecting network calls between the application and the server, we were able to mimic legitimate communications and probe the server without the use of a real iOS device. This broadened our analytic capabilities, allowing us to analyze the system's effectiveness of handling SQL injections, receiving garbage data, and more. In addition, we probed the server for unintended access points by unauthorized third parties. From there, we investigated the server endpoints to see whether we could access sensitive customer information. As a result, we discovered multiple vulnerabilities involving information leakage, outdated infrastructure, and weak password policies.

For each vulnerability discovered, we discussed the corresponding violation of security design principles, and have made recommendations to remedy the issue. For each recommendation, we provide an estimate of its impact, feasibility, cost and affect on user experience.

Security incidents that impact the confidentiality of payment information or the integrity of orders, payments, or restaurant data may cause financial harm to restaurants and their customers, and effect their willingness to use the application. It was valuable to perform security assessments to discover vulnerabilities in the system and provide feedback to the developers. We hope our findings will both assist them in developing a more robust and reasonably secure application, and protect them and their customers against attacks in the future.

References

[1] Tapicnic. Available: https://tapicnic.com.
[2] *InfoSec Institute*. Available: http://resources.infosecinstitute.com.
[3] "Penetration Testing of iPhone Applications - Part 1". *SecurityLearn*. Available: http://www.securitylearn.net/2012/02/12/penetration-testing-of-iphone-applications-part-1 (Feb. 12, 2012).
[4] "Getting free food with FoodPanda". *Appknox*. Available: https://blog.appknox.com/after-ola-foodpanda-is-the-target-of-a-new-hack-to-get-free-food.
[5] "Making businesses grow without compromising security". *Medium*. Available: https://medium.com/@_whitepearl_/is-the-competition-worth-if-it-s-not-secure-e490f23c3e10#.2vxfeh742
[6] "Burp Suite". *PortSwigger*. Available: https://portswigger.net/burp.
[7] *Wireshark*. Available: https://www.wireshark.org.
[8] *Postman*. Available: https://www.getpostman.com.
[9] "PHP: Supported Versions". *The PHP Group*. Available: https://secure.php.net/supported-versions.php.
[10] "PHP PHP 5.5.9: Related Security Vulnerabilities". *CVE*. Available: https://www.cvedetails.com/version/164957/PHP-PHP-5.5.9.html.
[11] "Session token in URL". *PortSwigger*. Available: https://portswigger.net/KnowledgeBase/Issues/Details/00500700_SessiontokeninURL.
[12] "noVNC contains the session token in URL and insecurely sets the session cookie". *Red Hat*. Available: https://access.redhat.com/solutions/1331003.

Appendix A
Project Code of Conduct

A. All of society benefits

Society benefits from having a security analysis done on a modern mobile food ordering system. For the users, our analysis highlights the potential risks of using such a system. In addition, we provide the system owner with a list of improvements to be made to increase the system's security and draw attention to how similar systems can be protected.

B. People are treated as an end

Since our goal is to find security flaws without exposing real customer data, our team was careful to only access user information related to accounts that we created specifically for security testing. This ensures that we do not access nor exploit customer data in our analysis.

C. Fairness to all involved parties

Following the principles of responsible disclosure, all involved parties will be treated fairly. The system owner will be given an adequate amount of time to make all suggested enhancements, with six months of time to address reported issues without external disclosure, while eventual disclosure will raise security awareness and encourage issues to be fixed, protecting system users.

D. Honors property rights

Our team gave due credit to tools we used, respected the licenses of our tools, as well as usage rights.

E. Respects other individuals rights to privacy

Following this principle our team was careful not to expose any information of the system users and instead created testing accounts for both testing and demo purposes.

F. Honors confidentiality

Since our goal is to help the end users and system owner we honor our NDA and did not discuss project findings with third parties. We will be responsible in how we disclose information in order to prevent potential damages to any party involved.

APPENDIX B
RESPONSIBLE DISCLOSURE

A team representative met with the project owner in-person and discussed vulnerabilities we discovered as well as answered questions. We will also send an email summary outlining our analysis results and recommendations. This will ensure that all of our findings are accounted for and accessible in written form.

Most likely, after the December 2017, our analysis results will be released from non-disclosure and will be allowed to be published in the public domain.

Owner contact information:

- Name: David Chong
- E-mail address: david@codeboxdev.com
- Phone number: 1.604.418.7305

The meeting is set to 20th December.
Location: Codebox Development Office, 602 - 510 W. Hastings St.
Time: 3 PM to 5 PM