# On Energy Security of Smartphones

Xing Gao[1,2], Dachuan Liu[1,2], Daiping Liu[1],Haining Wang[1]
[1]University of Delaware, Newark, DE, USA
[2]College of William and Mary, Williamsburg, VA, USA
{xgao, dachuan, dpliu, hnw}@udel.edu

## ABSTRACT

The availability of smartphones is still severely restricted by the limited battery lifetime. To help users understand the energy consumption, major mobile platforms support fine-grained energy profiling for each app. In this paper, we present a new threat, called energy collateral attacks, which can abuse and mislead all existing energy modeling approaches. In particular, energy collateral attacks are able to divulge battery stealthily through interprocess communication, wakelock, and screen. To defend against those attacks, we propose E-Android to accurately profile the energy consumption in a comprehensive manner. E-Android monitors energy collateral related events and maintains energy consumption for relevant apps. We utilize E-Android to measure the energy consumption under the attack of six energy malware and two normal scenarios. While Android fails to disclose all these energy-malware-based attacks, E-Android can accurately profile energy consumption and reveal the existence of energy malware.

## 1. INTRODUCTION

Smartphones have brought great convenience to our daily life. However, the limited battery still seriously impacts the availability of smartphones. Previous works center on profiling energy consumption of each app. They breakdown the power consumption of a smartphone by every component and build energy models. A well designed battery interface is further developed to visualize energy consumption to smartphone users. As a result, existing energy malware is not hard to detect under the combination of energy accounting and battery interface.

In this paper, we reveal that a set of mechanisms could be exploited to cause biased energy profiling and user confusion. Based on those mechanisms, we propose new energy attack techniques that can neatly sidestep current energy accounting, resulting in stealthy energy consumption in Android smartphones. We term them as energy collateral attacks.

Android apps rely on inter-process communication (IPC) for communication. Current energy accounting approaches overlook IPC in Android. Our first attack vector is IPC-based, in which energy malware can trigger other innocent apps to waste energy through the IPC mechanism.

The second attack vector is wakelock- and screen-based. Existing methods to model the energy consumption of screen fall into two categories. The first takes screen as an independent module. The second allocates screen energy to foreground app. Both mechanisms give malicious apps the possibility to deceive energy accounting. Energy malware could simply enhances the brightness or uses the screen wakelock to keep screen on. Besides screen, wakelock could also be acquired to keep CPU awake. Failing to release the wakelock can drain tremendous amount of extra energy. Thus, by interrupting other apps to release wakelock, malicious app can force the system running in a high power state.

To defend against energy collateral attacks, we design E-Android to reveal collateral energy consumption. E-Android consists of three major components: (1) an extension of Android framework, (2) an enhanced energy accounting module, and (3) a revised battery interface. E-Android collects apps' user IDs and the type of operations. E-Android also carefully monitors the activities of task stacks to accurately identify an energy collateral attack. The collateral energy consumption will be appropriately charged to the initiated source. E-Android is able to handle sophisticated situations such as collateral attack chains. We modify two battery interfaces to work jointly with E-Android.

We conduct a series of experiments to evaluate energy collateral attacks as well as the functionality of E-Android. We build six types of energy malware running in a Nexus 4 mobile phone. Our results indicate that all energy collateral attacks can successfully waste energy without being noticed. They also prove that E-Android is able to detect and expose all energy malware.

## 2. ENERGY COLLATERAL ATTACK

### 2.1 Potential Attack Vectors

**IPC-based Attack Vector.** We first look into a simple scenario. Bob uses the Message app to film a half minute video and send it to Alice. A small camera window is embedded in the Message UI. All operations seem to occur in the Message app. Thus, it is reasonable to expect that the Message should be accounted for the corresponding energy consumption. We illustrate the energy consumption measured by Android official app in Figure 1. The result, however, indicates that the Message only consumes a quite small



*CODASPY'16, March 9–11, 2016, New Orleans, LA, USA.*



ACM ISBN 978-1-4503-3935-3/16/03..

DOI: http://dx.doi.org/10.1145/2857705.2857738

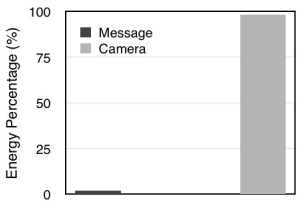


Figure 1: Energy consumption when filming in the Message.

portion of energy. The energy drained by video filming is assigned to the Camera.

The fact is that, when Bob clicks "Record Video" in the Message, the Message sends an *Intent* to request the Camera app. It is the Camera app that actually records the video. After the recording, the video is returned to the Message app. Interestingly, camera is reported as the most energy draining app [1]. However, we observe that energy consumption sometimes involves the communications between apps, and hence other apps should also be responsible for those indirect energy consumptions.

This scenario frequently occurs in Android since IPC serves as the only approach for the communication between apps. While the intent mechanism plays a critical role in Android, existing energy accounting modules overlook such factors.

**Wakelock- & Screen-based Attack Vector.** Android turns a device into deep sleep after some idle time to save energy. To override the aggressive power saving policy, Android introduces the wakelock, which is a special power management module to keep devices awake.

A wakelock must be released once acquired. Otherwise, battery will be drained up to 25% per hour [2]. Android does not release wakelock until the process has been killed. Pathak et al. [2] observed that a lot of developers fail to understand how to properly use wakelock. One improper usage is that, an app only releases wakelock when the process is destroyed. Normally, the app would be destroyed when the user quits the app, without causing any problem. However, a foreground activity could be easily interrupted by popup activities and then fails to properly release wakelock.

Mis-releasing wakelock also challenges the screen modeling policy. A wakelock could be acquired by a service. If the consumed energy is only allocated to the foreground app rather than the initiator, the energy modeling would confuse users on the internal energy consumptions.

## 2.2 Energy Malware

We define energy collateral attack as the misconducts that drain energy by exploiting specific attack vectors without being exposed via the battery interface. We assume that malware has been installed on the device with some necessary permissions. Also, several victims have been installed.

**Attack #1.** *Energy malware hijacks components belonging to other apps.* Existing energy modeling does not consider IPC communication. This mechanism enables malware to drain the target device's energy through a combination of legal operations, and hence bypass the energy monitoring.

**Attack #2.** *When malware is launched by user, malware open other apps concurrently and make them run in background.* It has been long reported that a background app definitely drains battery. Thus, triggering background apps is a very effective way to drain battery.

**Attack #3.** *Bind to services without unbinding.* Malware could further launch attacks on background services, where heavy computational workload normally runs as recommended by Android. A started service will not be terminated even the started component is destroyed and must be stopped by `stopService()` or `stopSelf()` to avoid running indefinitely. Similarly, a bound service must be unbound. Thus, an exported service bound by malware will keep alive infinitely and drain battery even after the victim attempts to stop service.

**Attack #4.** *Interrupt attacked apps to background.* Malware could also forcibly switch the victim to background through normal operations such as opening the launcher. Sophisticated malware could utilize other techniques to interrupt the foreground app without being noticed by users. For example, attackers can switch the app to background when the user attempts to quit the app.

The misinterpretation of wakelock could make the energy attack even more serious. Since the app enters background instead of being killed, it might fail to release wakelock. Since the wakelock is un-released by the victim, energy accounting will tax the energy into the victim, without disclosing malware behind curtain.

**Attack #5.** *Drain energy through changing screen configuration.* For both screen accounting policy, malware could easily bypass the battery interface. The brightness acts as the determining factor for screen energy consumption. Malware could change the brightness. Many apps enhance the brightness when they are running in foreground. Therefore, users probably would not perceive the malicious adjustment of brightness by malware. In particular, to avoid being noticed, malware could secretly escalate the brightness with a few levels. Such a slight enhancement might not affect users, but cut the battery lifetime. Advanced energy malware could camouflage as Android auto screen settings, by setting a higher value after obtaining current auto set brightness.

**Attack #6.** *Acquire screen wakelock without releasing.* Wakelock could also be utilized to conduct screen energy attack. Malware could easily keep screen on by intentionally acquiring but not releasing wakelock. The wakelock could even be acquired by service. The consumed part of screen energy will be allocated to the foreground app or Android launcher, rather than malware.

**Multi- & Hybrid Attack.** Sophisticated malware could combine the above attacks together for more effective attacks. Attackers could also conduct multiple attacks on the same attacked app. Also malware could conduct an attack on one victim, which unintentionally involves another, leading energy attack chains.

**Attack Scenarios.** Energy collateral attacks could be abused to launch denial of service attacks. Malware could reduce the battery's lifetime and degrade a user's experience. Furthermore, energy collateral attacks could mislead a user's attention to an innocent app and cause unfair competition.

Unlike traditional attacks targeting at leaking personal private information or controlling system resources, energy attacks aim to significantly reduce the battery's lifetime, as the battery is the most scarce resource of a mobile device. Also, energy attacks could be conducted accompanying with traditional attacks.

# 3. DEFENSE

We introduce E-Android to assist the battery interface to reveal collateral energy consumption. E-Android is com-

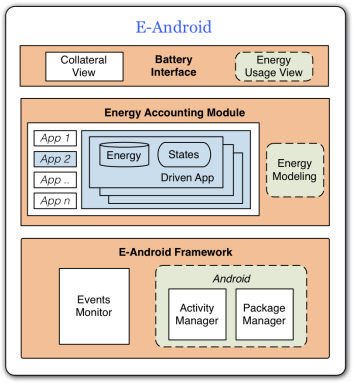



Figure 2: E-Android Architecture

posed of three major components: an extension of Android to log all potential energy operations, an enhanced energy accounting module to calculate energy consumption considering the collateral effects, and a revised battery interface to inform users of all related information.

We demonstrate the architecture of E-Android in Figure 2. Basically, E-Android monitors a series of events that potentially lead to energy collateral attack, e.g., starting an activity, changing screen settings. Each time an event is triggered, E-Android checks the user ID of both apps. If different, E-Android records the user ID of both apps as well as the type of the operation and notifies energy accounting module. The module then updates the relevant energy data.

The battery interface itemizes apps that consume a lot of energy. E-Android ranks apps by total energy consumptions with collateral energy combined. Moreover, for each app, E-Android provides a detailed inventory specifying contribution of all attack related apps. The breakdown assists users in better understanding the energy consumption.

We modified the framework of Android 5.0.1 to implement E-Android. We included the collateral attack modeling features to both Android official battery interface and powerTutor [3].

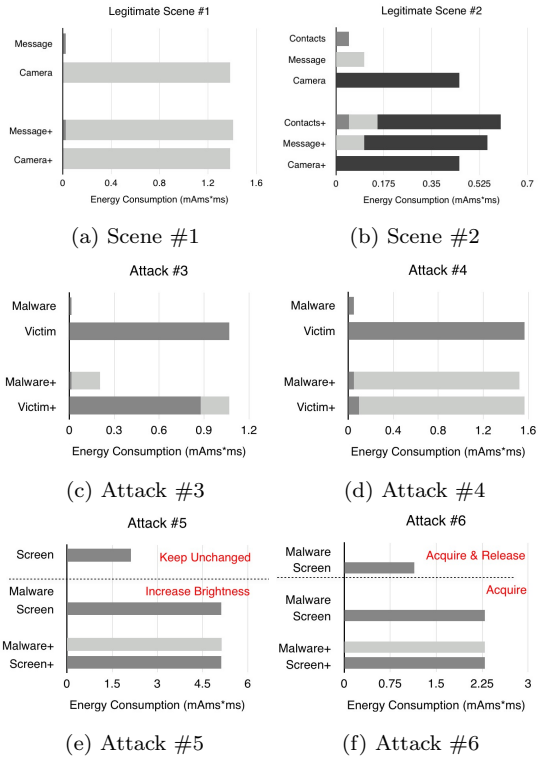
## 4. EXPERIMENTS

We demonstrated that energy collateral attacks could sidestep Android and deplete battery. To verify the functionality of E-Android, we compared the results of E-Android with those of Android.

We first conducted experiments simulating reality cases. We opened the Message app 30 seconds and then used it to take a 30s short video. Such case is similar to the malware attack #1. A more complicated scenario, which is similar to hybrid attack, is that we use the Contacts to open the Message, then films a 30s video.

Figures 3a and 3b illustrates the results for both reality cases. The '+' stands for the results of E-Android. In the original energy modeling, the Camera app expends much more energy than the Message app, regardless of the fact that the Camera is opened by the Message. In E-Android, the Message is also charged for the portion of energy consumed by the Camera.

We also implemented six types of energy malware as we mentioned before. Each experiment lasts 60 seconds. In the first two cases, malware either directly or simultaneously opens other activities. The results are similar. For malware




(a) Scene #1 (b) Scene #2

(c) Attack #3 (d) Attack #4

(e) Attack #5 (f) Attack #6

Figure 3: Functionality results

#3, our malware binds the victim's service once it detects the service is started. The connection bound by malware forces the service to run continuously after the attacked app stops the service. The malware #4 detects the quit dialog of the victim based on the shared virtual memory size of SurfaceLinger. It sends a transparent page to cover the dialog and start homepage once "OK" is clicked. In attack #5, we first measured regular screen energy consumption. Then, we measured the energy consumption after malware enhances brightness. While Android turns screen off after 30s, we measured the energy consumed by screen for 60 seconds under the conditions that malware #6 releases/not releases the wakelock.

The value is directly read from the Android battery interface. We displayed the results of attack #3 and attack #4 in Figures 3c and 3d, as well as the results of attack #5 and attack #6 in Figures 3e and 3f. As the figures show, all those attacks could successfully bypass the supervision of the battery interface. However, E-Android can detect all these attacks.

## 5. REFERENCES

[1] Top 10 android battery-sucking vampire apps. http://betanews.com/2014/02/27/top-10-android-battery-sucking-vampire-apps/.

[2] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *ACM MobiSys*, 2012.

[3] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.