ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 8, August 2015

Exploiting the Vulnerabilities of Android Camera API

Neha K. Malokar[1], Nidhi Subramanian[2], Shriranjani Sriram[3], Sneha Venkat[4], Zainab Khan[5], Seema Shrawne[6]

Student, Department of Computer Engineering & Information Technology, V.J.T.I, Mumbai, India[1,2,3,4,5]

Assistant Professor, Department of Computer Engineering & Information Technology, V.J.T.I, Mumbai, India [6]

Abstract: Most smartphones come with at least one camera, and camera is used mostly with media (photos, videos, etc.) sharing Android applications. Thus, hackers are finding devious ways to exploit the smartphone camera. Smartphones are almost always connected to the Internet. Most applications make use of the mobile data network or Wi-Fi to send and receive data, both in the foreground as well as in the background. In some cases, this sending and receiving of information helps in improving the efficacy of the application by connecting to, say, the social media accounts of the user or sending some user related information, such as location, to the application developer. This can help the developers to cater to the needs of the users in a better way and modify the application accordingly. However, as much as the Internet can be used to send and/or receive useful data, there is a very good chance of the same being exploited by malicious applications to transmit sensitive information which can be an infringement on the user's privacy. Such sensitive information can include personal photos, videos, SMS, etc. This transmission can very well happen without the user knowing about it.One attack that infringes on the user's privacy is a camera-based attack. There is a possibility that an Android Camera can be used to surreptitiously capture photos and/or videos without the user knowing about it. These captured photos and videos can be sent over mobile data networks or Wi-Fi to criminals/ hackers.Here, we attempt to implement the attack on mobile phones, and demonstrate the feasibility and effectiveness of the attack. Furthermore, we propose a defence scheme that can effectively detect these attacks and notify the user.

Keywords: Android Security, Camera API, Camera Attack, SurfaceView, Background service.

I. INTRODUCTION

Android, as we know, is primarily a mobile (Smartphone) operating system. Developed by Google, it is based on the Linux kernel.

Android has rapidly become the fastest-growing mobile OS, thanks to the contributions of the open-source Linux community and more than 300 hardware, software, and carrier partners [1].

Google Play Store (formerly Android Market) is currently the most popular app store for Android applications among commercial developers and the users can easily and quickly download new apps and games, with millions of apps being downloaded daily. The number of available apps in the Google Play Store has been most recently placed at 1.4 million apps in February 2015[2], and this popularity and accessibility of smartphone applications has, naturally, drawn the attention of attackers.

Smartphone malware is easily distributed through an insecure app store. Often malware is hidden in pirated versions of legitimate apps, which are then distributed through third-party app stores. Malware risk also comes from what's known as an "update attack"[3], where a legitimate application is later changed to include a malware component, which users then install when they are notified that the app has been updated.

Such vulnerabilities including the coarse permission system and over-privileging of applications can lead to exploitable applications. Consequently, several efforts have been made to investigate privilege problems in Android apps.

In this paper, we demonstrate ways to hack into the Camera Application which exploit the loopholes found in the visibility options of Background Service and Android SurfaceView and use the flexibility given in specifying the FrameLayout size of the camera screen in order to develop applications that take pictures of users without their knowledge, thus breaching the Android security protocols.

This is done in two ways. In the first method, the malicious application accesses the camera through SurfaceView. It then attempts to hide the camera preview by adjusting the preview size to '1 pixel x 1 pixel' (using FrameLayout), which is so small that it cannot be visible to the naked eye. Thus, a dummy surface view is visible to the user and he/she is completely unaware of the application clicking pictures as he/she does not receive any notifications about any such activity.

In the second method, we are using a base app 'FlashLight' that appears as an application that just simply provides an option switch on and off the flash in our smartphone. This is used to mask the underlying Android code for clicking pictures. The malicious application is made to run in the background (using Background Service) and captures images without the user's knowledge. Thus, the camera preview is automatically hidden, allowing the attacker to exploit the privacy of the user.

These pictures clicked by the application can then be stored temporarily on the user's phone by use of compression techniques (to compress the size of the captured images) and also creating hidden folders to store them. These then can be sent to the attacker's server through unauthorized access of the user's Internet connection.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 8, August 2015

Detection mechanisms involve the use of the getRunningTasks() method, which returns the list of all the applications running in the background. Thus the user can accordingly kill suspicious tasks. It can be extended to display permissions granted to these applications, and options to disable them at the user's discretion.

Another more specific method detects whether an application is using the Camera object at a particular instant of time, thus notifying the user about the same.

For prevention, we have proposed a solution to fix some minimum value for the dimensions of FrameLayout and thus, the camera preview size. For this purpose we show up to what value the human eye can see the FrameLayout, and in turn, prevent hiding of the preview screen. Implementation of this solution is left to the discretion of Android's Developers.

The paper is organized as follows. Section II describes the various aspects of the application. Section III demonstrates the two ways to exploit android camera in detail. Section IV explores some detection and prevention methods. Section V concludes the paper. Section VI lists the references.

II. APPLICATION DESIGN

Figure 1 explains the system design of Spying Application in which, it opens the camera and clicks pictures without user's knowledge. These images can then be sent to the attacker via the Internet using HTTP or FTP protocols, and can be used for various malicious purposes.

Figure 2 explains how the Detection Application works. It runs a code to attempt to obtain the Camera object and throws an Exception if it is found to be busy and further notifies the user about the same.

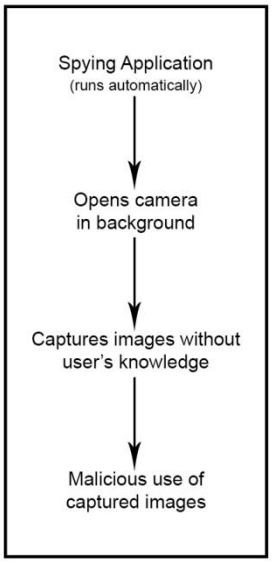

Fig. 1.

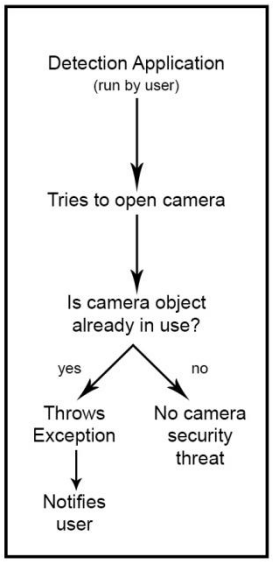

Fig. 2.

Figures 1 & 2 represent Spying and Detection applications

III. WAYS TO EXPLOIT ANDROID CAMERA

Camera use is popular with several applications that encourage photo sharing; hackers are finding sneaky ways to exploit them. Camera technically requires a preview to be displayed on screen in order to take video, but background services do not have associated visible activity. If you attach a preview to the screen from the background service, you can take a photo. Also, by adjusting the preview size to just 1 pixel x 1 pixel, an application can keep the camera running and click pictures without the user's knowledge.

A. 1 pixel x 1 pixel

Key Classes and Methods:

i. *FrameLayout*

Frame Layout is designed to block out an area on the screen to display a single item. Generally, *FrameLayout* should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other [4].

ii. *SurfaceView* and *SurfaceHolder*

SurfaceView have three primary usages: video playback, camera preview and 2D game. *SurfaceView* contains a *SurfaceHolder* which you can pass to *MediaPlayer* or *Camera* as display sink [5].

We have extended the *SurfaceView* class and implemented the *SurfaceHolder.Callback* interface. This class also contains a Camera object. When the surface is created, we set the Camera object's preview in the *SurfaceHolder* of that surface and start the preview. This surface, in turn, is set inside the *FrameLayout* forming a nesting of views

We can resize this *FrameLayout* to a very small size of 1 pixel x 1 pixel. We can then have a normal application running in the rest of the window. One of the buttons in this window should trigger clicking of pictures by the camera or the camera can automatically start clicking pictures using some timer mechanism. Such a small camera preview screen size cannot be seen by the naked eye and hence, users are unaware of the camera clicking pictures, either while they are clicking the button (which they assume is for another purpose) or the camera is just clicking pictures on its own. Such a *FrameLayout* can be embedded in any seemingly normal looking application.

B. Background Service

A service is a component which runs in the background without direct interaction with the user. As the service has no user interface, it is not bound to the lifecycle of an activity.

By default, a service runs in the same process as the main thread of the application. Asynchronous processing is required to perform resource intensive tasks in the background. A spying application could create and run a new *Thread* in the service to perform the processing in the background and then terminate the service once it has finished the processing.

Key Classes and Methods:

i. *PendingIntent* (android.app.PendingIntent)

Instances of this class are created with *getActivity(Context, int, Intent, int)* and *getService(Context, int, Intent, int)*; the returned object can be handed to other applications so that

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 8, August 2015

they can perform some action e.g. recording a video, capturing images, etc. Even if its owning application's process is killed, the *PendingIntent* itself will remain usable from other processes that have been given it [6].

ii. getActivity(Context context, int requestCode, Intent intent, int flags, Bundle options)
Retrieve a *PendingIntent* that will start a new activity, like calling *Context.startActivity(Intent)*.
Parameters:

- context - The Context in which this PendingIntent should start the activity
- requestCode - Private request code for the sender
- intent - Intent of the activity to be launched
- Flags
- options - Additional options for how the Activity should be started. May be null if there are no options

Returns:

- Returns an existing or new *PendingIntent* matching the given parameters

iii. Fragment
A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle [7].
In our implementation, the widget for our spy application is a part of the fragment. The fragment contains the buttons of various modes of capture.
The activity is sent to the background, when we exit our base application. We then attempt that our spy activity automatically captures pictures.

iv. performClick()- This method is called on a view V. It calls the *onClick(View v)* of a class without having the user to actually click the View. In our implementation, this method is used to trigger automatic clicking of pictures by calling *performClick()* on the 'auto' button.

IV. Detection & Prevention

Here, we explore various ways to detect a malicious access to camera by an Android application. Also, we suggest some preventive measures.

A. For 1 pixel x 1 pixel
There can be a limit set for the lowest dimensions any layout can assume. This should be implemented in future versions of Android so that privacy of users is not compromised.
After running the application with 5 users we could come up with an ideal limit for *FrameLayout* dimensions as follows:
Each column contains the dimensions up to which a user was able to immediately observe a running camera in a *FrameLayout* at varying degrees of clarity. We assume a square shaped layout so that the dimensions are same on each side.

TABLE I: FrameLayout Dimensions

User	Very clear	Faint	Not visible clearly
1	10	5	3
2	12	6	3
3	9	5	4
4	11	6	4
5	10	5	3

From the above table, we can see that, it would be better if we can keep a minimum limit of 10 pixel by 10 pixel size for *FrameLayout* in future versions of Android. Dimensions in the next 2 columns will not serve any purpose for genuine Android developers and hence should not be allowed.
The following screenshot shows a camera preview in a 10 by 10 pixel size *FrameLayout*.

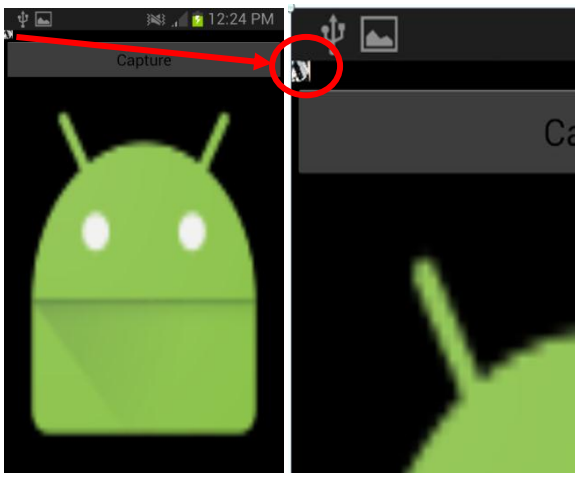

Fig. 3. Frame Layout of size- 10 pixel by 10 pixel

B. For Background Service
i. getRunningTasks(int maxNum)
Return a list of the tasks that are currently running, with the most recent being first and older ones after in order

Parameters:

- maxNum - The maximum number of entries to return in the list. The actual number returned may be smaller, depending on how many tasks the user has started.

Note: As of LOLLIPOP, this method is no longer available to third party applications: the introduction of document-centric recents means it can leak personal information to the caller. For backwards compatibility, it will still return a small subset of its data: at least the caller's own tasks, and possibly some other tasks such as home that are known to not be sensitive [8].

By using the method getRunningTasks(), we can display a *ListView* of all the running applications to the user. The user then gets to know if there is any malicious application running in the background.

ii. Using a detection application
An application should only have one *Camera* object active at a time for a particular hardware camera [9]. We have implemented this in our detection application by using the

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 8, August 2015

method open(), to test whether any application is currently using the camera in the background. If a *Camera* object is already in use by some other application, the method throws an *Exception* and the user is notified. Then he can take appropriate actions.

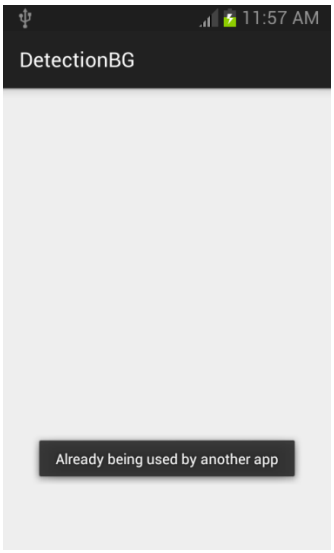

Fig. 4. Detection Application screenshot

V. CONCLUSION

We provide methods to exploit the vulnerabilities in Android camera. We also propose minimum dimensions for the *FrameLayout* used in Android applications, to prevent misuse. Further, we have developed an application that detects whether the *Camera* object is being used by some other application in the background.

Android smartphones, being used everywhere, are subject to a wide variety of attacks through third-party applications. Android phones are almost always connected to the Internet and this makes them all the more vulnerable to attacks on privacy. Also, with the tremendous number of applications available and the permissions to be granted for installing them, most users tend to ignore the kind of permissions they are granting. In future, the developers at Android should consider such vulnerabilities while designing applications and also, users should ensure that they install applications from trusted sources.

REFERENCES

[1] Android's Popularity http://enlightenedapps.com/blog/2014/09/01/android-the-worlds-most-popular-mobile-platform/
[2] Application Statistics http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
[3] Smartphone Malware Risk http://colectivodisenolatinoamerica.blogspot.in/2015/02/smartphone.html
[4] FrameLayout http://developer.android.com/reference/android/widget/FrameLayout.html
[5] SurfaceView http://developer.android.com/reference/android/view/SurfaceView.html http://pierrchen.blogspot.in/2014/03/anroid-graphics-surfaceview-all-you.html
[6] PendingIntent and getActivity() http://developer.android.com/reference/android/app/PendingIntent.html
[7] Fragments http://developer.android.com/guide/components/fragments.html
[8] getRunningTasks http://developer.android.com/reference/android/app/ActivityManager.html
[9] Camera Object http://developer.android.com/reference/android/hardware/Camera.html
[10] Smartphone Security http://www.crucial.com.au/blog/2014/10/21/smartphone-security-hurdles-and-potential-solutions/
[11] Background Service http://www.vogella.com/tutorials/AndroidServices/article.html
[12] Smartphone Camera Spying https://nakedsecurity.sophos.com/2014/05/28/yes-your-smartphone-camera-can-be-used-to-spy-on-you/ http://snacksforyourmind.blogspot.co.uk/2014/05/exploring-limits-of-covert-data.html
[13] Detection of Camera Running http://developer.android.com/guide/topics/media/camera.html

BIOGRAPHIES

Neha K. Malokar will be graduating with a Bachelor's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2015.

Nidhi Subramanian will be graduating with a Bachelor's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2015.

Shriranjani Sriram will be graduating with a Bachelor's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2015.

Sneha Venkat will be graduating with a Bachelor's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2015.

Zainab E. Khan will be graduating with a Bachelor's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2015.

Mrs. Seema Shrawne graduated with a Master's Degree in Engineering in Computer Science from Veermata Jijabai Technological Institute, Mumbai (India) in 2014. She has been an assistant lecturer in the department of computer engineering and information technology department at Veermata Jijabai Technological Institute, Mumbai (India) for the last 15 years. Her areas of interest are Databases, Computer Networks and Information Retrieval.