International Journal of Teknology Informatics and Engineering
e-ISSN :2961-8215;.p-ISSN :2961-9068
Volume 1 No. 3 , Desember - 2022

THREAT ATTRIBUTES HANGING IN THE WILD ANDROID

Irdha Yunianto
Universitas Sains dan Teknologi Komputer

Mars Caroline Wibowo
Universitas Sains dan Teknologi Komputer

Budi Raharjo
Universitas Sains dan Teknologi Komputer

Abstract. Android is a complicated system that applications and component are usable and support for multiple work together, giving rise to highly complex interdependence relationships. Meanwhile, the Android environment is notable for being greatlty disparate and decentralized: different Operation System version is personalized and re-personalized by different parties about fast and used by whoever that can develop an application for that version. Android secure its explanation sources over an app sandbox and permissions model, where each application execution in this part can entrance only suspectible overall assets and another application component (value providers, services, activities, publication receivers) by the appropriate liscense.
This study uses Harehunter measurement to automatically detect Hare vulnerabilities in Android system applications. Harehunter and HareGuard performance evaluations were carried out in this study, both of which proved to be highly effective. The approach used here is divergent investigation, by searching all quoted, decompiled script, and obvious data for targeted attribute determination as an initial step, and running an XML parser. The outcome of this research show that the impact of Hares is very significant. The application of HareGuard in this study proved to be effective in detecting all attack applications that were made. Further evaluation of the performance impact on the minimum system host. For future research, to make Harehunter more effective, it is suggested to use a more qualified analyzer. So that this direction can be explored in more depth.

Keywords: Harehunter, HareGuard, Android, Android Malware Detection.

PRELIMINARY

What's causing the issue here is the intrinsic interdependence relationship in different Android part like as framework application and services, linking one function to the other over matrix to the final’s attributes like package, activity, service name, provider authority content, and permissions. Adjustments made to these components, if not carefull on this then it can easy breach some of these relationships, outcoming in matrix to charaacteristic that don't exist (for example, SMS/MMS provider authority is not present on the tablet), in this study referred to as attribute references, hanging, or hares.

As other impact of Android's dissolution, Hares can also be involved by 3 component developers that design their applications to install on different versions of Android in the particular service part they use. As example, references to ordering value

providers that don't exist can also be stuck in 3-part apps that undertake on smartphones and tablets. “Compared to the personalization flaws found in previous research, namely about Linux layer device driver misconfigurations, hanging references are a scheme sheet problem and prospectly highly effluence, bring the case that the networks and applications system at that sheet are always the straight on personalization” (Nappa et al., (2012)). However, as the issue that never be fixed, its waeanty implications, field, and dimensions are, consequently unclear at all.

RELATED WORK AND LITERATURE REVIEW

Android Security Demystification

Android Security Survey

A high-level of Android waranty serve in the study of Chaudhuri et al., (2011) aims to better understand Android application attack vectors through a systematic characterization of popular Android application security. The authors consider various issues including malicious functionality and vulnerabilities and propose the Dalvik parser for analysis. Both works highlight concerns in excellence application and proportition Android's liscense-based safety design. A similar study Christin et al., (2011) provides another survey on Android security in general. The first job provides a mobile platform taxonomy of attack classes with specific examples (such as Android repackaging attacks, remote payload execution, etc.) as each class applies to the Android environment, then proposes mitigations where possible. Recently, SoK by Smith et al., (2016) systematized “study about Android waranty and privacy on the app platform”. To equitably estimate and match the existing various method, first is general understanding of the various challenges and attack models that threaten the Android ecosystem, and then produce a cohesive concept of the striker abilities.

Android permissions

Android permissions are aimed to safe important Android assets at the scheme layer. Optionally, apps can use built-in or custom permissions to protect their resources (value operators, services, activities, or broadcast receivers). Research by Wagner et al., (2011) utilizes dynamic analysis to demystify android permission usage, by mapping the API to its permission requirements. To overcome some of the limitations of Stowaway (Wagner et al., (2011)), PScout (Lie et. al., (2012)) proposed a fixed investigation instrument whose aim is to generate a complete specification for Android's permissions system that lists the permission requirements for each API call. This tool performs affordability analysis between API calls and permission checks across Android source code. A study conducted by Cowan et al., (2012) proposes a real realization of trusted UI in the form of access control gadgets that enable user-based representation of liscense to applications when the gizmo can be successfully segmental into application systems. Research conducted by Beznosov et al., (2015) aims to increase the effectiveness of Android permissions by using the idea of separation as dependent fairness. More specifically, the authors propose a new permission request model, which would only prompt the user when an app accesses sensitive data in a way that goes against the user's expectations. Sadeh et al., (2014) propose to reduce the list of permissions faced by users at the time of application installation and replace it with a concise list that reflects the privacy profile. Another study conducted by Xie et al., (2013) and Chen et al., (2014) on Android permissions uses NLP techniques to analyze Android application descriptions, obtain the required permissions, and check whether the permissions match the effective permissions requested. in the android manifest file. Yin et al., (2015) took a different

approach and produced a security-centric application description from an analysis of the application code to educate Android users about understanding the functionality of a particular application at installation time.

Android and Web

Other researchers are focused on uncovering vulnerabilities in specific Android applications across the web landscape. Yinet al., (2011) presented the first systematic study of WebView security issues and found several attacks that revealed underlying problems in TCB and the weakened sandbox of WebView Android infrastructure. Chen et al., (2013) conducted another systematic study to understand unauthorized origin crossing of mobile OS and highlight the existence of such vulnerabilities in high-profile applications. Research conducted by Peri et al., (2014) has broadened the field to wrap the code shot strike on all HTML5-based cellular applications. The strike especially goal effective applications that take advantage of the "WebView" appearance that available in Android.

Previous research by Wagner et al., (2011) has studied the receipt of illegitimate intent where attackers can hijack activities and services if there is an implicit intent. The research not immediately address Hare's weaknesses as it unwanted any legitimate activities/services to be referenced. Instead, it addresses facts that various receivers are current on n instrument. Other study includes evaluating security risks resulting from design flaws in push-cloud messaging (Han et al., (2014)), identifying risks of Android app uninstall processes (Qiu et al., (2016)) and risks of Android Clipboard components and mechanisms sharing (Smith et al., (2013)). "2 new studies have more investigated code abuse in Android applications" (Fratantonio et al., (2013); Lee et al., (2013)). Other work includes finding vulnerabilities in flawed Android designs, such as research conducted by Huang et al., (2015) exploiting weaknesses in Android system servers to mount multiple DoS attacks and Qian et al., (2015) research uncovering Android root providers and shows that this well-engineered exploit is no good protected, and can be extremely dangerous if exploited.

GUI security

GUI guarantee has been studied widely in the context of the Android OS with its unique design and GUI subsystem. It has been proven that the confidentiality of the Android GUI can be breached by embedding UI components from malicious sources (Yinet al., (2011); Roesner&Kohno et al., (2013)), via the ADB command to take screenshots without the user's knowledge (Wang et al., (2014)), via other side channels such as shared memory (Mao et al., (2014)), or reading device sensors (Balakrishnan et al., (2012)). Recently, research conducted by Liu et al., (2015) conducted an in-depth evaluation of the system security of Android multitasking and ActivityManagerService design and found a wide-open attack surface that allows confusing the user about the displayed UI and threatens its confidentiality.

Android security model and attribute reference

Various Android components (application or its internal activities, services, value operators, acceptancor) are connected to each other the references become dangling, that can have serious security implications. Other categorically, an application can define permissions for different part and only can run messages requests from those who have acceptance. Apps that want to get these permissions must seek user approval. However, when the party that specified the permissions doesn't stay on a particular history, liscense preservation hangs: whoever assigned the permissions could be quitely boost privileges to entry guarded application part.

Opposing models

Scenarios, that a malware application has been installed on the object instrument in this research, are considered. Nevertheless, the application doesn't use to have any careful acceptance. Literally, in fact suspended liscense safeguard, it can decribe itself the lost acceptance to releas all type of invasion. To transmit the stolen information by the device, the application requires conversation capabilities. This could be done absolutely by inquisitive for system liscense, that almost all apps already ask for. "Alternatively, malicious applications can harness different lines, such as browsers, to transmit data" (Brodley et al., (2013)).

Exploit Hares

As noted previous, a hanging aspect mention can be an "ICC-call" to a missing bundling, action, assistance (that can be determined essentially by an activity filter) or content operator authority. In the existence of anologous references, a malware application claiming its objected attributes can developed entry to information sources discovered or protected by permissions. Another categorically, if the mention is unmaintained forward the execution line demanding Hare, malicious software that bring in the characterize automatically gains the privileges idnetial bythe aspect and is titled to receive perceptive letter by the operator.

It's important to note that not all hanging references are exploitable. It can be guaranteed by authenticating the presence of the package that is supposed to describe it and authecticating this contribution "FLAG_SYSTEM", or by countering the current instrument method, or other setting (e.g. getProperty). The existence of that safeguards was consistent in this research over automated code investigation. On the other hand, if guarantee checks are not performed, Hare becomes vulnerable to exploitation, although it is still difficult to find conditions to trigger the code. In this study, several ninety-seven Android insudtry images by extensive instrument manufacturers as "Google", "Samsung", "LG", "HTC", and "Motorola" were analyzed systematically and get 21.557 hanging aspects mention that are expected to be accessible. To recognize the safety hazard they may posture, an end-to-end attack was created on multiple Hare instances. For a minor percentage which found phisically, that boosted the perfect study, mostly Hares, especially those in pre-installed applications, were identified naturally adopting Hare hunter.

Ghost in Samsung Galaxy

Samsung Task Manager is a system application that offers convenient memory management to users. Through this service, one can monitor which applications are currently running on the device. In this study, the Task Manager was analyzed and it was found that it does not display the applications on the white list. Interestingly, the particular application is orderly this bundle name and there is no signature verification to check its authenticity. Also, many of them are missing on various devices. The consequence here is that an adversary can build malware that exploits those hanging references, using the official app name to ensure that the app won't attract attention while operating, for example, recording phone conversations in the background. Some ghost apps are implemented in Samsung Note 8.0 and removed from Task manager. Once again this deliberately installed attack app masquerades as a logger, bypassing Appstore security checks.

Fake "Drop-box" smartphone LG

"LG FileManager" is a system application on LG smartphone devices that support users to supervise their file that help to use "Dropbox", that can be opened by clicking on the button with the "Dropbox" icon. Attractively, in android "LG G3" factory images, the analyzers in this study get that the button in 1st test to release a process inside "com.vcast.manager", and only goes to the Dropbox web login page after an unsuccessful attempt. The program logic can be tailored to devices distributed by Verizon but leave references to device-dependent service with other carriers and development phones. In this study, the attack application was created to simulated "com.vcast.manager" and hijack the activity indicated by hanging references. Because "LG FileManager" does not review the target application's signature before starting its activity, "LG FileManager" blindly runs this application every. This allows the app to display fake Dropbox login activity to steal user confidental.

Changing the Proper Registrar

"S-Voice" appliance voice recording using the default recorder. There are 2 recorders, "com.sec.android.app.voicerecorder" and "com.sec.android.app.voicenote". What happens is "S-Voice" 1st tries to use the voice recorder activity and only if this fails (the application doesn't exist) it switches to voice note. Again, like a 2-choose-1 process does not implicate proper target verification. This allows building an attacker application that mimics a voice recorder application with the "VoiceRecorderMain" Action to handle the goal references. Experiments on Samsung Note 8.0, show that the rusher activity is always requested, even in the appearance of a voice note, that allows to record considerate user conversations or carry out phishing attacks.

Calling Intent Piracy

The amazing finding of this study is that the Hare fine value operator inside "Google Email" authorize a malware application to fully rechange this internal account appearance by malware action. In particular, Google Email, the standard email application on every Google phone, allows users to configure various email accounts (Gmail, exchange, etc.) through the Settings interface. To call this activity, the app sends an implicit Intent with the action android.intent.action.EDIT and the content "data://ui.email android.com/settings?account=x", x is the ID of the email account used to inform the setup activity account which email settings to edit. Both of these parameters are defined in the account settings activity "Intent" sift, ensuing code:

```
<!-- Account Settings Intent Filters-->
<activity
android:name=".activity.setup.AccountSettings" android:exported="true">
<intent-filter>
<action android:name="android. intent. action. EDIT"/>
<category android:name= "android. intent. category. DEFAULT"/>
<data android:scheme="content"
android:host="ui.email.android.com"
android:pathPrefix="/settings"/>
</intent-filter>
```

This implicit intent can be accepted by any application that defines the above Intent filter this hustle. Nevertheless, during this come, Android pops up screen listing all electable recipients to allow the user to vote. So protection can be circumvented, making malicious apps the sole eligible addressee. The data in end section of the "Intent filter" in the script is analyzed and examined how the "ActivityManagerSer" ("AMS" for short)

complete the “Intent” sent to this “Intent filter”. Figure 1 illustrates the Intent resolution steps in this scenario. If the data schema is content, AMS will try to infer the “MIME” (“Multi-Purpose Internet Mail Extension”) from the attached data to identify recipients that can handle this type: the data type here should be provided by the “ui.email.android.com” value operator. Moreover, this operator doesn't subsist and as n outcome, the type is usually ignored and the Intent is sent to whoever specified the action. EDIT and filter data (schema="content") with no MIME type specified.

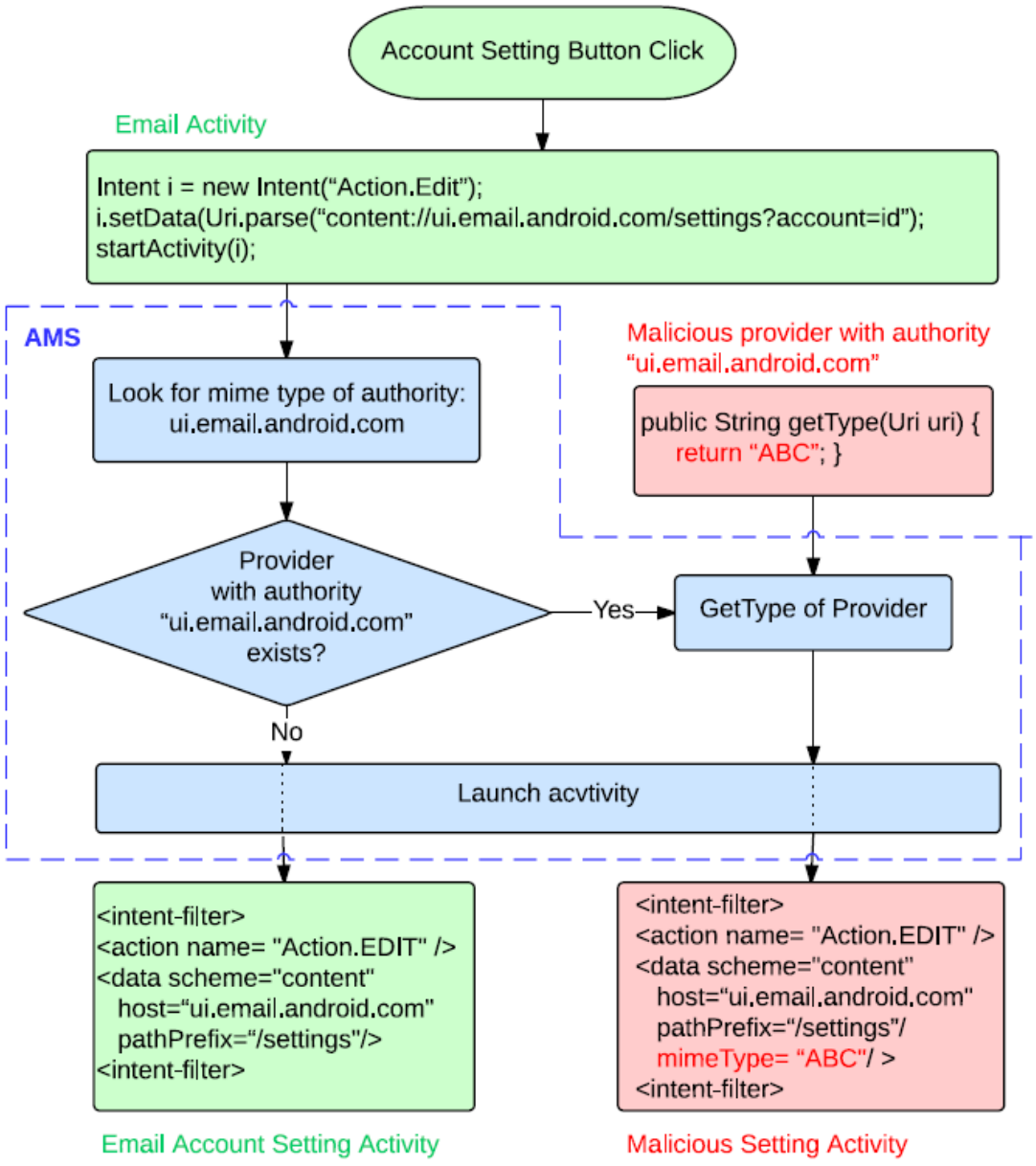

Figure 1. Leveraging “Hare Authority” to Hijack Email Account (Setup Activity)

The safety hazard here is a hint to a crashed value operator and can be utilized by a malware application that defines that provider. What the malicious software can do is name the provider authority “ui.email.android.com” to accept queries from AMS, give back a “MIME” type of its own choice to provide false information, and meanwhile specify this type in its activity's Intent filter itself, making it the only app eligible for Intents (to perform account setup activities). In this study, the built attack application

takes through the value operator and responds to queries from AMS with the MIME type “vnd.android.cursor.dir/vnd.example.ABC”. Additionally, the attacker defines an Intent filter as serve in the next code snippet, by claiming a mineType with the type that AMS tells it.

“<!--Malicious Setting Activity Intent Filters-->”
“<activity android:name=”. MaliciousSetting">
<intent-filter>
<action android:name="android. intent. action. EDIT"/>
<category android:name= "android. intent. category. DEFAULT"/>
<data android:scheme="content"
android:host="ui.email.android.com"
android:pathPrefix="/settings"
android:mimeType= "vnd.android.cursor.dir/vnd.example.ABC"/>
</intent-filter>

In this way, the Intent of the app only points to the malware, directing the user to a malicious process that would allow to enter his password.

“Tango in the dark”

“Tango” is a popular through-platform messaging app, that offers audio, and Vcalls in any signal include LTE, and Wi-Fi networks. This application has been installed more than 100 million times from Google Play. To display received “SMS” messages, set up an Intent filter with the “android.provider.Telephony.SMS_RECEIVED” action to get the Intent that carries the message from the “TelephonyManager”. When a user sends a message via Tango, the app saves it to sms, the phone's value operator. On devices without a Phone, Tango's reference to its value operator hangs. Therefore, a malware application can determine the value operator is using the authority sms to obtain user-sent “SMS messages”. This can happen the first time the malicious software sends the “SMS”, causing the unintentional user to reply. What can be exploited here is another vulnerability in Tango: the app doesn't secured SMS recipients with “android.permission.broadcast_sms” system permission, as it should. This allows. It’s enable any party broadcasting the “SMS_RECEIVED” action to inject fake short messages into the application. In this study, the attack was implemented on Samsung Tab 8.4, sending fake messages to “Tango” and receiving user responses using a malware value operator.

Scam on Smartphone LG “CloudHub”

“LG CloudHub” is a system application that enable to handle the cloud accounts, uploading data to the cloud, and accessing it by various instrument. The app supports “Dropbox” and “Box”, and on various devices, it can also connect users to other services, including LG's cloud provider. Information about these additional services is stored in the value operator “com.lge.lgaccount.provider”, which “LG CloudHub” looks up every time it's called. Interestingly, on some phones, this provider is missing. A prominent example is smartphone “LG G3”. When this happens, LG CloudHub only shows the default services, Dropbox and Box. However, this refers to the value operator as a case of Hare and exposes it to malware app manipulation. Specifically, an attack application that defines the “com.lge.lgaccount.provider” is implemented and placed in the entry for the LG Cloud account in the value operator. This account is then shown in the list of available LG CloudHub accounts. Once clicked by the user, the app sends an implicit Intent with the action “com.lge.lgaccount.action.ADD_ACCOUNT”. On the device (G3), there is no pre-installed app that defines the action, which allows the malicious software to define

the action, claiming it can handle Intents. The effect is that user's click on a system app triggers malware activity disguised as a login page for an LG Cloud account, which is used to trick users into revealing passwords and other credentials.

Confiscation of Permits

Hare defects can be caused by acceptance, that describe by network applications and used to oversight the access to different system or application-establish assets (e.g., value operator s, broadcast receivers, etc.). While the operation system personalization activities, applications that determine liscense (their "original" owners) may be removed. Meanwhile, if the resource protected by this permission still exists, the use of the permission (for protection) becomes a dangling reference. To exploit such a weakness, adversaries can easily pinpoint missing permissions but still use them to developed the access to the assets that was protecting. This problem was also found to be widespread in this study and appeared in all ninety-seven scanned factory images. Making this threat especially insidious is the fact that Google Play doesn't review double permissions: all the rush apps made here were successfully uploaded there.

Get contacts from S-Voice

The S-Voice system application cinsist of a value operator ("com.vlingo.midas.contacts.value operator") that stores information about your contacts, such as name, email address, phone number and home address. Access to the operator is preserved. By the pair "com.vlingo.midas.contacts.permission.READ" ("READ") and "com.vlingo.midas.contacts.permission.WRITE" ("WRITE"). Nevertheless, it turned out that the Samsung Galaxy Note three and Samsung Tab Galaxy Note eight were not specified, opening the door for exploitation. In particular, the attack application, in this study he was created for two devices and determines "READ and "WRITE" permissions. Applications are known to successfully read all contact details from "S-Voice" and freely improve data maintained by value operators. Modifying contact email addresses, URLs, and phone numbers can lead to information disclosure and other consequences (for example, if a user enters a hostile URL placed in a friend's access, etc.).

Designing

The idea behind this method is very gentle. Different research is first performed for different industry images. Explicitly extract all attributes defined by preinstalled applications and all hints to aspects in their scripts, Compare the citation with the definition. The difference in 2 endings represent the potential appearance of rabbits. For example, if a package name is used to start an activity ("startActivity") or bind a service ("bindService"), but is not owned by any app pre-installed on the device, references to that Hares name will be there is likely to be. On the other hand, such clues are surprisingly well hidden. For example, before referencing a package, a system application first checks for this, collects its signature information (such as "getPackageInfo" with the "GET_SIGNATUREflag"), and compares it with the original application's signature. Identifying truly accessible rabbits requires analyzing code between potential guards (e.g. signature verification functions) and examining possible dangling references (e.g. "startActivity") for relevance. To implement this idea, the system is designed with three main components: Pre-Processor, "Differ", and "GuardCatcher", as shown in Figure 2. A preprocessor extracts an application package from an operating system image and transforms it into a format can be investigated in subsequent steps. "Differ" conducts branch surveys and reports possible dangling estimates. Catcher looks at her APK that contains references to determine if the references are protected.

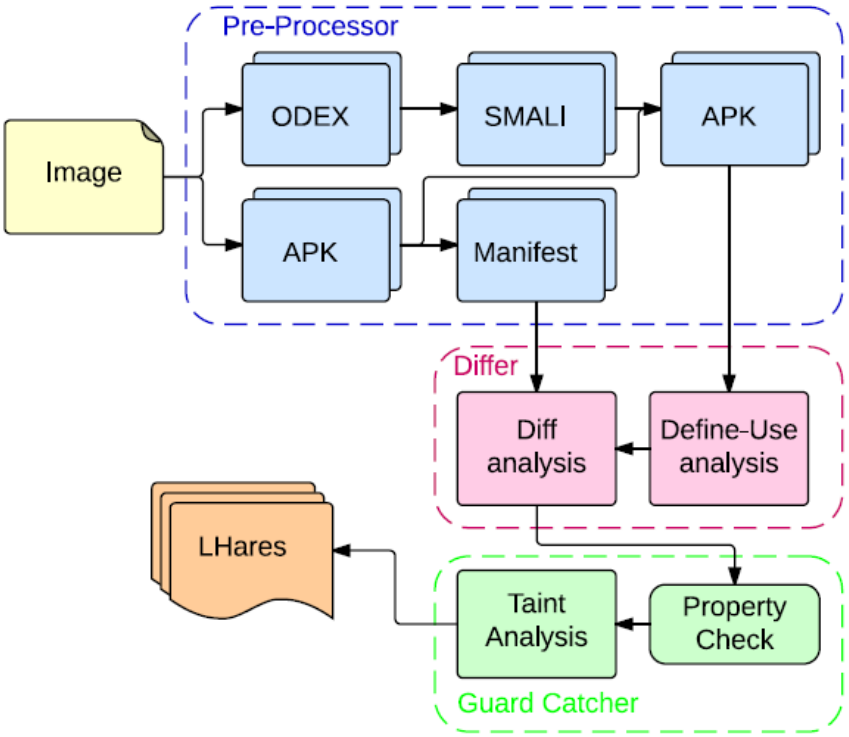

Figure 2.: Harehunter Design.

Pre-processing

By different manufactory image, firstly "Harehunter" raise all of the to pre-installed application, form "APK" & "ODEX" files, and process to "Apktool" to concentrate different applications obvious file to gather the apps into "Smali" code. Several instrument especially Samsung, the application's system "ODEX" files are frequently separated by their "APK" files, to increase their loading times, during the "Flowdroid", the passive authority where this build network exists in this study only run on "APK". To solve this problem, before processor is armed to extract the "ODEX" files automatically, decompile them, then recompile and compress them, along with their asset to current "APK file". More complexing this process is that for Android with five point zero Lollipop type, the "ODEX" files are changed by the O-AT files, that consist of origina code. This apps as forms, firstly, "Harehunter" unzips this O-AT file and then runs "oat2dex" to convert it to the "ODEX" format, allowing the above process to move forward.

Divergent investigation

To execute various investigation, firstly revamp exploration all cited and unabigous files for the definition of the focus attribute. Processing an "XML" resolver, this method can be easyly gather detailed packages, behavior, and gratified provider advisors and liscense by single app unabiguous files. Note that all of these aspects can only be specified uniquely, except for actions that receive broadcast messages. The performs usage definition analysis of each calling page to find the Control Progress generated by Flowdroid. Restore the attribute names of interest using Graph (CFG). The problem here is that some program entry points like onHandleIntent are missing, so Flowdroid can't render a complete CFG. In this implementation, as many discoverable entries as possible are added back, but there is still some target function calls that Flowdroid cannot make associated CFGs with. For this call, the current prototype can only handle situations where the attribute name is hardcoded in the associated function.

Shield Identification

Quotation to loose attributes are regularly secured. There are two basic ways of protection, signature shield and feature shield. The appearance of authentic packages can also confirm the correctness of action and activity names. Another way to secure this attribute is to review the new develop method of the device, as only several that have specific appearances (in terms of bundle, content operator, and so on): for example, input mechanism, and email of application, they can all differ from build to building; The SMS/MMS provider might not even be on the tablet.

To identify like protections, "Guard-Catcher" performs stigma analysis over application data progress and control progress, adopting the functionality served by "Flowdroid". In particular, this method 1st recognize series of shield appointment as well as "hasSystemFeature" and "getPackageInfo" by the "GET_SIGNATURES" parameters and then tries to build up the relationship of them and the hanging hints found by divergent investigation, the importance conditions for that hint to be safe. For this aim, the output of this guard is specified as the taint source, and references are marked as stigma decline. "Flowdroid" is processing to resolve if a stain could be raised by the first to the last. For a sink that can't be soiled, it's most likely reported as Hares.

```
public boolean extendAccessToken(Context context, ServiceListener
    servicelistener){
   Intent intent = new Intent();
   try{
  PackageInfo pi = context.getPackageManager().getPackageInfo
      ("com.facebook.katana", PackageManager.GET_SIGNATURES);
  // Compare signature to the legitimate Facebook
  // app Signature
  if (!compareSignatures (pi.signatures[0].toByteArray())){
     return false;
  } else{
     intent.setClassName("com.facebook.katana", "com.facebook.katana.platform.
         TokenRefreshService");
     return context.bindService(intent, new
         TokenRefreshServiceConnection(context, servicelistener), 1);}
   }catch(PackageManager.NameNotFoundException e){
  return false;
   }
}
```

Figure 3.: Example of a Signature Based Guard

```
private void ViewVideo(Uri uri){
   Intent intent = new Intent("android.intent.action.VIEW", uri);
   if (getPackageManager().hasSystemFeature ("com.google.android.tv")){
      intent.setPackage("com.google.android.youtube.googletv");
   } else{
      intent.setPackage("com.google.android.youtube");
   } startActivity(intent);
}
```

Figure 4.: Example of a Feature-Based Guard

Processing full stigma investigation for each shield and hint set will be rally low. To create a shield identification more extensible, “Catcher” employs a multi-step combined approach, combining rapid property inspections by stain investigation. In particular, it first checks whether the corresponding source and sink are in the same mechanism. In the vast majority of facts, they are linked and therefore the hint is treated secured. If not, the next approach analyzing the bundle names convoluted in the signature review by those used for hint. Compatibility found between partners almost always express a protective relationship. For example: “com.facebook.katana” (fig. 3) that appears inside getPackageInfo and setClassName. Only when both tests fail, heavyweight stain investigation will be adopted. In this study's wide-field investigation of manufarturies drawings, it was get the most of the time, the shielding for hint can be found in 2 steps.

EVALUATION

The evaluation of implementation effectiveness in this dimension study was conducted, involving Operation System images for niety-seven popular devices, all together in more than 24,000 system applications. “Harehunter” informed 21557 possible Hares. From all these Hares, a random sample of 250 was taken and the code was analyzed manually. Only thirty-seven, that is fourteen percent, that can catch to be wrong detections: incorrectly warning the guarded hint as Hare. The Guard Catcher's false negative rate was then measured by randomly examining possible hanging quotation informed by the Differ and measuring the outcome by those detected by the “Catcher”. In all two hundred and fifty samples, fourty six or nineteen percent were lost by this application. “Flowdroid is known to be problematic in handling ICC” (Octeau et al., (2014)) and other problems such as lost entry points and call chart is not completed. While this occurs, stain investigation can’tbe performed.

RESULTS

Great Balance Mastering

To figure out the field and significance of the safety hazard caused by Hares, a big-proportion analysis research on ninety-seven industry images was carried out in this study. The research prove that “Hares” actually widespread, by an important effect on the Android enviironment.

OS Image Collection

In this study, a total of 97 factory images were found in “Samsung Update”, “Android Revolution”, and “physical devices”, that includes approximately one hundred and eighty-three applications per image and 24,185 applications in total. All of that images are personalized for fourty nine various samartphone android or android tab models, in thirty-six countries, and twennty-three various shipper. All of that operate Android versions from Adnroid type 4.0.3 to Android type 5.0.2.

Table 1: Set of Android Images

Vendor	Images count	System apps count	Avg # of System apps per Image	Countries count	Carriers count	OS versions count
Vendor A	83	21733	261	36	23	10
Vendor B	7	1561	223	1	1	4
Vendor C	1	174	174	1	1	1
Vendor D	4	398	99	1	1	3
Vendor E	2	319	159	2	1	2
Total	97	24185	183	36	23	10

Landscape

On investigating those manufacture of images, it was got around thirteen percent could not be decompiled by "Apktool" or investigated by "Flowdroid". By that investigation, "Harehunter" found 21,557 weakness (unattended hanging hint) in 3,450 accessible applications. Several weaknesses may occur more than once inside the same application, and some accessible applications appear on several devices. This study revealed that each image consist-of big amount of Hare weakness, rating from eight to five hundred and ninety eight. In regularly fourteen-point three percent of application preinstalled in "4.X" and eleven-point seven percent in "5.X" that accessible (detailed in table 2).

Table 2. Prevalence of Hares in System Applications per Vendor

Vendor	Hares in Android 4.X		Hares in Android 5.X		Avg Hares per Device	Min Hares per Device	Max Hares per Device
	Hares count	Vulnerable apps count	Hares count	Vulnerable apps count			
Vendor A	19279	3045 (18%)	608	99 (6%)	239	23	598
Vendor B	679	121 (13.3%)	425	85 (15.5%)	157	100	224
Vendor C	N/A	N/A	248	33 (21.5%)	241	248	248
Vendor D	107	31 (12.4%)	8	5 (5%)	29	8	45
Vendor E	187	23 (15.6%)	16	8 (12.1%)	101	16	187
Total	20252	3220 (14.3%)	1305	230 (11.7%)	153	8	598

As can be seen from the table above, the problem extends over eacg device builder: among tge "VendorsA and C" have a important percentage of their system applications compelling hanging hint. In correlation, Vendor D has the minimum total of weakness is twenty-nine and the minimum ratio of broken applications (8%). The probable reason is that the OS image the device is running on is the least personalized, that smallest chances of introducing Hares. Figure 6 illustrates the identification of defects in different categories of Hares. Most of the problems stem from undefined action names. In analogy, a relatively minimum proportion of acceptance was catch to be ramified in dangling hints.

Hares Impact

The impact of Hares is very important. In other to the end-to-end invasion made in this research, a total of 33 samples of weaknesses were also taken randomly and manually analyzed what could happen after being harnessed. In consequence of the deficiency of a big amount of physical devices, all that can be done is static analysis to deduce the possible consequences after a successful exploit. This kind of investigation might be inaccurate, but it is still importance to understand the effect of this kind of security weakness that has not been note previously. The outcome of this investigation is in Table 3.

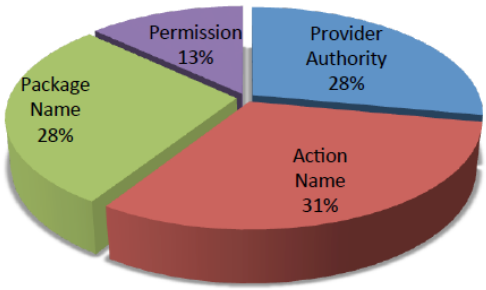

Figure 5: Distribution of Hares in Different Hare Categories

As can be seen here, Four Hares can root content leaks (browser notes and bookmarks) after the malicious software masquerades as an unspecified value operator, where the victim app feeds data into it. 4 examples might reveal the user's confidental

information wthen the hanging bundle name is hijacked. Specifically, it on Samsung Note 8.0, hanging references involve an accurate Intent being passed to a packet that doesn't exist. The intent includes a content “URI” that points to confidental data (for example, a photo) as well as the “FLAG_GRANT_URI_PERMIS SION” permission, which allows the beneficiary to read the data with no asking for permission. As a outcome, illegal applications adopting the object of bundling name can develop access (data).

Also, on the LG G3 type, hanging references to non-existent value operators maybe open access for adversaries to pinpoint that operator to infect data synced to another users' devices. Furthermore, this investigate announce three examples that can start to a denial-of-service invasion when an adversary creates an undefined value operator used by the victim application and sets the expressed emblem to fake. By the application script, this invasion can aim a safety omission while the casually application tries to read or write to this provider. Some other Hares can start to unforeseen situations: an application with a certain package name won't appear in the system's Task Manager and another application on the LG G3 type can't be enforced to quit by the LG Settings application and catch that Hares in three applications may only show notifications. There are six bunnies similar to the lost service with that function can't be figured out. Lastly, the entry point for 4 Hares was not found which could be dead code.

Responsible party

For this purpose, 6 images were examined. The proportion of “Hare” defects typically introduced by this model spacing by nine percent to twenty-nine percent. Next, the images were grouped into subgroups and examined which showed the best proportion of familiar Hare facts. The android tab model had the top proportition of frequent Hares at 63%, while the mobile model had the second highest percentage of common Hares at 56%. The familiar Hare case in android tab and android smartphone models is mostly thirty-eight percent. Adapting an operation system to a Android tab model or a android smartphone model propose a lot of Hares. Meanwhile, the weaknesses catch on the same model adjusted for various operators were also compared.

Table 3. Potentially Effect of thirty-three Randomly Picked Hares

Impact	Hare Category	# of Hares
Activity Hijacking	Package and Activity Name	3
Activity Hijacking	Action Name	2
Activity Hijacking	Provider Authority	1
Data Leakage	Provider Authority	4
Data Leakage	Package and Activity Name	1
Data Pollution	Provider Authority	1
D.O.S.	Provider Authority	3
Dialog Popup	Action Name of Activities	3
Others	Package Name	5
Impact Not Clear	Action Name of Services	6
Maybe Dead Code	All Categories	4

Table 4. "Hare" Defects on Different Models of Hawker Processing on Android 4.4.2

Model	# of New Hares Introduced by Model
Tablet 1	106 (27%)
Phone 2	35 (21%)
Phone 3	75 (29%)
Tablet 4	57 (22%)
Tablet 5	22 (9%)
Tablet 6	72 (20%)

The image of Phone 3, its adjustment on 6 carriers results in around a 3% to 20% shortfall. Neither the manufacturer nor the carrier causes Hare's weakness. However, the former seems to have to be more responsible than the latter. Also, most of the Hares will most likely be popularized while operation system customization for various instrumen models (phones or tablets).

Tendency

Figure 6 more analyze the ratio of acessible apps to various operating systems versions over production in industry. For VendorB, the trend is close fixed: the ratio is fourteen point three percent in 4.2.2 and fiveteen point one percent in 5.0.1. Moreover, all of this devices, the Hare risk remains necessary indicating that manufacturers are not yet aware of the extent of this kind of vulnerability.

CONCLUSION

"HareGuard" application has proven to be potentially in discovering all rush applications made intentionally. Further evaluation of performance was also carried out and minimum system host impact: the scanner was found to use only four point twenty-nine megabyte of memory and use zero-point twenty nine percent CPU during scanning the application obvious. Hares aren't just some random, isolated bug introduced by implementation lapses. The appearance of these weakness defined weaknesses in the design philosophy of Android and its ecosystem. Fundamentally, Android is a c sysomplicated sytem, that parts and applications are signify to run together, leading to very complex interdependence relationships between them. Meanwhile, the Android environment is notable for being highly different and decentralized: different Operation Systems version is personalized and re-personalized by different parties fairly fast and used by whoever that can develop an applicaton for that version.

In the lack of more guidelines and appropriate execution method, hanging quotation make shunless. As this study proves, Hares are widespread, present in every device examined, and are also critical for security, compromising sensitive user data and moreover the correct implementation of system applications. While not all of the problems informed by "Harehunter" are usable, that depend on situation to run suspectible code, the pervasiveness such like exposed code is worrisome: without a thorough examination of individual cases, there's no telling if they're exploitable down the specific terms, reputable to incidental effect.

To eliminate Hare risk, it is significance to have well-documented intersuspended relationships and make them open to aspects relevan in operational system customization and application expansion. In addition, there should be a policy requiring that anyone that transform the operational system or developing applications must not create dangling relationships as well as pointing to missing attributes, and mechanisms for policy compliance checks. Policy enforcement here can take advantage of the available Android

compatibility schedule, that is presently unable to perform security checks. The suspect section is the accumulation of interdependent relationships for all known versions of Android. There is no such information yet. This research shows that manufacturers seem to be oblivious to the linkages in their own devices. A systematic equipment, such as Harehunter, is needed to recognize this information.

While efforts should be made to safe every character hint, the most important thing here is obvious authentication rather hints. It's too regularly seen that quotation are only implicitly secured: for example, references to system apps are secured by the app's presence on the device, that except another apps adopting the same bundle name. Like security is fragile, literally broken the application is changed during the operation system is customized for a current device model. Other that, the secured checks can be more complex than they appear. More specifically, though, package name references can be directly safeguarded by signature checks. Another aspects like content operator and process. usable immediately and their appearance on a particular instrument is regularly verifable by countering the present instrument role and another appearance. Correctness of checks, depends on recognition of component/application relationships across various versions, models, etc., which Harehunter and other similar tools need to restore.

System protection legacy

Before even thinking about how to wipe out Hare in expanding next method and applications, as this study shows, are full of all type of Hare weaknesses. The method builded, “Harehunter” and “HareGuard”, made the early action towards identifying and protecting that vulnerabilities. In particular, as previously claimed, “Harehunter” also can manipulated an important model in meeting interdependent relationships to support safety current system and applications. By this big potential, the current implementation is still in its infancy: it introduces about fourteen percent error positive and misses nineteen percent of really susceptible facts in this research. It's imaginable that “Harehunter” would become more potential once more competent analyzers were used. In additionan instrumen identical to Harehunter, but processing with the source code, can more accurately detect Hare's weaknesses.

BIBLIOGRAPHY

A. Al-Haiqi, M. Ismail, and R. Nordin, "On the best sensor for keystrokes inference attack on android," in The 4th International Conference on Electrical Engineering and Informatics (ICEEI), Procedia Technology, 2013.

AP Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11, (New York, NY, USA), pp. 627–638, ACM, 2011.

B. Liu, J. Lin, and N. Sadeh, “Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help?,” in Proceedings of the 23rd International Conference on World Wide Web, WWW'14, (New York, NY, USA), pp. 201–212, ACM, 2014.

C. Lin, H. Li, X. Zhou, and X. Wang, “Screenmilker: How to milk your android screen for secrets,” in 21st Annual Network and Distributed System Security Symposium (NDSS), The Internet Society, 2014.

E. Miluzzo, A. Varshavsky, S. Balakrishnan, and RR Choudhury, “Tapprints: Your finger taps have fingerprints,” in Proceedings of the 10th International Conference on

Mobile Systems, Applications, and Services, MobiSys '12, (New York, NY, USA) pp. 323–336, ACM, 2012.

F. Roesner and T. Kohno, “Securing embedded user interfaces: Android and beyond,” in Proceedings of the 22nd USENIX Conference on Security, SEC'13, (Berkeley, CA, USA), pp. 97–112, USENIX Association, 2013.

F. Roesner, T. Kohno, A. Moshchuk, B. Parno, HJ Wang, and C. Cowan, “User-driven access control: Rethinking permission granting in modern operating systems,” in Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012.

H. Huang, S. Zhu, K. Chen, and P. Liu, “From system services freezing to system server shutdown in android: All you need is a loop in an app,” in Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, (New York, NY, USA), pp. 1236–1247, ACM, 2015.

H. Zhang, D. She, and Z. Qian, “Android root and its providers: A double-edged sword,” in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, (New York, NY, USA), pp. 1093–1104, ACM, 2015.

J. Caballero, G. Grieco, M. Marron, and A. Nappa, “ Undangle: Early detection of dangling pointers in use-after-free and double-free vulnerabilities,” in Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISTA 2012, ACM, 2012.

KWY Au, YF Zhou, Z. Huang, and D. Lie, “Pscout: Analyzing the android permission specification,” in Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, (New York, NY, USA), pp. 217–228, ACM, 2012.

L. Li, A. Bartel, J. Klein, YL Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, “I know what leaked in your pocket: uncovering privacy leaks on android apps with static taint analysis,” arXiv preprint arXiv:1404.7431, 2014.

L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, “The impact of vendor customizations on android security,” in Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS '13, (New York, NY, USA), pp. 623–634, ACM, 2013.

M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An empirical study of cryptographic misuse in android applications,” in Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS '13, (New York, NY, USA), pp. 73–84, ACM, 2013.

M. Zhang, Y. Duan, Q. Feng, and H. Yin, “Towards automatic generation of security-centric descriptions for android apps,” in Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, (New York, NY, USA), pp. 518–529, ACM, 2015.

P. Brodley and leviathan Security Group, “Zero Permission Android Applications.” https://www.leviathansecurity.com/blog/zero-permission-android-applications/ . Accessed: 10/02/2013.

P. Ratazzi, Y. Aafer, A. Ahlawat, H. Hao, Y. Wang, and W. Du, “A systematic security evaluation of Android's multi-user framework,” in Mobile Security Technologies (MoST) 2014, MoST'14, (San Jose, CA, USA), May 17, 2014.

P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, “Android permissions remystified: A field study on contextual integrity,” in Proceedings of the 24th USENIX Conference on Security Symposium, SEC '15, (Berkeley, CA, USA), pp. 499–514, USENIX Association, 2015.

QA Chen, Z. Qian, and ZM Mao, “Peeking into your app without actually seeing it: Ui state inference and novel android attacks,” in Proceedings of the 23rd USENIX Conference on Security Symposium, SEC'14, (Berkeley, CA, USA), pp. 1037–1052, USENIX Association, 2014.

R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “Whyper: Towards automated risk assessment of mobile applications,” in Proceedings of the 22nd USENIX Conference on Security, SEC'13, (Berkeley, CA, USA), pp. 527–542, USENIX Association, 2013.

R. Wang, L. Xing, X. Wang, and S. Chen, “Unauthorized origin crossing on mobile platforms: Threats and mitigation,” in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, (New York, NY, USA), pp. 635–646, ACM, 2013.

S. Fahl, M. Harbach, M. Oltrogge, T. Muders, and M. Smith, “Hey, you, get off of my clipboard,” in proceeding of the 17th International Conference on Financial Cryptography and Data Security, 2013.

SH Kim, D. Han, and DH Lee, “Predictability of android OpenSSL's pseudorandom number generator,” in Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS '13, (New York, NY, USA), pp. 659–68, ACM, 2013.

T. Li, X. Zhou, L. Xing, Y. Lee, M. Naveed, X. Wang, and X. Han, “Mayhem in the push clouds: Understanding and mitigating security hazards in mobile push-messaging services,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, (New York, NY, USA), pp. 978–989, ACM, 2014.

W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, “A study of android application security,” in Proceedings of the 20th USENIX conference on Security symposium, 2011.

X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and GN Peri, “Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, (New York, NY, USA), pp. 66–77, ACM, 2014.

X. Zhang, K. Ying, Y. Aafer, Z. Qiu, and W. Du, “Life after app uninstallation: Are the data still alive? data residue attacks on android,” in NDSS, 2016.

X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, CA Gunter, and K. Nahrstedt, “Identity, location, disease and more: Inferring your secrets from android public resources,” in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, (New York, NY, USA), pp. 1017–28, ACM, 2013.

Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith, “Sok: Lessons learned from android security research for appified software platforms,” in 37th IEEE Symposium on Security and Privacy (S&P '16), IEEE, 2016.

Y. Michalevsky, D. Boneh, and G. Nakibly, “Gyrophone: Recognizing speech from gyroscope signals,” in Proceedings of the 23rd USENIX Conference on Security Symposium, SEC'14, (Berkeley, CA, USA), pp. 1053–1067, USENIX Association, 2014.

Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh, “Powerspy: Location tracking using mobile device power analysis,” in 24th USENIX Security Symposium, 2015.

Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, “Autocog: Measuring the description-to-permission fidelity in android applications,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS'14, (New York, NY, USA), pp. 1354–1365, ACM, 2014