December 12th, 2011 by ahoog

**Forensic security analysis of Google Wallet**

As the first real payment system leveraging NFC on Android, Google Wallet is a very intriguing new consumer app for smartphones. But as we always ask: is Google Wallet secure?

This analysis provides a high-level review of the data transmitted and stored by Google Wallet, with the goal of determining if any sensitive data is at risk. Security vulnerabilities in Google Wallet, if they exist, could place the consumer at risk for financial or identity theft. The testing performed here is the basis for our [appWatchdog review of Google Wallet](http://viaforensics.com/appwatchdog/google-wallet-android.html).

I started the analysis on the initial release, however several updates were pushed out – and one security issue I had uncovered was addressed by an update.  Ultimately I froze my analysis to Version: 1.0-R33v6 of Google Wallet.

Before I dive into the details, if you find our high level security analysis of Google Wallet interesting, you might find these related services and studies equally interesting:

1. [appSecure](http://viaforensics.com/services/security/mobile-security-audit/), viaForensics’ mobile app security service
2. [appWatchdog](http://viaforensics.com/appwatchdog/), our free mobile security and privacy service
3. [Mobile App Security Study](http://viaforensics.com/education/white-papers/appwatchdog-findings-mobile-app-security-iphone-android/), a free report on 100 popular mobile apps
4. [Mobile Security Risk Study](http://viaforensics.com/education/white-papers/appwatchdog-findings-mobile-app-security-iphone-android/), an 85 page report focusing on iPhone and Android in the Enterprise

**Changelog** (changes made to this blog):

1. **12/12/2011, 0539**: We disclosed our findings to Google on 11/30/2011 and had some good discussions with the involved teams. On 12/09/2011, an updated build of Google Wallet (Version 1.1-R41v8) was provided to us for additional testing. The findings will be updated in the Summary section near the bottom.

**Overview of Google Wallet app**

Google Wallet is a new offering from Google which allows consumers to use their Android device to perform contactless payments at retailers.  It incorporates [Near Field Communication (NFC)](http://en.wikipedia.org/wiki/Near_Field_Communication) technology and in the initial release supports 1 major credit card as well as some gift and loyalty cards.

**Google Wallet functionality**

The Google Wallet application offers significant functionality despite being a very early release.  To best illustrate the functionality, the following list details the various screens available in the Google Wallet app in a hierarchical way:

1. Payment Cards
	* Credit Cards (Citi MasterCard at this time)
	* Google Prepaid Card
		+ Add Funds
		+ Show ID
	* Gift Cards
		+ American Eagle Outfitters, Bloomingdale’s, The Container Store and Macy’s at this time
2. Loyalty Cards
	* Champs Sports VIP, Foot Locker VIP, Footaction USA Star Club, Guess List, Kids Foot Locker VIP, Lady Foot Locker VIP, Office Max

**Overview of NFC**

Near Field Communication (“NFC”) is not actually a new technology.  It was developed by Sony and NXP Semiconductors around 2002 with the goal of allowing secure communication between two nodes.  Since 2002, momentum and support for NFC has grown to encompass over 135 countries and hundreds of core technology providers.

NFC works by generating an RF field that should only have a range of a few centimeters which can interact with the other node to exchange data.  While many people think of NFC as sending financial data, this is not a limitation of the standard.  In fact, today several apps exist which allows users to share other data over NFC such as business cards.

Beyond securing the transmission of data, it is also necessary to securely store the financial card information on a device, whether that is a smart card or a mobile phone.  For secure mobile phone storage, a special module referred to as the Secure Element was integrated.  This same embedded technology is found in the implementation of smart cards (contactless payment credit cards, secure id cards, etc.) however in this case the integrated circuit is hardwired onto the mobile phone circuit board.  Access to the data on Secure Element is tightly controlled and there are defensive measures built in which, for example, will permanently erase the data if the passcode is entered incorrectly too many times.  Full testing of these defenses are needed but were not in the scope of this app analysis.

It is worthwhile to quickly compare the security of payments done via Google Wallet/NFC and a traditional credit card in the United States. A credit card may simply have the data encoded on the card or can have a chip in it as well which can provide contactless payments (such as PayPass in the US). In most countries outside of the US, if a credit card has a chip in it, the card is protected with a pin, thus chip and PIN. This means to use the card, you must supply the PIN as well as the card. Unfortunately in the US, it was decided to omit the PIN part so if you have a credit card that provides contactless payment features, it may be possible for someone to scan the card while still in your wallet (although this is non-trivial) and then attempt to use your credit card. The nice thing about Google Wallet is they put the PIN back in (chip and PIN) here in the US. So, if a criminal can get your physical credit card, it will be far easier for them to use than if they get your Android device.

While NFC is only beginning to develop traction in the US, other areas of the world have a far larger implementation.  There have been many reasons for resistance to this technology in the US which are well beyond the scope of this post.  However, anecdotally, there appear to be a growing number of retailers in the US who have added modules to their POS hardware which enable NFC transactions.

**Google Wallet setup**

To setup Google Wallet, you first must have a supported phone and credit card.  While this list is growing, for this review I used a Sprint Nexus S and a CitiBank Mastercard.  The Nexus S had not yet received the update from Sprint with Google Wallet (you can’t download it on the app store) so we manually installed the update.

To configure Google Wallet, you must have a Google Account added to the device.  Since I use many devices for testing, I had not yet setup my Gmail account on the phone.  After doing so, I could proceed with the next steps for Google Wallet setup.  I did not document the exact order of the setup steps so please take the follow overview as a broad representation of the steps involved.

First, you select which account you are going to use:



And I then chose to setup my CitiBank Mastercard:



I already had the Mastercard setup with CitiBank and an email address associated with it.  After completing the information and submitting it (which took sometime), I was presented with a screen that allowed me to have a $100 limit until the account was further verified.  An email was sent to the address registered with the MasterCard which had an activation code I needed to enter:



The first time I did this, it took a little over an hour for me to retrieve the activation code from the email address as it went to another person on the account.  The email stated the activation code would time out in an hour but it still worked even through just over an hour had passed.  I don’t think of this as a huge issue and from a consumer perspective, I was glad I did not have to get a new activation code sent.

With the card fully activated, I could now see my full details and had access to my entire credit line:



**Using Google Wallet**

Using Google Wallet was, surprising, a bit nerve wrecking the first time.  Like any good mobile geek, I was excited to try it out but I didn’t want to hold up a retail line while I fumbled to get something to work.  So, I took a few team members from viaForensics along and we stopped by a local 7-11, where I momentously bought a pack of gum.

After a little fumbling, the transaction went through fine.  Here’s what I did:

1. Cashier rang up my purchase
2. I unlocked my Nexus S
3. I ran Google Wallet
4. I unlocked Google Wallet
5. Selected “Payment Cards”
6. The MasterCard was my default card and was ready to transmit
7. I tapped my phone on the NFC/PayPass reader
8. Done

It really seems like a lot of steps but I’m on my mobile device constantly and as I used it a few more times, it was not an issue. Besides, I would often hear things like the following:

Cashier: Did you just pay with your phone

Me: Yes

Cashier: That’s so cool, you made my day

Yes, that conversation really happened and bodes well for people wanting to have a digital wallet on their mobile phones.

It’s important to note that by default, Google does not enable the “Display Location” setting but for testing, I enabled it.  Later, as we uncover this information it’s important to note it was the user’s choice to do this.  That does not mean it should be easily recoverable but was still a choice I made.

**Attacking Google Wallet**

There are many different ways we could attempt to exploit Google Wallet.  However, in the interest of time, to date I have only done the following:

* Attempt Man In The Middle (MITM) attacks
* Forensically analyze data stored on the device
* Examine system logs

At the end of this post, I highlight some additional tests we will perform in the future as time permits.

**Network attacks against Google Wallet**

While there are many different types of network attack we utilize in a full appSecure audit, for this test we only attempted the Man In The Middle attack over Wi-Fi.  This was attempted at both account registration and when adding a new credit card and the Google Wallet successfully protected against the attack.  Here are some screenshots of how Google Wallet handled a MITM attack.

Screenshot of Google Wallet stopping a MITM attack on account setup



Screenshot of Google Wallet stopping a MITM attack on credit card add



**Forensic analysis of Google Wallet**

After setting up my Google Wallet account and using it a few times, I performed three types of forensic acquisitions of the device:

1. Logical acquisition (does not require escalated privileges)
2. File system acquisition (requires escalated privileges)
3. Physical acquisition of user data partition (requires escalated privileges)

For the logical acquisition, as of this time we have not determined whether Google Wallet exposes any data to Content Providers (which is the mechanism we use to extract logical data) so that did not yield any results.  However, additional analysis is required and this may change in the future.

In the remaining two forensic acquisitions, you’ll note that escalated privileges (root) is needed on the device. The most common way in which escalated privileges are obtained on a device are through exploits discovered in the developer, enthusiast (rooting) and security research communities. These can be used intentionally by the device owner or potentially by attackers in the form of malware and other attacks. Once someone has root privileges, they have nearly full access to the device and can overcome the layered approach to security the device provides. On Android, these layers include isolating the app to their own sandbox which means the app cannot access files, processes or the memory of another app. Of course, as a forensic geek we simply skip over these permissions and access the data directly. However, without escalated privileges on the device the following data would not be accessible.

The file system acquisition did, as expected, produce files and I performed some basic analysis of the data.  The Google Wallet application directory was located at:

/data/data/com.google.android.apps.walletnfcrel

and contained four directories:

1. app\_sslcache
2. cache
3. databases
4. shared\_prefs

**Contents of app\_sslcache directory**

The app\_sslcache directory contained only one file, clients5.google.com.443 which appears to be certificate related.

**Contents of cache directory**

The cache directory contained filenames that were 32 characters long and used hex values.  The name convention was not immediately apparent to me however all of the files were PNG images.  An example of a file name is 00502c2ef25cff75c6aa0f76a98c2f5f.

Most of the files were pictures of blank cards however one of the cards had been updated to reflect some of my credit card information.  While this was an initial and small finding, before I could complete this article Google pushed out an update to Google Wallet and the card image no longer contained the data.  While very little data was exposed, the most problematic was the card expiration date which is a key piece information needed for Card Not Present  (“CNP”) transactions.  Here’s a redacted version of the cache image that was originally stored (and is no longer):



**Contents of databases directory**

The databases directory contained four SQLite database files:

1. google\_analytics.db
2. walletDatastore
3. webview.db
4. webviewCache.db

The webview.db and webviewCache.db databases were empty and did not appear to have deleted data.  These are usually present when an app uses the WebKit browser to render HTML.

Upon initial inspection, the google\_analytics.db does not appear to contain much data.  Only 4 of the 11 tables appear to have any data.  They are:

1. android\_metadata which contains en\_US and is simply locale information
2. sqlite\_sequence which I normally overlook because it is a table controlled by SQLite for tracking auto incrementing sequence numbers.  But, it caught my attention since I had entries for both custom\_var\_cache and hits.  But the hits table was not showing any data while the sequence number was 65.  I suspected data was being written to the table and then deleted (see below).
3. session which had 1 row and 5 columns of data which we can use to better understand some of the network and Google Analytics data:
	1. timestamp\_first: 1317913264
	2. timestamp\_previous: 1317913264
	3. timestamp\_current: 1317914226
	4. visits: 2
	5. store\_id: 1644624826
4. custom\_var\_cache which appears to contain different Google Wallet events some of which are labeled initialProvision, version and fgpsEnabled.

I wanted to better understand what was in the google\_analytics.db so, like any good forensic geek, I took time to view it in a hex editor.  This data was deleted but was easily recoverable from the hits table.  For example, here’s a hex dump from a database showing part of a GET request to the Google Analytics service:

0003110: 3d5f 5f75 746d 6125 3344 312e 3136 3434  =\_\_utma%3D1.1644
0003120: 3632 3438 3236 2e31 3331 3739 3133 3236  624826.131791326
0003130: 342e 3133 3137 3931 3332 3634 2e31 3331  4.1317913264.131
0003140: 3739 0000 02be 0083 7705 2f5f 5f75 746d  79......w./\_\_utm

As you can see, there is a Store ID (row 1-2, 1644624826) and an Event time stamps (rows 2-4) in the ASCII data.

The walletDatastore obviously bears a lot of attention and it is the largest of the databases.  There are 20 tables in it so I won’t list them all out but you have to chuckle at some of them.  For example, there are 2 tables that have the word fake in them (fakeSecureElement and fakeSecureElementLog) and one of the rows in the metadata table is called gmad\_bytes\_are\_fun.

Anyway, one table I focused on was the nfcTapEvent table which contains the fields id, proto, purchase\_time (for example 1317917991629 which is Unix Epoch in milliseconds) and is\_dirty (which is not a flag for adult stores).  Here we can find a lot of data, most notably in the proto field which has some binary data and needs to be further analyzed or decoded.

However, it’s quite easy to find interesting info including the address of the transaction, which card I used, the card expiration date as well as my name. In the data below, comments are in italics and we changed sensitive values.

X3:fc89d233-a55f-4815-9a19-ba9eb121feba
$fc89d233-a55f-4815-9a19-ba9eb121feba
MasterCard-xxx-9999"
Andrew HoogP *My name*
99,999.99 *My credit limit*
2,222.22 *Available balance*
1,111.11 *Payment due*
Oct 9, 20110 *Due date*
&bKhttps://www.accountonline.com/cards/svc/img/mobile/cards/C\_DIV\_W\_Mobile.pngj
9999 *Last 4 digits*
Citi
877-762-2989
http://www.citi.com/advantage2
https://ssl.gstatic.com/commerce/wallet/20110109/jhfae70rio980yhbnsox6vkc9sjkdcuy223hnso08udmnnds8776vp6n5744ghopeewdx/demo\_card\_citi\_small.png8
Citi MasterCardP *Card type*
$a367cb1e-e246-4a39-bd8b-e46f74d4867b
100.00
Email
XXXXXX@SOMEEMAILDOMAIN.net *Email CC was registered with*
$fc89d233-a55f-4815-9a19-ba9eb121feba
3:fc89d233-a55f-4815-9a19-ba9eb121feba
26:fc89d233-a55f-4815-9a19-ba9eb121feba
$fc89d233-a55f-4815-9a19-ba9eb121feba
$fc89d233-a55f-4815-9a19-ba9eb121feba
651 South Blvd *Address of 7-11 where I made my purchase*
Oak Park, IL 60302
Oak Park, Illinois 60302\*
Khttps://www.accountonline.com/cards/svc/img/mobile/cards/C\_DIV\_W\_Mobile.png
9999"01\*152 *Last 4 digits and expiration date*
Andrew Hoog8 *My name*
MasterCard-xxx-9999 *Last 4 digits*
26:fc89d233-a55f-4815-9a19-ba9eb121feba
deviceInfo
$fc89d233-a55f-4815-9a19-ba9eb121feba
@c27e508c6baac1d100a1193300abe7e721c7adc264edfe8b9c68b8613e0debdd
MYGMAILACCOUNT@gmail.com\*^ *My Gmail account*

Another table in walletDatastore that contains quite a bit of data is the credentials table.  From the proto filed, I can see data (see above) such as:

* Card type and last 4
* Card holder name
* Current balance
* Available to spend
* Statement balance
* Payment due date
* Citi contact number

Other interesting information I can find by looking at the data in a hex editor (see above) includes:

* Gmail account used for the Wallet
* Masked email address the Citicard was registered to.  It’s stored in the following format: XXXXXX@youremaildomain.com

**Contents of shared\_prefs directory**

The shared\_prefs directory contains 4 files:

1. com.google.android.c2dm.xml
2. OtaProxy.xml
3. THROTTLER\_SHARED\_PREFERENCES.xml
4. ui\_shared\_preferences\_repository.xml

Initial inspection of the files did not reveal any sensitive data.  It’s unlikely that configuration information can be used to exploit the application but analysis well beyond the superficial is needed and beyond the scope of this post.

**Network analysis**

While executing network tests, I performed a network capture of all packets.  Analysis of the pcap files show a few interesting pieces of information.  Most notably are, again, the Google Analytic URLs which send some limited data about the user in URL string format such as:

GET /\_\_utm.gif?utmwv=4.7ma&utmn=1671855394&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmp=%2FsecureElement%2Fadded&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1319235946.1319320579.10&utmht=1319320887188&utmqt=19980 HTTP/1.1

You might notice this is similar to the data found in the google\_analytics.db and is tracking store\_id, transaction times and such.  They also seem to use these URLs to track various status messages such as card provisioning and in the above example “secureElement added”.

**Reset Google Wallet**

There is an option in Google Wallet to “Reset Google Wallet” under the settings menu:



This is also where I told Google Wallet to use location information and set my timeout to 1 minute. I decided to test the reset function to see how well it worked.



As expected, the provision data was no longer accessible by the app and I had to re-add my credit cards to use Google Wallet again.  However, in a rather common misconception made by developers, simply deleting data from a database (or usually even a file) does not really delete the data.  As such, after I ran the Reset Google Wallet function and then examined the databases, I could still recover all of the card, payment and user information I detailed above.  This is certainly an area where Google needs to improve their functionality.  If, for example, you were going to sell your phone after using Google Wallet, I would suggest you do a complete reset of the device as you cannot rely of the reset function inside Google Wallet to sufficiently remove the data.

**Re-activate Google Wallet**

Re-activation process went without a hitch.  And I was not sure how well Google would handle the $10 prepaid card they provisioned but they did it quite well.  I had chanrged several small items to the card and when it was re-issued, I did not get the full $10 (i.e. a new prepaid card) but in fact it reflected my true remaining balance.  I suspect the accountants kind of demanded this part of the app work as they expect it to.

**System logs**

After running Google Wallet, I did an adb bugreport so I have the full system logs and such.  Since the system logs can be read but other apps, it’s important to make sure sensitive data is not stored there.  While nothing of a serious nature is store in the system logs, more information is logged that I would think is needed in a production payment app.  For example, it appears some of the SOAP data is logged and while most field are removed in the syslog, it does provide some insight and shows my IMEI:

10-06 10:06:30.995 I/SIMPLE\_SOAP\_CLIENT(  401): Sending the following request: <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://citi.com/msm/v1.0/paypass/ota/personalization/service" xmlns:mes="http://citi.com/msm/v1.0/paypass/ota/personalization/service/messages" xmlns:com="http://citi.com/msm/v1.0/paypass/ota/personalization/common"> <soapenv:Header/><soapenv:Body><ser:initiatePersonalizationRequest><mes:requestHeader><com:deviceToken></com:deviceToken><com:IMEI>A000002A292F80</com:IMEI><com:CPLC>01:R5BQREeRAHgzABBBApR0lUiWAAAAAAAAAAAIBCYyOTQ3NA==</com:CPLC></mes:requestHeader><mes:KYC><mes:name>xxx</mes:name><mes:accountNumber>xxx9999</mes:accountNumber><mes:expirationDate>xxx</mes:expirationDate><mes:YOB>xxx</mes:YOB><mes:zipCode>xxx</mes:zipCode></mes:KYC><mes:deviceInfo><com:IMEI>A000002A292F80</com:IMEI></mes:deviceInfo></ser:initiatePersonalizationRequest></soapenv:Body></soapenv:Envelope>

When I get a chance to proxy the full requests, I suspect we’ll see the actual data being sent but I have to verify this.

It’s also pretty obviously that I have a CitiBank MasterCard however I’m not sure this is much of an issue:

10-06 10:07:04.503 W/OTA\_STATUS\_SCANNING\_SERVICE(  401): Checking status on Citi MasterCard

A bit more annoying is seeing all the Google Analytic hits.  Again, I know they track everything but you can learn quite a bit about someone or their transactions by simply reviewing this data:

10-06 10:07:04.917 V/GoogleAnalyticsTracker(  401): Called dispatch
10-06 10:07:04.944 I/GoogleAnalyticsTracker(  401): Host: www.google-analytics.com
10-06 10:07:04.944 I/GoogleAnalyticsTracker(  401): User-Agent: wallet-android-sprint-us\*PRODUCTION/Wallet\*1.0-R27v9\*MiFare\*9 (Linux; U; Android 2.3.7; en-us; Nexus S 4G Build/GWK74)
10-06 10:07:04.944 I/GoogleAnalyticsTracker(  401): GET /\_\_utm.gif?utmwv=4.7ma&utmn=1502428448&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmp=%2Fdashboard&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1317913264.1317913264.1&utmht=1317913449507&utmqt=175439 HTTP/1.1
10-06 10:07:04.952 V/GoogleAnalyticsTracker(  401): Scheduled next dispatch
10-06 10:07:04.952 V/GoogleAnalyticsTracker(  401): Sending 10 hits to dispatcher
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): Host: www.google-analytics.com
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): User-Agent: wallet-android-sprint-us\*PRODUCTION/Wallet\*1.0-R27v9\*MiFare\*9 (Linux; U; Android 2.3.7; en-us; Nexus S 4G Build/GWK74)
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): GET /\_\_utm.gif?utmwv=4.7ma&utmn=395829529&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmp=%2FtokensBrowser%2FpaymentCards&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1317913264.1317913264.1&utmht=1317913450032&utmqt=175175 HTTP/1.1
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): Host: www.google-analytics.com
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): User-Agent: wallet-android-sprint-us\*PRODUCTION/Wallet\*1.0-R27v9\*MiFare\*9 (Linux; U; Android 2.3.7; en-us; Nexus S 4G Build/GWK74)
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): GET /\_\_utm.gif?utmwv=4.7ma&utmn=249947280&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmp=%2FprepaidCardDetails&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1317913264.1317913264.1&utmht=1317913458970&utmqt=166238 HTTP/1.1
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): Host: www.google-analytics.com
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): User-Agent: wallet-android-sprint-us\*PRODUCTION/Wallet\*1.0-R27v9\*MiFare\*9 (Linux; U; Android 2.3.7; en-us; Nexus S 4G Build/GWK74)
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): GET /\_\_utm.gif?utmwv=4.7ma&utmn=1055820794&utmt=event&utme=5(Citi%20PROD\*provision%20started)8(1!initialProvision\*3!fgpsEnabled)9(1!true\*3!true)11(1!3\*3!3)&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1317913264.1317913264.1&utmht=1317913552909&utmqt=72299 HTTP/1.1
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): Host: www.google-analytics.com
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): User-Agent: wallet-android-sprint-us\*PRODUCTION/Wallet\*1.0-R27v9\*MiFare\*9 (Linux; U; Android 2.3.7; en-us; Nexus S 4G Build/GWK74)
10-06 10:07:05.206 I/GoogleAnalyticsTracker(  401): GET /\_\_utm.gif?utmwv=4.7ma&utmn=413589664&utme=8(1!initialProvision)9(1!true)11(1!3)&utmcs=UTF-8&utmsr=480x800&utmul=en-US&utmp=%2Fprovision%2Fstart%2FCiti&utmac=UA-18047900-3&utmcc=\_\_utma%3D1.1644624826.1317913264.1317913264.1317913264.1&utmht=1317913553038&utmqt=72171 HTTP/1.1

On a more positive note, here’s what happens when the MITM attack is launched.  Generally providing stack traces in production is not the best idea (may give the attacker info they need to advance their techniques) but we see so many apps that do not properly validate SSL that I guess I’m just happy to see this:

10-06 10:05:54.487 W/HttpTask(  401): Exception executing HTTP exception for request
10-06 10:05:54.487 W/HttpTask(  401): javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.
10-06 10:05:54.487 W/HttpTask(  401):    at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl.startHandshake(OpenSSLSocketImpl.java:477)
10-06 10:05:54.487 W/HttpTask(  401):    at
SNIP
10-06 10:05:54.487 W/HttpTask(  401): Caused by: java.security.cert.CertificateException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.

followed by:

10-06 10:05:55.038 I/ForegroundProvisioningService(  401): Provisioning failed for MasterCard-xxx-9999
10-06 10:05:55.038 I/ForegroundProvisioningService(  401): Provisioning complete notification
10-06 10:05:55.058 I/ForegroundProvisioningService(  401): Stop foreground provisioning service.
10-06 10:05:55.062 W/CitiKycPresenter(  401): failure!!!

There’s quite a bit more in the system log but, again, it takes a lot of time to go through all of that so this is just a sample.

**Other observations**

While I was testing Google Wallet, I jotted down a few things that came to mind.  These are simply some observations I made and are really mostly notes for further follow up.

* Why does the app display my email/account when initially run?  Is it possibly/likely that one phone will have multiple accounts on it?  If so, will each have its own PIN?  Initial reaction is that the splash screen should probably not disclose my user account (i.e. my Gmail address).
* Will consumers use the same PIN on their main screen as they do in Google Wallet?  Probably.  Is a 4-digit PIN sufficient?  Can we brute force it?



**Further testing**

In a full security audit, far more extensive testing would be done.  Some additional tests that come to mind (and time permitting we’ll do over time) include:

1. Test NFC tap events with USB NFC reader
2. Inspect/decompile the APK
3. Determine how gift card activity is stored
4. Attempt relay attack
5. Proxy the network traffic to determine data protocols
6. Attempt to access the Secure Element data

If you are particularly interested in us doing any of these, please [let us know](https://viaforensics.com/contact-us).

**Summary of Google Wallet security findings**

So, in summary, here are the items of note from my high level analysis.  Bear in mind this is nowhere near the level of testing an app like this deserves but since this is done on our own time, it’s all I could manage thus far.  Anyway, here goes:

1. A fair amount of data is stored in various SQLite databases including credit card balance, limits, expiration date, name on card, transaction dates and locations and more.
2. The name on the card, the expiration date, last 4 card digits and email account are all recoverable
3. [Fixed in Version 1.1-R41v8] When transactions are deleted or Google Wallet is reset, the data is still recoverable.
4. The Google Analytic tracking provides insights into the Google Wallet activity.  While I know Google tracks what I do, it’s a little frustrating to find it scattered everywhere and perhaps in a way that can be intercepted on the wire (non-SSL GET request) or on the phone (logs, databases, etc.)
5. [Fixed in Version 1.0-R33v6] The application created a recoverable image of my credit card which gave away a little more info than needed (name, expiration date and last 4 digits).  While this is not enough to use a card, it’s likely enough to launch a social engineering attack.

While Google Wallet does a decent job securing your full credit cards numbers (it is not insecurely stored and a PIN is needed to access the cards to authorize payments), the amount of data that Google Wallet stores unencrypted on the device is significant (pretty much everything except the first 12 digits of your credit card). Many consumers would not find it acceptable if people knew their credit card balance or limits. Further, the ability to use this data in a social engineering attack against the consumer directly or a provider is pretty high. For example, if I know your name, when you’ve used your card recently, last 4 digits and expiration date, I’m pretty confident I could use the information to my advantage. When you add data that is generally available online (such as someone’s address), an attacker is well armed for a successful social engineer attack.

And this testing was really only very high level. Far more sophisticated and comprehensive security analysis is needed to determine if other vulnerabilities are present.  In addition, privacy conscious consumers so understand that analyzing nearly everything you use Google Wallet for is basically the price you pay for the service. For a tech standpoint, it’s very exciting to see Google Wallet in production. However, it has consistently been viaForensics’ position that the largest security risk from apps using NFC do not stem from the core NFC technology but instead the apps that use the technology. In this case, the amount of unencrypted data store by Google Wallet surpasses what we believe most consumers find acceptable.

**About viaForensics**

[viaForensics](http://viaforensics.com/about/) is an innovative digital forensics and security firm providing services to corporations, law firms and law enforcement/government agencies. We have published several books on mobile forensics and security and have 2 patents pending.

If you found our high level security analysis of Google Wallet interesting, you might find these related services and studies equally interesting:

1. [appSecure](http://viaforensics.com/services/security/mobile-security-audit/), viaForensics’ mobile app security service
2. [appWatchdog](http://viaforensics.com/appwatchdog/), our free mobile security and privacy service
3. [Mobile App Security Study](http://viaforensics.com/education/white-papers/appwatchdog-findings-mobile-app-security-iphone-android/), a free report on 100 popular mobile apps
4. [Mobile Security Risk Study](http://viaforensics.com/education/white-papers/appwatchdog-findings-mobile-app-security-iphone-android/), an 85 page paid report focusing on iPhone and Android in the Enterprise

December 12th, 2011 | Category: [Mobile Security](http://viaforensics.com/category/mobile-security/)