

Security Threats to Mobile Multimedia Applications: Camera-Based Attacks on Mobile Phones

Longfei Wu and Xiaojiang Du, Temple University

Xinwen Fu, University of Massachusetts Lowell

ABSTRACT

Today's mobile smartphones are very powerful, and many smartphone applications use wireless multimedia communications. Mobile phone security has become an important aspect of security issues in wireless multimedia communications. As the most popular mobile operating system, Android security has been extensively studied by researchers. However, few works have studied mobile phone multimedia security. In this article, we focus on security issues related to mobile phone cameras. Specifically, we discover several new attacks that are based on the use of phone cameras. We implement the attacks on real phones, and demonstrate the feasibility and effectiveness of the attacks. Furthermore, we propose a lightweight defense scheme that can effectively detect these attacks.

INTRODUCTION

Since 2007, the Android operating system (OS) has enjoyed an incredible rate of popularity. As of 2013, the Android OS holds 79.3 percent 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 permission requests stand for; 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 app markets. Most large anti-virus software companies have published their Android-version security apps, and tried to provide a shield for smartphones by detecting and blocking malicious apps. In addition, there are data protection apps that provide users the capability to encrypt, decrypt, sign, and verify signatures for private texts, emails,

and files. However, mobile malware and privacy leakage remain a big threat to mobile phone security and privacy.

Generally, when talking about privacy protection, most smartphone users pay attention to the safety of SMS, emails, contact lists, calling histories, 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, various types of camera-based applications have appeared in Android app markets (photography, barcode readers, social networking, etc.). Spy camera apps have also become quite popular. As for Google Play, there are nearly 100 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 cameras in malicious apps such that the phone camera is launched automatically without the device owner's notice, and the captured photos and videos are sent out to these remote attackers. Even worse, according to a survey on Android malware analysis [1], camera permission ranks 12th of the most commonly requested permissions among benign apps, while it is out of the top 20 in malware. The popularity of camera usage in benign apps and relatively less usage in malware lower users' alertness to camera-based multimedia application attacks.

Nowadays, people carry their phones everywhere; hence, their phones see lots of private information. If the phone camera is exploited by a malicious spy camera app, it may cause serious security and privacy problems. For example, the phone camera may record a user's daily activities and conversations, and then send these out via the Internet or multimedia messaging service (MMS). Secret photography is not only immoral but also illegal in some countries due to the invasion of privacy. Nevertheless, a phone camera could also provide some benefits if it is con-

trolled well by the device owner. For example, when the owner wants to check if someone has used his/her phone without permission, the phone camera could be used to record the face of an unauthorized user. Besides, it can also help the owner find a lost phone.

In this article, we first conduct a survey on the threats and benefits of spy cameras. Then we present the basic attack model and two camera-based attacks: the remote-controlled real-time monitoring attack and the passcode inference attack. We run these attacks along with popular antivirus software to test their stealthiness, and conduct experiments to evaluate both types of attacks. The results demonstrate the feasibility and effectiveness of these attacks. Finally, we propose a lightweight defense scheme.

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 [2] 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.* [3] 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 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 an Android system. In this work, we are able to hide the whole camera app in Android. Moreover, we implement advanced forms of attacks such as remote-controlled and real-time monitoring attacks. We also utilize computer vision techniques to analyze recorded videos and infer passcodes from users' eye movements.

Several video-based attacks targeted at keystrokes have been proposed. The attacks can obtain user input on touch screen smartphones. Maggi *et al.* [4] 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. However, this attack requires an additional camera device, and issues like how to place the camera near the victim without catching an alert must be considered carefully. Moreover, it works only when visual feedback such as magnified keys are available. iSpy [5], proposed by Raguram, shows how screen reflections may be used for reconstruction of text typed on a smartphone's virtual keyboard. Similarly, this attack also needs an extra device to capture the reflections, and the visual key press confirmation mechanism must be enabled on the target phone. In contrast, our camera-based attacks work without any support from other devices.

THREATS AND BENEFITS OF SPY CAMERA

As mentioned above, the role a spy camera plays depends on the way it is used and who is in control of it. In the following, we discuss some threats and benefits of using a spy camera.

LEAKING PRIVATE INFORMATION

A spy camera works as a thief if it steals private information from the phone. First, the malware finds a way to infect the victim's smartphone. For example, it appears to be a normal app with legitimate use of a camera and the Internet. On one hand, it performs the function it claims. On the other hand, it runs a background service to secretly take pictures or record videos, and store the data with obscure names in a directory that is seldom visited. Then these data are sent out to the attacker when WiFi (fast and usually unlimited) access or other connection is available.

WATCHDOG

Watchdog is another thing a spy camera can do. Nobody wants other people to use or check his/her phone without permission. A spy camera can stealthily take pictures of the phone user and deter those who use or check other people's phones.

ANTI-THIEF

On the other hand, a spy camera could play a completely different role if it is used properly. When a user loses his/her phone, the spy camera could be launched via remote control and capture what the thief looks like as well as the surrounding environment. Then the pictures or videos along with location information (GPS coordinates) can be sent back to the device owner so that the owner can pinpoint the thief and get the phone back.

THE BASIC CAMERA ATTACK MODEL

We want to discover possible attacks based on a 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.

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

Nowadays, people carry their phones everywhere; hence, their phones see lots of private information. If the phone camera is exploited by a malicious spy camera app, it may cause serious security and privacy problems.

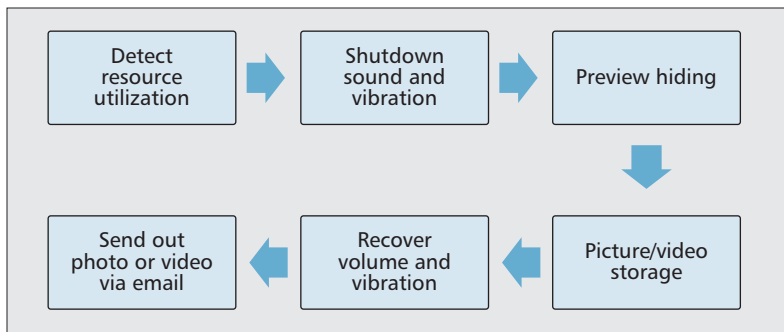


Figure 1. The basic camera attack architecture.

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`.

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.

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 set the view parameters by changing 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.

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.

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.

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 a video should have an upper bound specific to the email service.

THE REMOTE-CONTROLLED REAL-TIME MONITORING ATTACK

The basic camera attack can be further enhanced to more aggressive attacks. For example, the attacker can remotely control the spy camera app such that the time to launch and end the attack is under control. The simplest way to implement the remote control is by socket. After the malicious app is downloaded and installed on a victim’s phone, it sends a “ready” message along with the IP address and port number to the attacker’s server. Then the attacker can control the app with orders like “launch” and “stop” or specify a time schedule.

There are many Android apps that turn the phone into a security surveillance camera, such as Android Eye [6]. The spy camera can easily be extended to a stealthy real-time monitor based on the way an IP camera is built. `NanoHttpd` [7] is a lightweight HTTP server that can be installed on a phone. In our case, we can start an HTTP server at a given port which supports dynamic file serving such that the captured videos can be played online upon requests from a browser client. Figure 2 shows the video taken by a real-time spy camera of a mobile phone. Figure 2a is the environment in which an Android phone is located. Although the phone’s screen is showing its app menu, it actually captures videos through the front-face camera. Figure 2b is the view of the phone camera, which is accessed from a PC browser. The address is the IP address of the phone and the port number of the server.

In this section, we discuss the remote-controlled real-time monitoring attack, which could pose a big threat to a phone user’s privacy: daily activities and surrounding environment are all under the eye of the attacker. Camera-based attacks can be detected when multiple apps request the camera device at the same time or if the camera is being used by another app. But this can easily be avoided by selecting the time to launch attack. The malicious camera app can periodically check the screen status and run the stealthy video recording only when the screen is off, which means that the user is not using the phone and the camera device is idle. The status of the phone screen can be obtained by registering two broadcast receivers, `ACTION_SCREEN_ON` and `ACTION_SCREEN_OFF`.

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 measurement of 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. Thus, 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. We also discuss the computer vision techniques for eye tracking that can be utilized in the attacks.

THE APPLICATION-ORIENTED ATTACK

The first type of attack is the application-oriented attack, which aims at getting the credentials of certain apps. Figure 3 gives some examples of app passcodes. Most apps (like Facebook) that require authentication contain letters, which need a complete virtual keyboard, as shown in Fig. 3a. Figures 3b and 3c show two other types of popular passcodes, pattern and PIN, which we 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 *Activity-Manager*, we can get the name of the most recently launched app. Meanwhile, 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 a 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 lists frequently leads to extra consumption of energy resource. To improve the efficiency of

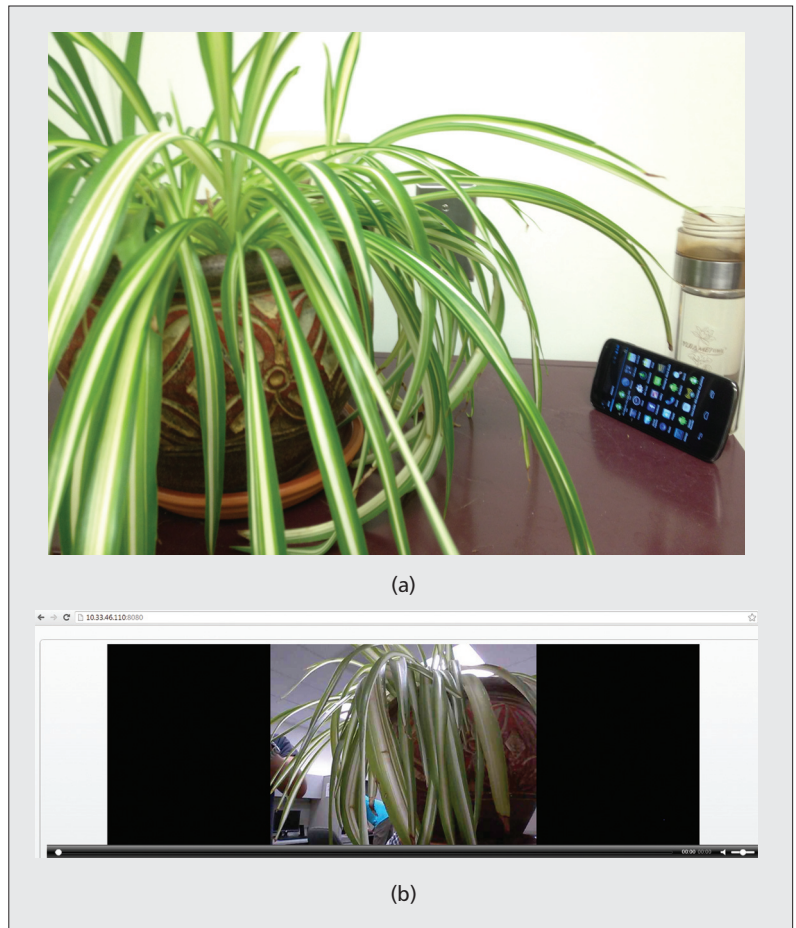


Figure 2. Demo of the real-time monitoring attack: a) overall view of the phone environment; b) scene captured by phone camera.

scanning, the detection service is active only when a user is using the phone. As mentioned before, this can be determined by screen status. The detection service will cease when the screen is off and continue when the screen lights up again. Moreover, the scanning frequency should be set properly. In a phishing attack [8], a malicious app needs to poll the running task list every 5 ms to prevent the user from noticing that a new window (the fake app) has replaced the original one. In our phone camera attack, the view is totally translucent to users, so that worry is unnecessary. However, we still need to keep the frequency at around two scannings per second; otherwise, the attack may happen after the user starts entering the passcode (which makes the attack unsuccessful).

THE SCREEN UNLOCKING ATTACK

In this subsection, we discuss another type of attack. The attack is launched when a user is entering a screen unlocking passcode. We categorize this scenario as a different attack model since it is unnecessary to hide the camera preview under this circumstance.

To achieve privacy, an 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:

- Registering a BroadcastReceiver to receive *ACTION_SCREEN_ON* when a user lights up the screen and begins the unlocking process
- Registering another BroadcastReceiver to receive *ACTION_USER_PRESENT* 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

with no screen locking passcode. 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, the 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:

- The number of dots chosen must be at least 4 and no more than 9.
- 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 screen unlocking 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.

VIDEO-BASED EYE TRACKING TECHNIQUES

In the eye tracking field, two types of imaging approaches are commonly used: visible and infrared spectrum imaging. Visible spectrum imaging passively utilizes the ambient light reflected from the eye, while infrared spectrum imaging is able to eliminate uncontrolled specular reflection with active infrared illumination. Although infrared spectrum eye tracking is more accurate, most smartphones today are not equipped with infrared cameras. Hence, we focus on visible spectrum eye tracking. For images captured by visible spectrum imaging, often the best feature to track is the contour between iris and sclera known as the limbus [9].

Li *et al.* [9] propose the Starburst eye tracking algorithm, which can track the limbus of the eye. As we can see from Fig. 4a, in visible spectrum, they can locate where the eye is looking in a real-time manner. However, Starburst requires calibration by manually mapping between eye-position coordinates and scene-image coordinates. This can be performed only by the phone owner, which makes it infeasible in spy camera attacks.

Aldrian [10] presents a method to extract fixed feature points from a given face in visible spectrum, which is based on the Viola Jones adaboosted algorithm for face detection. But it is able to track pupil movement without scene image and calibration, as shown in Fig. 4b. We adopt this eye tracking algorithm in our research to extract eyes from videos.

IMPLEMENTATION AND EVALUATION

We have implemented the remote-controlled real-time monitoring and passcode inference attacks on real phones including the Nexus S 4G (Android 4.1), Galaxy Nexus (Android 4.2), and Nexus 4 (Android 4.3 with Security Enhanced support). For passcode inference

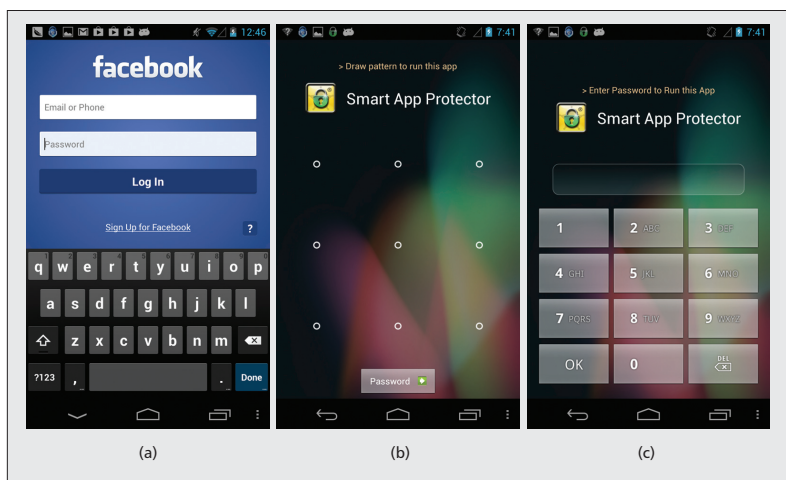


Figure 3. Different types of passcodes: a) password; b) pattern; c) PIN.

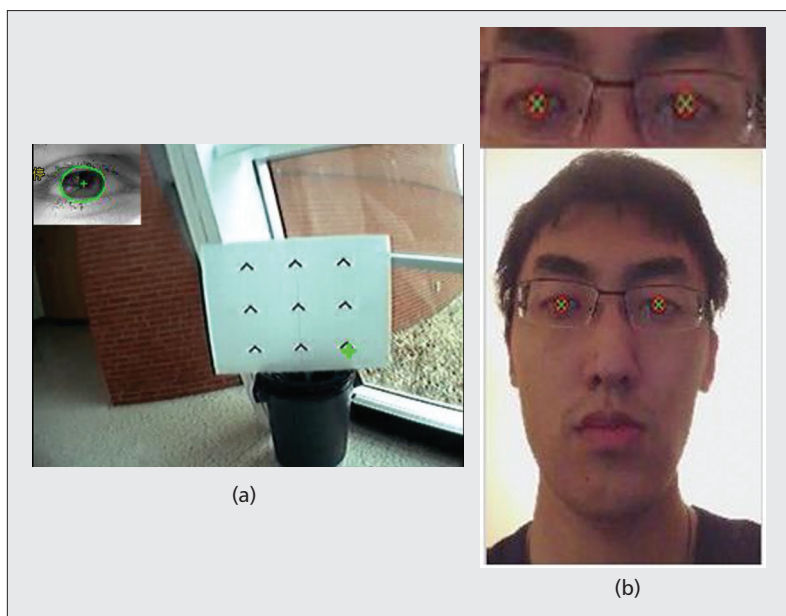


Figure 4. Demo of existing eye tracking techniques: a) Starburst eye tracking demo; b) fast eye tracking demo.

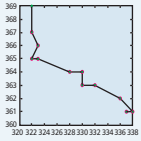
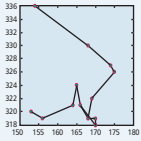
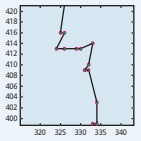
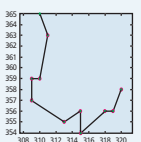
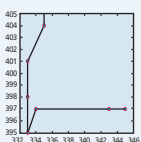
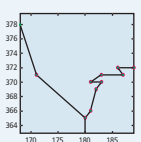
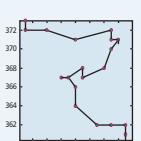
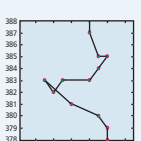
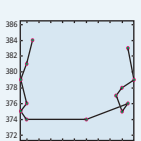
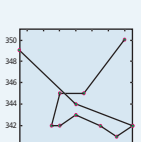
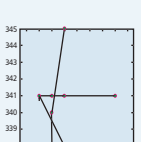
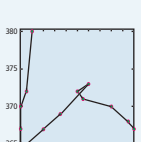
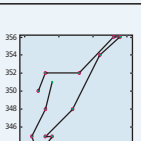
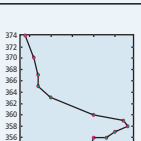
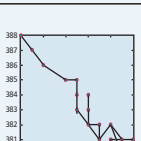
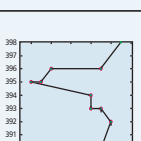
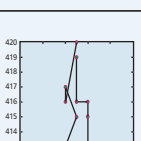
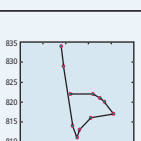
Real Psscd	Eye movement	Possible Psscds	Real Psscd	Eye movement	Possible Psscds	Real Psscd	Eye movement	Possible Psscds
1459		1459	1687		1687 16857 168587	1450		1450 1458 2569
1486		1486 14786 1786	1479		1479	1856		1856 14856
1359		1359 13659 13589 136589 12359 123659 123589 1236589	2548		2548 3659	1793		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

Table 1. Passcode inference results for four-digit passcodes.

attacks, 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.

IMPLEMENTATION

Both camera-based 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 a local HTTP server or email. To test the stealthiness of the attacks, we install two popular antivirus apps: AVG antivirus and Norton Mobile Security. Neither of the two antivirus apps has reported warning during the entire video capturing and transmission process. This demonstrates their resistance to mobile antivirus tools.

FEATURE ANALYSIS OF THE PASSCODE INFERENCE ATTACK

An important feature that enhances the effectiveness of a passcode inference attack is that it can be launched repeatedly, which allows certain passcodes to be “attacked” many times. In this way, an 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 screen [4] or its reflection [5], 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 conditions, velocity of eye movements, 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.

The defense app is able to decide the dynamic launch pattern of camera related apps by polling the task list. If a camera app calls camera in one of the above situations, the defense app would give warnings to the phone user.

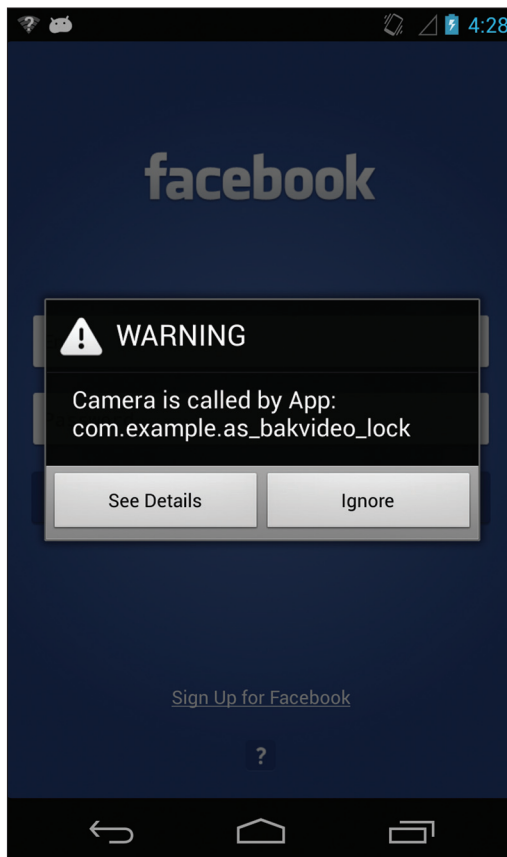


Figure 5. Defense against spy camera attack.

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 four-digit PIN for screen unlocking. Hence, in our experiments we choose a four-digit pattern/PIN for testing.

PERFORMANCE EVALUATION

We process videos containing user eye movement with the aforementioned computer vision technique [10]. Due to the tight configuration of the virtual keyboard and limitation of visible spectrum imaging, the performance of inferring a 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 1.

In Table 1, each group consists of three components: real passcode (Real Psscd), eye movement, and possible passcodes (possible Psscds). Since the shape of the 9-dot pattern keypad and

10-digit PIN keypad is 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 correct passcode can be inferred accurately, while in other cases, it is in a small group of possible passcodes. For example, from the first row in Table 1, 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.

COUNTERMEASURES

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

In an Android system, no application programming interface (API) or log file is available for a user to check the usage of a camera device. Hence, detection of camera-based attacks requires modification to the system. We make changes to the *CheckPermission()* function of *ActivityManagerService* and write a lightweight defense app such that whenever a 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 a Camera (*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 further 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 a specific feature from its activity pattern. The activity pattern of spy camera apps are totally different from legitimate camera apps. To avoid collision in the use of cameras, a remote-controlled real-time monitoring attack runs when the screen is off. 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 a camera app calls a camera in one of the above situations, 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 later. As shown in Fig. 5, 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 credentials.

CONCLUSION

In this article, we study camera-related vulnerabilities in Android phones for mobile multimedia applications. We discuss the roles a spy camera can play to attack or benefit phone users. We discover several advanced spy camera attacks, including the remote-controlled real-time monitoring attack and two types of passcode inference attacks. Meanwhile, we propose an effective defense scheme to secure a smartphone from all these spy camera attacks. In the future, we will investigate the feasibility of performing spy camera attacks on other mobile operating systems.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation under grants CNS-0963578, CNS-1022552, CNS-1065444, CNS-1116644, and CNS-0958477.

REFERENCES

- [1] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *IEEE Symp. Security and Privacy 2012*, 2012, pp. 95–109.
- [2] R. Schlegel *et al.*, "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones," *NDSS*, 2011, pp. 17–33.
- [3] N. Xu *et al.*, "Stealthy Video Capturer: A New Video-Based Spyware in 3g Smartphones," *Proc. 2nd ACM Conf. Wireless Network Security*, 2009, pp. 69–78.
- [4] F. Maggi, *et al.*, "A Fast Eavesdropping Attack against Touchscreens," *7th Int'l. Conf. Info. Assurance and Security*, 2011, pp. 320–25.

- [5] R. Raguram *et al.*, "ispy: Automatic Reconstruction of Typed Input from Compromising Reflections," *Proc. 18th ACM Conf. Computer and Commun. Security*, 2011, pp. 527–36.
- [6] "Android-eye," <https://github.com/Teaonly/android-eye>, 2012.
- [7] "Nanohttpd," <https://github.com/NanoHttpd/nanohttpd>.
- [8] A. P. Felt and D. Wagner, "Phishing on Mobile Devices," *Proc. WEB 2.0 Security and Privacy*, 2011.
- [9] D. Li, D. Winfield, and D. Parkhurst, "Starburst: A Hybrid Algorithm for Video-Based Eye Tracking Combining Feature-Based and Model-Based Approaches," *IEEE Computer Soc. Conf. Computer Vision and Pattern Recognition — Workshops*, 2005, p. 79.
- [10] P. Aldrian, "Fast Eyetracking," <http://www.mathworks.com/matlabcentral/fileexchange/25056-fast-eye-tracking>, 2009.

BIOGRAPHIES

LONGFEI WU (longfei.wu@temple.edu) is a Ph.D. candidate in the Department of Computer and Information Sciences at Temple University under the supervision of Dr. Xiaojiang Du. He received his B.E. degree in communication engineering from Beijing University of Posts and Telecommunications, China, in 2012. His research interests include smartphone security and wireless networks.

XIAOJIANG (JAMES) DU [SM] (dux@temple.edu) is an associate professor in the Department of Computer and Information Sciences at Temple University. His research interests are security, cloud computing, wireless networks, and computer networks and systems. He has published over 100 journal and conference papers. He has been awarded more than \$3 million in research grants from the U.S. NSF and Army Research Office. He is a Life Member of ACM.

XINWEN FU (xinwenfu@cs.uml.edu) is an associate professor in the Department of Computer Science, University of Massachusetts Lowell. He obtained his Ph.D. (2005) in computer engineering from Texas A&M University. His current research interests are in network security and privacy, and digital forensics. He has published papers in conferences such as IEEE S&P, ACM CCS, and ACM MobiHoc; and in journals such as *ACM/IEEE Transactions on Networking*. His research is supported by NSF.