Analyzing Mobile Phone Vulnerabilities Caused by Camera

Longfei Wu[†], Xiaojiang Du[†], Li Wang[*], Xinwen Fu[‡], Ralph O. Mbouna[§], and Seong G. Kong[§]
[†]Dept. of Computer and Info. Sci, Temple Univ., Philadelphia, PA, USA, {longfei.wu, dux}@temple.edu
[*]School of Electronic Eng, Beijing Univ. of Posts and Telecommunications, Beijing, P.R. China, liwang@bupt.edu.cn
[‡]Dept. of Computer Science, Univ. of Massachusetts Lowell, Lowell, MA, USA, xinwenfu@cs.uml.edu
[§]Dept. of Electrical & Computer Engineering, Temple Univ., Philadelphia, PA, USA, {oyini, skong}@temple.edu

***Abstract*—Nowadays mobile phones have been widely used, and Android is one of the most popular mobile operating system. The security issue of Android has caught great concerns among mobile users and researchers. In this paper, we study the vulnerabilities related of phone cameras. Specifically, we discover and present several camera-based attacks including the basic camera attack and advanced passcode inference attacks. We implement these attacks on real phones (with anti-virus software installed) and demonstrate the feasibility and effectiveness of the attacks. Furthermore, a lightweight defense scheme is proposed to secure phones against these attacks.**

Keywords—Mobile phones; camera; Android

I. INTRODUCTION

Since 2007, Android operating system (OS) has enjoyed an incredible rate of popularity. As of 2013, Android OS holds 79.3% of global smartphone market shares. Meanwhile, a number of Android security and privacy vulnerabilities have been exposed in the past several years. Although the Android permission system gives users an opportunity to check the permission request of an application (app) before installation, few users have knowledge of what all these permissions stand for [1]; as a result, they fail to warn users of security risks. Meanwhile, an increasing number of apps specified to enhance security and protect user privacy have appeared in Android application markets (e.g. anti-virus applications and locker applications). However, mobile malwares and privacy leakage remain a big threat for user security and privacy.

Generally, when talking about privacy protection, most mobile phone users pay attention to the safety of SMS, email, contact list, calling history, location information and private files. They may be surprised that the phone camera could become a traitor, for example, attackers could stealthily take pictures and record videos by using the phone camera. Nowadays, spy camera apps have become quite popular on Android application markets. As for Google Play, there are nearly one hundred spy camera apps, which allow phone users to take pictures or record videos of other people without their permission. However, believe it or not, phone users themselves could also become victims. Attackers can implement spy camera in malicious apps such that the phone camera is launched automatically without the device owner's notice, and the captured photos and videos that contain user's daily activities and conversations are sent out to the remote attackers. Even worse, according to a survey on Android malware analysis [2], camera permission ranks 12th of the most commonly requested permissions among benign apps, while it is out of the top 20 in malwares. The popularity of camera usage in benign apps and relatively scarce usage in malware lower user's alertness to camera-based multimedia attacks.

In this paper, we present the basic attack, and two passcode inference attacks: the application-oriented attack and screen unlocking attack. We run these attacks along with several popular antivirus softwares to test their stealthiness, and conduct experiments on video-based passcode inference attacks. The results demonstrate the feasibility and effectiveness of these attacks. Finally, we propose a lightweight defense scheme.

The rest of the paper is organized as follows. Section II discusses related works. Section III describes the basic camera attack. Section IV presents the video-based passcode inference attacks. Section V describes the implementation and evaluation. Section VI proposes the countermeasure against the camera-based attacks. Section VII concludes the paper.

II. RELATED WORK

A number of recent works have studied the issue of obtaining private information on smartphones using multimedia devices such as microphones and cameras. For example, Soundcomber [3] is a stealthy Trojan that can sense the context of its audible surroundings to target and extract high-value data such as credit card and PIN numbers. Stealthy audio recording is easier to realize since it does not need to hide the camera preview. Xu et al. [4] present a data collection technique using a video camera embedded in Windows phones. Their malware (installed as a Trojan) secretly records video and transmits data using either email or MMS. Windows phones directly offer a function, *ShowWindow(hWnd, SW HIDE)*, which can hide an app window on the phone screen. However, it is much more complicated (no off-the-shelf function) to hide a camera preview window in Android system. In this work, we are able to hide the whole camera app. Moreover, we further utilize computer vision technique to analyze recorded videos and infer passcode from user's eye movement.

Several video based attacks targeted at keystrokes have been proposed. The attacks can obtain user input on touch screen smartphones. Maggi et al. [5] implement an automatic shoulder surfing attack against modern touch-enabled smartphones. The attacker deploys a video camera that can record the target screen while the victim is entering text. Then user input can be reconstructed solely based on the keystroke feedback displayed on the screen. iSpy [6], proposed by Raguram,

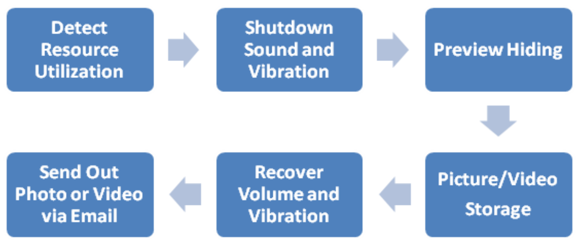

Detect Resource Utilization
Shutdown Sound and Vibration
Preview Hiding
Picture/Video Storage
Recover Volume and Vibration
Send Out Photo or Video via Email

Fig. 1. The Basic Camera Attack Architecture

shows how screen reflections may be used for reconstruction of text typed on a smartphone's virtual keyboard. However, both the attacks require additional camera devices to capture phone screen or its reflections, and issues like how to place the camera near the victim without catching alert must be carefully considered. Besides, they work only when visual feedback such as the magnified keys function or key press confirmation mechanism is available. In contrast, our camera-based attacks work without any support from other devices, or dependence on any specific function.

III. THE BASIC CAMERA ATTACK

We first introduce possible attacks based on spy camera. The attacks should appear normal to user experience. The main challenge is to make the attacks run stealthily and silently so that they do not cause a user alert. Specifically, the attacks are supposed to have a translucent view, make no sound or vibration, and check phone resource utilization before launching themselves. The general architecture should include the following six parts. Figure 1 shows the architecture of a basic spy camera attack.

1) Step 1: To prevent the user from suspecting, the malware should consider the current CPU, memory usage and battery status. Launching the attack when CPU and memory usage are already high could make a phone's performance even worse. Users tend to be concerned about the unsmooth experience, and check if any app or service is running in the background. Similar concern happens with energy consumption, especially when the phone's battery is low and is not being charged. A camera attack could drain the battery faster than the user's expectation, and cause user suspicion about possible attacks. Hence, before launching the attack, malicious camera apps want to ensure that system resources are plentiful. For Android phones, memory usage could be obtained through the *getMemoryInfo()* function of *ActivityManager*, information related to CPU utilization is available from "*/proc/stat*", while current battery level and charging status can be obtained by registering a BroadcastReceiver with *ACTION_BATTERY_CHANGED*.

2) Step 2: After ensuring sufficient resources for launching attacks, a malicious camera app can continue on the remaining actions. First, the app can turn off the phone's sound and vibration, which can be achieved by setting the system sound *AudioManager.STREAM_SYSTEM* to 0 and the flag to *FLAG_REMOVE_SOUND_AND_VIBRATE*. The app can log the current volume level and vibration status, and resume the parameters after the attack.

3) Step 3: The difficult task is to hide the camera preview. At the beginning, the layout containing the *SurfaceView* is inflated into a *view* via *LayoutInflater.inflate()*. Then the app can change the parameters of that view by setting the attributes of *WindowManager.LayoutParams*. Two important attributes must be set: *TYPE_SYSTEM_OVERLAY*, which makes the preview window always stay on top of other apps; the other one is *FLAG_NOT_FOCUSABLE*, which disables the input focus of a spy camera app such that input values would be passed to the first focusable window underneath. This would turn the camera preview into a floating and not focusable layer. Then the app changes the size of preview (*SurfaceView*) to the minimum pixel (1 pixel), which human eyes cannot notice. This cannot be set directly through *setPreviewSize()*. Instead, the app needs to get the layout parameter of *SurfaceView* by using *SurfaceView.getLayoutParams()*. Notice that the type of *SurfaceView.getLayoutParams()* is *ViewGroup.LayoutParams* instead of the aforementioned *WindowManager.LayoutParams*. Finally, the app can add the hidden preview dynamically to the window by the *addView* function.

4) Step 4: After setting up the layout, the attack could be launched as follows: initialize the *SurfaceHolder*, choose which camera (front or back) is used, and open the camera to take pictures or record videos. The photo/video data are supposed to be stored in disguises, including using confusing filenames and seldom visited directories. The app releases the camera after the above actions.

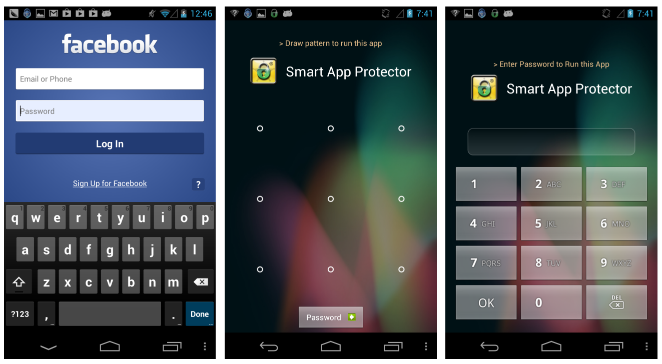
5) Step 5: After the camera attack finishes, the app sets the audio volume and vibration status back to its original values. This way, the device owner would not find any abnormality.

6) Step 6: The last step of the attack is to transmit the collected data to the outside. Since cellular network usage and MMS may cause extra fees, the best choice is to wait until free WiFi access is available. For example, it could use the *javax.mail* to send the data as an email attachment. Most email systems limit the maximum size of attachments, so the length of the captured video should have an upper bound specific to the email service.

IV. THE VIDEO-BASED PASSCODE INFERENCE ATTACK

Since the virtual keyboard in a touch screen smartphone is much smaller than computer keyboards, the virtual keys are very close to each other. Based on the measurement on a Galaxy Nexus 4 phone, even an offset of 5 mm could result in touching the wrong key. Hence, when typing, users tend to keep a short distance to the screen, which allows the phone (front) camera to have a clear view of a user's eye movements. A user's eyes move along with the keys being touched, which means that tracking the eye movement could possibly tell what the user is entering. It is of great importance to investigate whether an attacker could obtain a phone user's passcode by tracking the eye movements.

As computer vision techniques are advancing and becoming more accurate, an offline processing of the video can extract the eye position in each frame and draw the path of eye movements, which means that an attacker could infer the passcode based on the video captured by a spy camera app. In this section, we discuss two types of camera attacks for inferring passcodes.



(a) Password (b) Pattern (c) PIN

Fig. 2. Different Types of Passcodes

A. *The Application-oriented Attack*

The first type of attack is the application-oriented attack, which aims at getting the credentials of certain apps. Figure 2 gives some examples of app passcodes. The passcodes of most apps (like Facebook) that require authentication contain letters, which need a complete virtual keyboard, as shown in Figure 2(a). Figure 2(b) and Figure 2(c) show two other types of popular passcodes, pattern and PIN, which we will discuss in detail later. Smart App Protector is a locker app by which a user is able to lock apps that need extra protection (i.e. Gallery, messaging and dialing apps).

For a successful passcode inference attack, the video must be captured during user authentication. An effective way is to poll the running task list and launch the attack as soon as the target app appears on top of the list. Specifically, using the *getRunningTasks()* function of *ActivityManager*, we can get the name of most recently launched app. Therefore, the detection service scans the running apps and resource utilization periodically. When attack conditions are met, it opens the camera and secretly takes videos of the user's face (especially the eyes) with front-face camera for a time long enough to cover the entire authentication process.

There are several other factors we need to consider to ensure the attack is effective and efficient. First, the detection service of a spy camera app must be launched beforehand, by either tempting the user to run the app or registering an *ACTION_BOOT_COMPLETED* receiver to launch when booting is finished. The *RECEIVE_BOOT_COMPLETED* permission is a commonly requested permission that would not be considered dangerous. Second, polling task list frequently leads to extra consumption of energy resource. To improve the efficiency of scanning, the detection service is active only when a user is using the phone. Specifically, it will cease when the screen is off and continue when the screen is lighting up again. The status of the phone screen can be obtained by registering two broadcast receivers *ACTION_SCREEN_ON* and *ACTION_SCREEN_OFF*.

B. *The Screen-unlocking Attack*

In this subsection, we discuss another type of attack. The attack is launched when a user is entering screen unlocking passcode. We categorize this scenario as a different attack since it is unnecessary to hide the camera preview under this circumstance. To achieve privacy, Android system would not show the user interface until a visitor proves to be the device owner by entering the correct credential. This accidentally provides a shield for spy camera attacks targeted at the screen unlocking process. Users never know that the camera is working, even though the camera preview is right beneath the unlocking interface.

We demonstrate the screen-unlocking passcode inference attack. Its difference from the application-oriented attack is the condition to launch the attack and the time to stop. Intuitively, the attack should start as soon as the screen turns on and should end immediately as the screen is unlocked. This can be achieved in two key steps: (1) registering a BroadcastReceiver to receive *ACTION_SCREEN_ON* Intent when a user lights up the screen and begins the unlocking process, and (2) registering another BroadcastReceiver to receive *ACTION_USER_PRESENT* Intent when a passcode is confirmed and the screen guard is gone. The second step guarantees that the camera service would stop recording, and end itself immediately when the user interface is switched on. In addition, the attack should consider the situation in which no screen locking passcode is set. To avoid being exposed, the spy camera app should check *keyguardmanager* with the *isKeyguardLocked()* function to make sure the screen is locked before launching the attack.

To simplify the screen unlocking process, Android system provides alternative authentication methods in addition to the conventional password: pattern and PIN. A pattern is a graphical passcode composed of a subset of a 3×3 grid of dots that can be connected in an ordered sequence. There are some rules for the combination of dots: (1) the number of dots chosen must be at least 4 and no more than 9; (2) each dot can be used only once. A PIN is a pure-digit passcode with length ranging from 4 to 16, and repetition is allowed. Both alternatives are extensively used in the screen locking of Android phones. The relatively larger distance between adjacent keys effectively relieves a user's eye fatigue problem. However, this also brings vulnerability to video-based passcode inference attacks since the larger scale of eye movement makes the attack easier.

V. IMPLEMENTATION AND EVALUATION

We have implemented the video-based passcode inference attacks on real phones including the Nexus S 4G (Andriod 4.1), Galaxy Nexus (Android 4.2) and Nexus 4 (Android 4.3 with Security Enhanced support). Furthermore, a computer vision technique for eye tracking is used to process the captured video. The evaluation of the feasibility and effectiveness of the attacks is based on the experimental results.

A. *Implementation*

Both of the passcode inference attacks are successfully implemented on Android phones equipped with a front-face camera. The spy camera apps are completely translucent to phone users and work without causing any abnormal experiences. When WiFi access is enabled, the captured data is transmitted to the attacker via email. To test the stealthiness of the attacks, we install two popular antivirus apps: AVG antivirus [7] and Norton Mobile Security [8]. Neither of the two antivirus apps has reported warning during the entire video capturing and transmission process. This demonstrates their resistance to mobile anti-virus tools.

B. Feature Analysis

An important feature that enhances the effectiveness of passcode inference attacks is that it can be launched repeatedly, which allows certain passcodes to be "attacked" many times. In this way, as attacker could get a set of possible passcodes and keep launching attacks until the correct one is found.

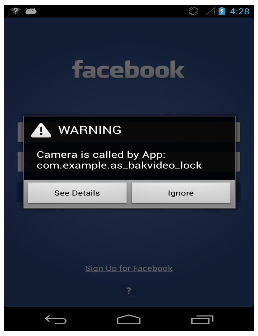
The passcode inference attack depends on the victim's eye movement instead of analyzing videos containing the phone screen [5] or its reflection [6], which makes it harder to achieve high and stable one-time success rates. In addition, there are complex factors that may influence its performance, such as the distance between face and phone, lighting condition, velocity of eye movement, pause time on each key, and head/device shaking when typing. Among these experimental conditions, only the lighting condition can be kept constant during our experiments.

To test the effectiveness of the passcode inference attacks with different types of passcodes, we use the conventional password, pattern, and PIN in our experiments. By comparing the rules of pattern and PIN, we find that pattern combination is actually a subset of PIN. In addition, the outlines of the two passcodes are similar (both are squares). Hence, we present their results and discuss their performance together. Another consideration for experiments is the length of pattern and PIN. In fact, people rarely use long PINs and complex patterns since they are hard to memorize and impractical for frequent authentications such as screen unlocking. This can be best illustrated by Apple iOS's 4-digit PIN for screen locking. In our experiments, we choose 4-digit pattern/PIN for testing.

C. Performance Evaluation

We process the videos containing user eye movement with computer vision technique for visible spectrum eye tracking. Fast Eyetracking [9] extracts feature points from faces to track eye movement, which is adopted in our experiments. Due to the tight configuration of virtual keyboard and limitation of visible spectrum imaging, the performance of inferring conventional password is poor and unstable. However, the possibility of compromising patterns and PINs is shown to be much higher. In our evaluation, 18 groups of 4-digit passcodes were tested, and the results are listed in Table I.

In Table I, each group consists of three components: real passcode (Real Psscd), eye movement (Eye Movement), and possible passcodes (Possible Psscds). Since the shape of the 9-dot pattern keypad and 10-digit PIN keypad are both similar to a square, the results of eye movement are drawn in a square. Therefore, position projection can be used to infer input keystrokes. We find that in some cases, the passcode can be inferred uniquely and accurately. While in other cases, it is in a small group of possible passcodes. For example, from the first row of Table I, the passcode "1459" can be directly inferred, while "1687" and "1450" both have three candidates. The attacker could further narrow down the possible passcode set by launching more attacks and finding out the intersection. Furthermore, when the owner is not using the phone, the attacker may try different possible passcodes and see which one works.



WARNING

Camera is called by App:
com.example.as_bakvideo_lock

Fig. 3. Defense to Spy Camera Attack

VI. Countermeasures

In this section, we discuss possible methods that can protect Android phones against these spy camera attacks.

In Android system, no application programming interface (API) or log file is available for a user to check the usage of phone camera. Hence, the detection of the camera-based attacks requires modification to the system. We make changes to the *CheckPermission()* function of *ActicityManagerService*, and write a lightweight defense app such that whenever the camera is being called by apps with CAMERA permission, the defense app will be informed along with the caller's Application Package Name. The Application Package Name is a unique identifier in Android, and third party apps cannot reuse the name of built-in apps like the default camera app (com.android.gallery3d). By looking at the app name, we are able to identify the built-in camera app (that is known to be safe) and no alert will be generated.

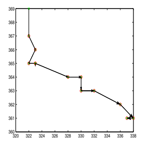
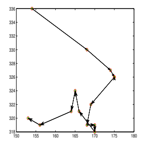
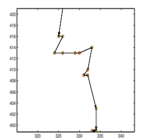
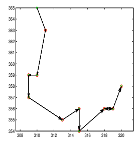
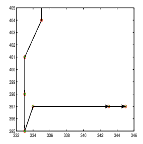
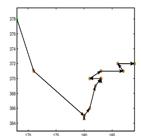
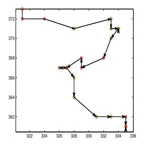
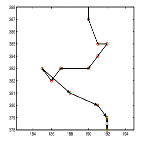
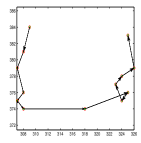
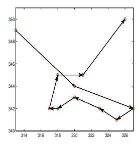
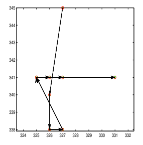
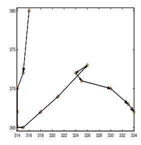
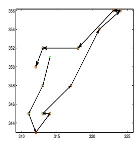
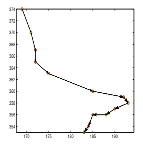
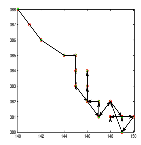
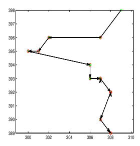
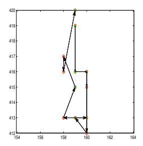
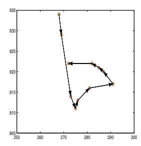
Then, we design a checking mechanism for third-party camera-based apps. Analyzing activity pattern is an effective approach in malware detection. For each type of spy camera attack, we are able to extract specific feature from its activity pattern. The application-oriented passcode inference attack launches immediately after another app runs. The screen unlocking attack runs when the screen turns on but remains locked. The defense app is able to decide the dynamic launch pattern of camera related apps by polling the task list. If an app calls the camera in one of the above manner, the defense app gives warnings to the phone user.

There are three parts of warnings in our defense scheme. First, an alert dialog including the name of the suspicious app is displayed. In case the warning message cannot be seen immediately by the user (e.g., the user is not using the phone), the defense app will also make sound and vibration to warn the user of spy camera attacks. Besides, the detailed activity pattern of suspected apps are logged so that the user can check back. As shown in Figure 3, the spy camera app named *com.example.as_bakvideo_lock* calls the camera as soon as Facebook is launched. This process is detected by the defense app, and a warning message with its name is displayed before the user enters his/her credential.

VII. Conclusion

In this paper, we study camera-related vulnerabilities in Android phones. We discover the basic camera attack, and two passcode inference attacks specified for an application and screen unlocking, respectively. We implement these attacks on

TABLE I. PASSCODE INFERENCE RESULTS FOR 4-DIGIT PASSCODES

Real Psscd	Eye Movement	Possible Psscds	Real Psscd	Eye Movement	Possible Psscds	Real Psscd	Eye Movement	Possible Psscds
1459		1459	1687		1687 16857 168587	1450		1450 1458 2569
1486		1486 14786 1786	1479		1479	1856		1856 14856
1359		1359 13659 13589 136589 12359 123659 123589 1236589	2548		2548 3659	1793		1793 1452 2563 4785 5896
1953		1953 15953	2856		2856 25856 1745 14745	1759		1759 14759
4734		4734 47534 47354 475354	1490		1490 1790	1595		1595
2450		2450 2480 2458 3569 3469	1741		1741 14741 2852 25852 5085 58085 3963 36963	1865		1865 1895 1065 1098 1095

real devices with anti-virus softwares installed. The experimental results demonstrate the feasibility and stealthiness of the attacks. We also propose a lightweight defense scheme that can effectively detect these attacks.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under grants CNS-0963578, CNS-1022552, and CNS-1065444.

REFERENCES

[1] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, 2012, pp. 3:1–3:14.

[2] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 2012, pp. 95–109.

[3] R. Schlegel, K. Zhang, X. yong Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones," in *NDSS*, 2011, pp. 17–33.

[4] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3g smartphones," in *Proceedings of the second ACM conference on Wireless network security (WiSec)*, 2009, pp. 69–78.

[5] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Proceedings of the 7th International Conference on Information Assurance and Security (IAS)*, 2011, pp. 320–325.

[6] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM conference on Computer and communications security (CCS)*, 2011, pp. 527–536.

[7] AVG AntiVirus for Android Mobiles, http://www.avg.com/us-en/antivirus-for-android.

[8] Norton Mobile Security, http://us.norton.com/norton-mobile-security/.

[9] P. Aldrian, "Fast eyetracking," http://www.mathworks.com/matlabcentral/fileexchange/25056-fast-eyetracking, 2009.