Real Dialog Android Trojan Horse

Jeremy Klein jlk3@WUSTL.EDU and Parker Spielman cpspielman@wustl.edu (A project report written under the guidance of Prof. Raj Jain)

Download 

Abstract

It is no secret that Android (Google's mobile operating system) applications have access to a lot of personal information when granted certain permissions. Apps can read and edit contacts, send and receive texts and phone calls, read your phone number and e-mail account information, track your physical location, and much more. In most cases, these capabilities provide a positive service to the user. However, these permissions can easily be abused to silently collect personal information and act in very malicious ways. Our Android Trojan Horse is a proof of concept to show how some of the permissions commonly granted to Android applications can be used in a malicious manner, without user even knowing. The malicious code in our app can be embedded into any Android application and will run silently behind an otherwise harmless app. In the background, the app scans all of the infected user's sms messages, tracks the user's gps location, and steals the primary email password through a spoofing dialog. In this paper, we explain how the trojan horse works, why it is dangerous, and how users can protect themselves against similar attacks.

Keywords

android virus, android trojan horse, android security, android, security, trojan horse, android virus, sms, android spy, android password

Table Of Contents

1 Introduction

This project shows how easy it is for any Android application to steal users private information.

2 What the App Does

The malicious code embedded in the app allows the app to accomplish several tasks. The app can access all of a user's phone contacts and text messages, which can then be sent to any other source. The app also takes GPS readings of the user's location every 15 minutes, and presents a bogus dialog window, prompting the user to enter their personal email account information. The app also uses a form of social engineering to propagate itself to other devices.

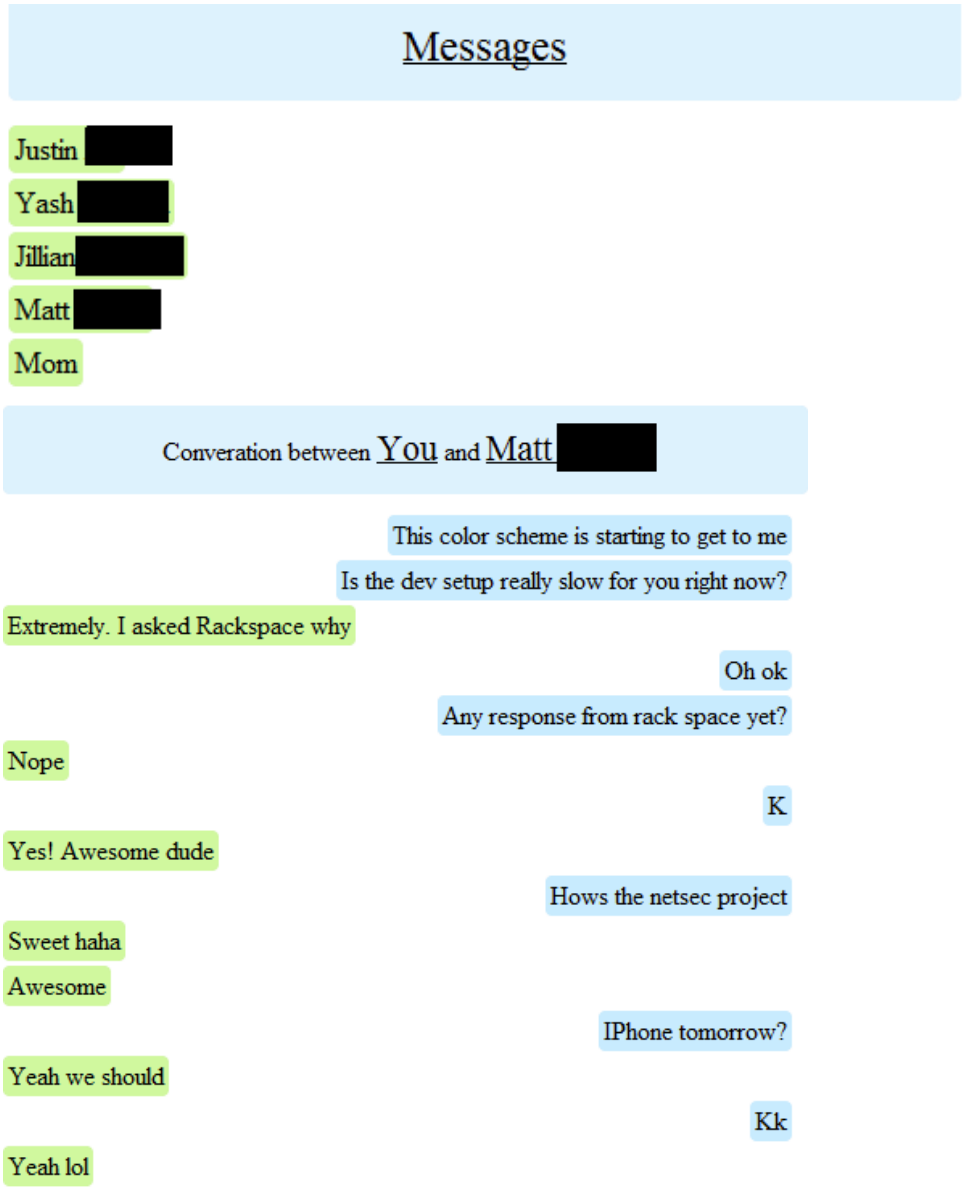
This host application does not have to be running in order for the malicious actions to take place. The malicious code acts entirely in the background, invisible to the user.

2.1 Sms Scanning

Upon installation the app scans all of a user's text messages. Text messages are broken into threads, where each thread contains all of an individual's messages with another contact. The app then cross-references the address book to receive additional information about the other party. The app does this for all text messages that are currently stored on a user's phone. Therefore, in the case of a user who has not deleted his text messages since he started using the phone, his entire messaging history will be instantly accessed and recorded by the malicious app.

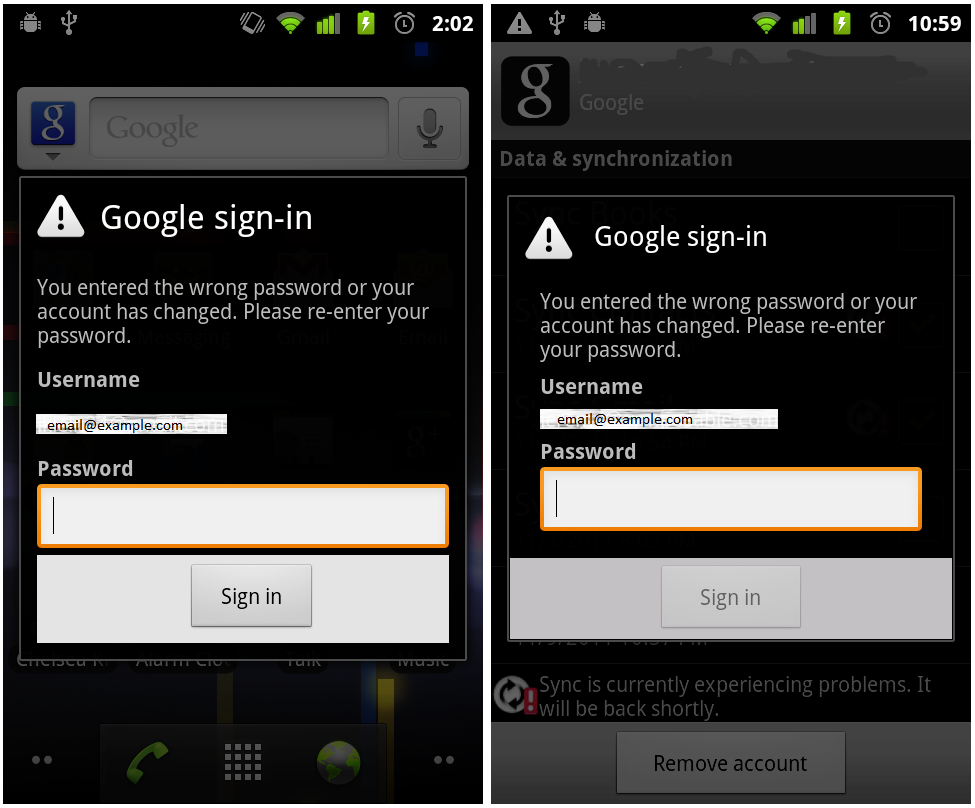
Additionally, after this first run through of all of a user's text messages, for every additional text the user receives, the application will forward the text message content and details to the server designated by the malicious app. The application can easily capture the entirety of the users text messages, from all the text messages initially stored on the phone, and every text thereafter without knowledge of the user, or any interference to normal

The app posts the initial content as a JSON array. This is parsed on the server, and server can feasibly do anything with these messages once they are received. Afterwards, the app continues to post each text message to the server when it detects new messages. In this proof of concept, the text messages are displayed in a manner which mimics the user interface on the actual phone, allowing one to read text messages in chronological order of the conversation per contact.



2.2 Password Theft

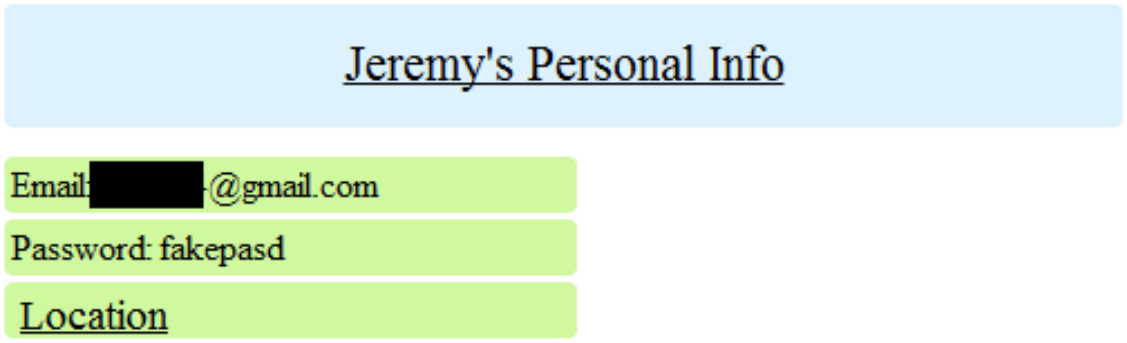
The application also displays an email notification message, which alerts the user of an error with the email account, and asks the user to re-input their email password. The user's email address is already accessible to the application through the GET_ACCOUNTS permission, so it is not necessary to request this information. This dialog is displayed by default 3 minutes after the application is installed, to prevent any correlation between the two events. Ultimately, this time is arbitrary and could be delayed significantly longer to completely isolate the dialogue from the application launched. The dialog shown to the user is an almost exact replica of its real counterpart shown to users when they have entered an incorrect gmail password, or their password has changed. Most users will not think twice when entering their passwords into this fake dialog. Once the user's password is captured once, this dialog is never shown again. Below is a comparison between the malicious and the real email dialogs:



The above images are almost identical, and would be virtually impossible to discern for legitimacy unless compared adjacently with knowledge of the correct dialog screen. Additionally, this type of phishing represents a new stage of fraudulent activity in mobile platforms. As technical security features advance, this area will become less prevalently attacked by hackers due to its large technical and time requirements. However, this type of phishing scheme is a very plausible attack for all sorts of identity theft. Because dialogs are a common feature in banking, PayPal, social networks, and email, this is part of the user's expected behavior on the platform.

These attacks can also be the most difficult to catch ahead of time. They can be easily embedded in the application, and feasibly activated remotely by the hacker.

The app posts the user email and password to the server, and it is subsequently entered into a database and stored for further use. Below is an example display of the content received by the server, and a link to see the GPS tracking information explained in the next session.



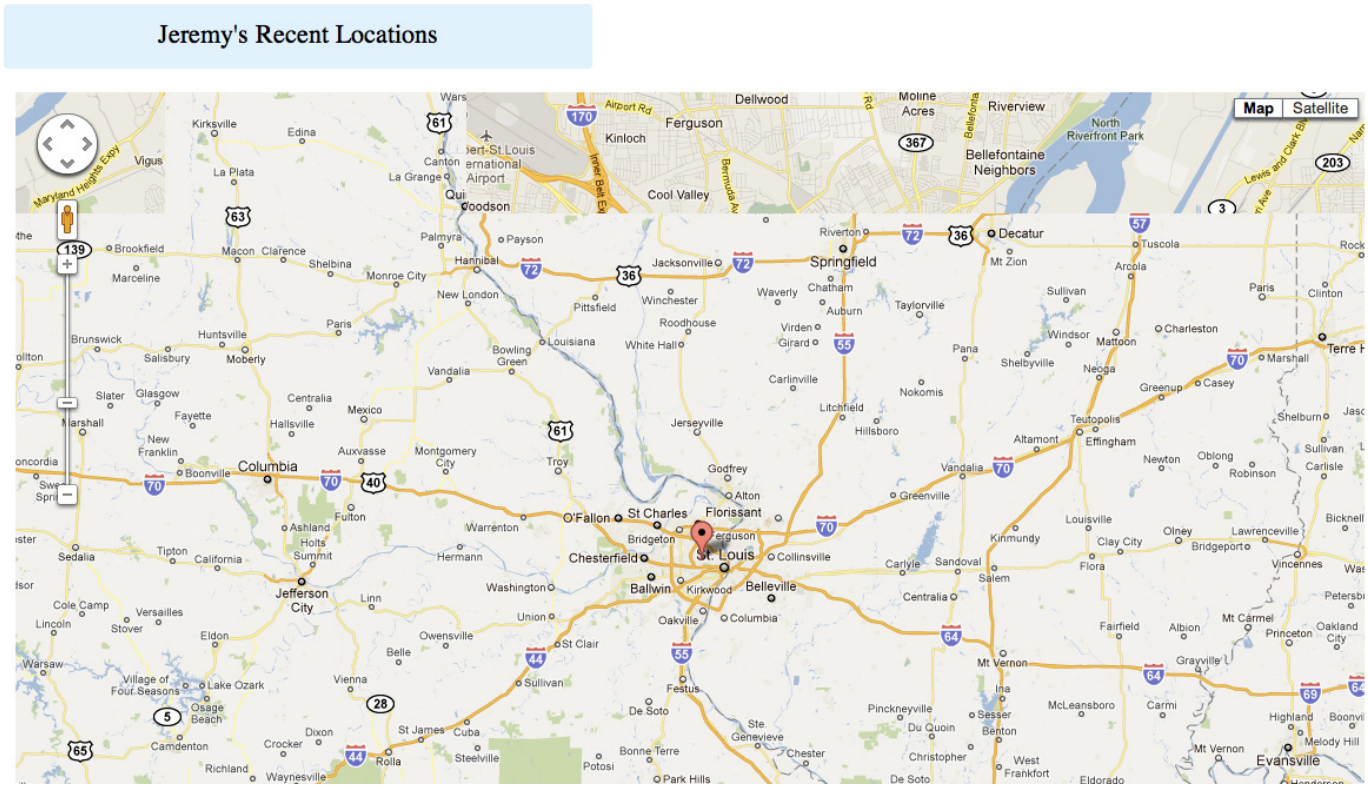
2.3 GPS Tracking

The last targeted attack in this application is geo-location, done through GPS when available, and cell-tower

triangulation when GPS coordinates are not available. Geo-location is an extremely prominent in location based apps, such as Foursquare, Groupon, Google Offers, and many others. As so, this permission is extremely common, and often overlook on app permission dialogs.

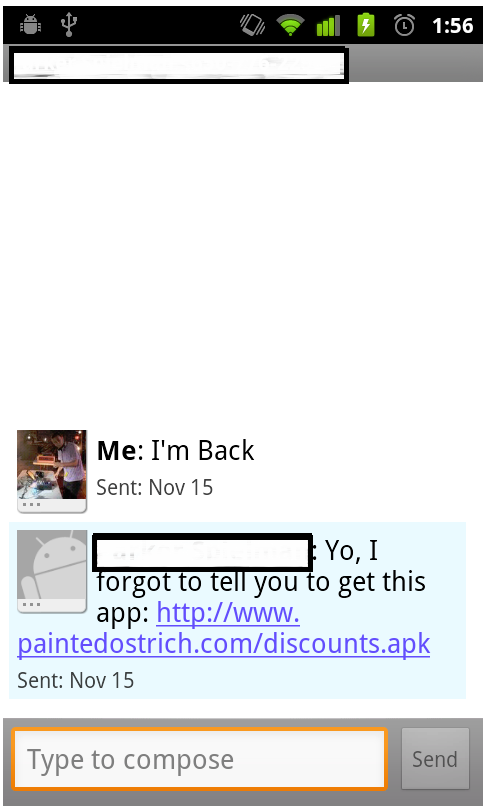
The malicious code in our proof of concept demonstrates that with this permission accepted, an app can pull location based coordinates at whatever frequency it desires. We plot these coordinates on our server, and visually track a user's location throughout the day.

The GPS coordinates are posted to the server as a latitude and longitude point pair. They can then be entered into a database, and easily organized onto a map view like the one shown below, showing an individual location point, multiple recent locations, or a progression of user locations.

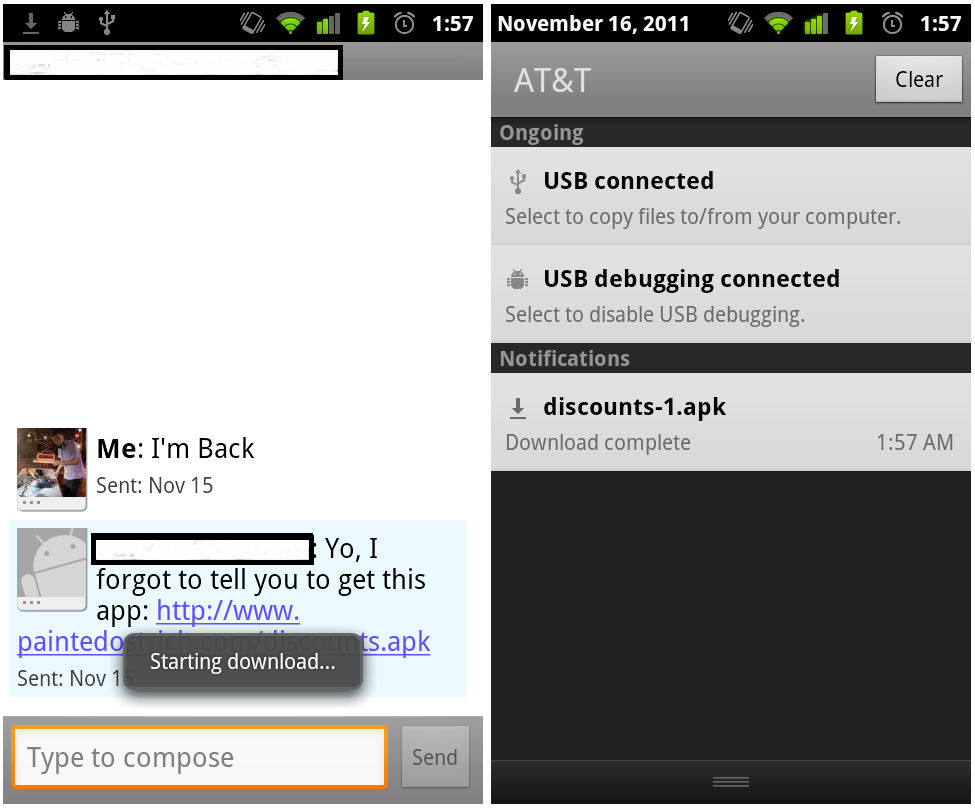


2.4 Propagation

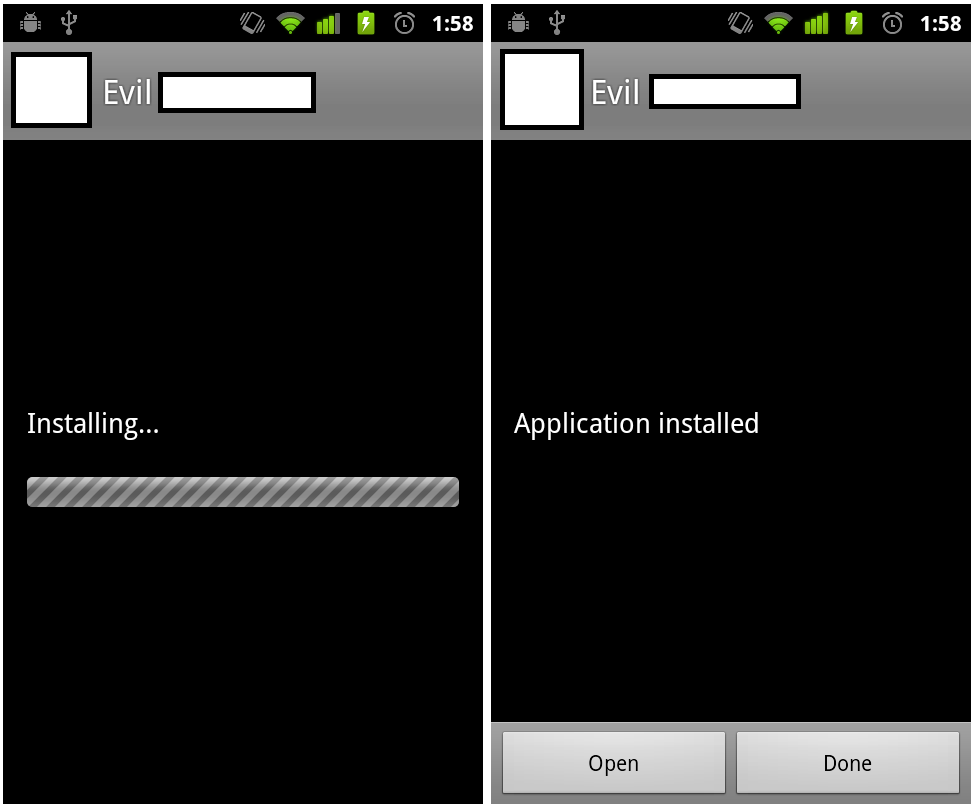
The malicious code in our app propagates itself by relying on inherent trust networks within the user's social life. One minute after a user ends a phone call, their friend receives a text message colloquially saying, "Hey, I forgot to tell you to download this application". See more about relying on trust networks in the Social Engineering section.



The user then has the opportunity to click the link in the text message, which will immediately initiate an FTP download of the application. Note that the user whose phone is infected will never have any record of sending the text.



These screens show the final portion of the short (in the matter of seconds) installation process.



All that remains for the user to do to be completely infected with the virus is open up the application once to begin the timers and the theft of sensitive user information. For any app with the malicious code embedded under a larger useful purpose, all of these steps will be comfortable to the user, and they will end up with another useful app on their phone, while the Trojan operates undetectably.

2.5 Other Feasible Attacks on User Data

There are many additional permissions that can exploited to cause further harm to the user and loss of sensitive information. Some other interesting hacks include remote operation of technical features of the phone (camera/video camera, ringer/vibration), automated reseting of a user's Facebook password through their Gmail account, and smarter social engineering to play off of the context of user interactions among contacts. The exploits demonstrated in this application can also be expanded further. Text messages can be sent on behalf of users without their knowledge, and there are many interesting exploits possible through information about how a user interacts with friends in the context of sovereignty over their phone and email conversations. Additionally, other dialogs can also be created to gain sensitive account information such as banking info. Even if a dialog is suspected as flawed or fake, it will be difficult to trace back to any particular application. In addition, in building this application, we did nothing to conceal the location of the server or our identity as attackers. However, in a more realistic attack, hackers would most likely send data over https and would use some sort of proxy to throw off the trace of where the user's data was being sent. If this virus was deployed as is, it would take almost nothing for authorities to locate the attackers through their server url.

3 Dangers

There are several features which make this attack especially dangerous. Phishing, hacking, and non-rooted control of mobile devices is a relatively new form of attack, and users need to be especially aware of the possibilities of these attacks.

3.1 Silence

Because the phone's content can be silently hijacked, users will never know that their information has been compromised. This is particularly dangerous as it allows the hacker to monitor your behavior and information over time without your knowledge. There will be no trace of the app in any sms history, logs, or notifications. Additionally, with the information gleamed silently from one's user accounts, a hacker may have more than enough information to easily fraudulently use your identity, without notifying you in any way.

3.2 Easy Embedding

Another dangerous aspect of this particular code is that is can very easily be embedded into any non-malicious application. The malicious code was written in a modular manner that does not rely on any code from the host application. This means that the code can exist silently within absolutely any application and will have the same malicious functionality. A developer can easily slip this code into their app to collect private data without affecting the existing functionality of the app and without giving the user any indication of its existence.

3.3 Fast Propagation

This application relies on user networks to propagate itself. Thus being said, elegant social propagation and quick social propagation present a paradox similar to utilizing space or time complexity in computer programming algorithms. Regardless of the actual speed of user propagation (depending on a social engineering or spamming method), the app will still spread exponentially, and be difficult to initially detect as a malicious app. Therefore the impact of this act could be irreversible by the time it is caught and that many users' personal information is irrevocably compromised.

3.4 Social Engineering

Social engineering is perhaps one of the most interesting engineering contexts surrounding this Trojan Horse. It is also the area in which the most interesting and believable exploits can be accomplished by the Trojan Horse. This application depends on the user's social network to propagate itself. It also sends itself only to users who the user is in closer interaction with, in this case determined by phone conversations.

Complex social engineering methods can be used to propagate the app even more wisely, to send messages on behalf of the user to retrieve more personal information, to conduct man in the middle attacks, as well as to hijack the user's social identity, either programmatically or through personal actions. As artificial intelligence continues to become smarter and more adept, the plausibility and extent of these attacks will grow as well.

4 Protection Against

4.1 Read App Permissions

The primary way to protect yourself from apps that silently have access to your personal information is to always read app permissions when installing new apps to your Android phone. As stated in section 1.2 (The Permissions), not all of the permissions granted to this malicious code should be given to most apps. When installing apps, you should always check all permissions and think whether it makes sense for the app to be granted the permissions it desires. There is a constant balance for users when granting apps permissions.

While supplying apps with certain permissions can offer the user better functionality and useful services, these permissions can also be used maliciously as we have done with this Trojan Horse. Users are forced to trust the developer with the use of these permissions and accept vulnerability for added functionality.

4.2 Install from Trusted Sources

It is likely that an app that contained this malicious code would be caught and removed from the Android Market fairly quickly. Google has also, on occasion, remotely wiped malicious apps from all devices that downloaded them. By installing apps only from trusted sources, like the Android Market, you can drastically reduce your vulnerability to attacks like this one. If you were to get a text from a friend like the one used to propagate this Trojan Horse, the best course of action would be to respond to your friend asking what the app was. That would quickly alert the infected user that they are infected and would protect you from becoming infected.

5 Conclusion

It is clear from our creation of this Trojan Horse for the Android operating system that applications running on Android have access to lots of personal data and can fairly easily use that access maliciously. Our app can collect sms messages, e-mail passwords, and GPS location of an infected user, all in complete silence, invisible to the user. We were surprised to see how easy it was for us to propagate the malicious app through sms, utilizing social engineering principles. Due to the Trojan Horse's silence to the user, ability to embed in any non-malicious app, fast propagation, and access to information that enables social engineering, it is extremely dangerous. The frightening part is really that we barely scratched the surface of what can be done with this type of attack. The only ways for Android users to protect themselves from this sort of Trojan Horse is to always read app permissions when installing new apps and to only install apps from trusted sources. If anything strikes you as strange or suspicious about an app, do not install it.

6 Source Code

Download Source Code

Disclaimer:

If released into the public, this code can cause serious harm and can spread very quickly. Do not, under any circumstances, create or distribute an app with this code included. This paper and the corresponding source code are intended for educational and security purposes only. We take no responsibility for any users who might use this code maliciously.

6.1 Files Included

- add to manifest.txt
    - This file includes code snippets that need to be added to the android application manifest of the host application. This includes permissions and service, activity, and receiver declarations.
- CallService.java

    - This is a service that runs in the background and listens for a phone call to end. Once it recognizes that a call has ended, it will set an alarm for 1 minute to trigger the SendSmsReceiver to call its onReceive method.

- GetGPSReceiever.java
    - This receiver is triggered 5 minutes after the first time the application is opened and every 15 minutes silently after that. When it is triggered, it will find the user's GPS location using the open source MyLocation class [1]. It will then send this location to the server and set alarm to trigger itself again in 15 minutes.
- goes in main activity.txt
    - This file includes code snippets that need to go in the activity that is first lauched when a user opens the application. The code in this file sets several alarms for the first time, so that they can set themselves from there on out.
- pass_dialog.xml
    - This file is the layout for the fake password dialog box. It should be placed properly in the layouts directory of your application.
- ScanSMSReceiver.java
    - This receiever is triggered 1 minute after the application is first launched and every 15 minutes after that. When triggered, it scans the users sms messages and sends any messages it has not already sent over to the server. It then sets an alarm to trigger itself again in 15 minutes.
- SendSMSReceiver.java
    - This receiver is triggered 1 minute after any phone call ends. When triggered, it will send a text to the person that the user most recently spoke to, saying "Yo, I forgot to tell you to get this app: [apk url]". This is used to propagate the application and is explained in section 2.4.
- ShowPassDialog.java
    - This is the activity of the fake password dialog. It controls the dialog so that when a user hits "sign-in", the password they entered is sent to the server.
- ShowPassDialogReceiver.java
    - This receiver is triggered 3 minutes after the first launch of the application. If a user enters a password, the receiver is never called again. All the receiver does is start the ShowPassDialog activity.
- Sms.java
    - This class is a basic object representation of an sms message. It also includes a toJson method that converts an sms into a JsonObject representation for transmission to the server.
- SmsHelpers.java
    - This class contains a few basic helper methods for use on sms messages. It helps get all text messages and find a contact's name.

- SmsReceiver.java
    - This receiver is triggered automatically every time an sms is receieved. It will instantly send the receievd sms to the server.

6.2 The Permissions

- INTERNET
    - This permission is needed in order to access the internet from the app. We need this in order to send user data to our server. This is an extremely common permission that is present in most modern apps because many apps need to communicate with some server.
- GET_ACCOUNTS
    - This permission is needed in order for us to get information about the user's accounts. We use this permission to get the user's primary e-mail address, but could also get other account names. It is also a fairly common permission that is used to embed user account information to personalize an application.
- READ_PHONE_STATE
    - This permission is needed in order to see whether the user is currently in a phone call. We use this permission to register a phone state change listener, which can tell us when a user has ended a phone call. It is commonly used by apps to save application state or pause a game when a phone call is received.
- READ_CONTACTS
    - As expected, this permission is needed in order to read the user's contacts. We use this permission in order to match SMS message addresses to the contact who sent/received them. Only social or contact-based applications should be granted this permission. Users should make sure that apps they install with this permission should actually have access to their contacts to ensure privacy.
- ACCESS_COARSE_LOCATION
    - This permission is used to access the user's location via cell towers. It is less accurate than GPS and we use it as a fallback in case GPS location can not be determined. This is a common permission used in apps that need to provide location-based services.
- ACCESS_FINE_LOCATION
    - This permission is used to access the user's location via GPS. We try to use GPS to get the user's location before resorting to less accurate forms of location. This is also a common permission used in apps that need to provide location-based services.
- SEND_SMS
    - This permission is required in order to send sms messages. We use this to send an SMS that will help propagate the trojan horse. Most apps should not have this permission unless they are some sort of sms application or are a social-based app that needs to communicate via sms to a user's

contacts. People should be very wary of installing apps with this permission, unless they know that the app needs to be able to send sms.

- READ_SMS
  - This permission is required in order to read sms messages stored on the phone. We use this permission when scanning the user's sms messages and sending them to the server. Apps should not have this permission unless they serve some sort of purpose having to do with sms. When installing apps with this permission, users should be careful in thinking whether or not the app needs access to this permission.
- RECEIVE_SMS
  - This permission allows applications to receive and handle sms messages when they arrive. This allows us to immediately send any incomming messages to the server as they arrive. Sometimes apps include this permission to save state when a text arrives or to show the user some sort of notification about the text they received. Users should be cautious when installing apps that show this permission, but should also realize that it may be used in a harmless manner.

7 References

This application and its exploits, with the exception of the below open source code incorporated, were designed entirely by the authors of this article: Jeremy Klein and Parker Spielman

[1]The only code referenced that was not originally created by the authors of this paper is the open source (under a GNU GPL v2) MyLocation class from MessesInfo which can be downloaded from: http://www.java2s.com/Open-Source/Android/App/messesinfoandroid/cef/messesinfo/maps/MyLocation.java.htm or http://code.google.com/p/messesinfoandroid/

Last Modified: December 6, 2011