MAY 11-12

BRIEFINGS

Stealthy Sensitive Information Collection from Android Apps

Bai Guangdong@UQ, Zhang Qing@ByteDance, Xia Guangshuai@ ByteDance

About Us

Bai Guangdong

Associate professor

from UQ

Zhang Qing

Senior security and privacy expert

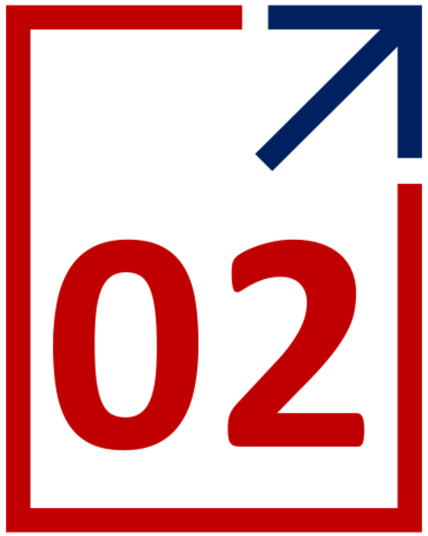
(CIPT/CIPP/FIP) from ByteDance

Xia Guangshuai

Security researcher from ByteDance

PRIVACY

MUCH HAS BEEN TALKED, BUT NOT MUCH DONE



Background

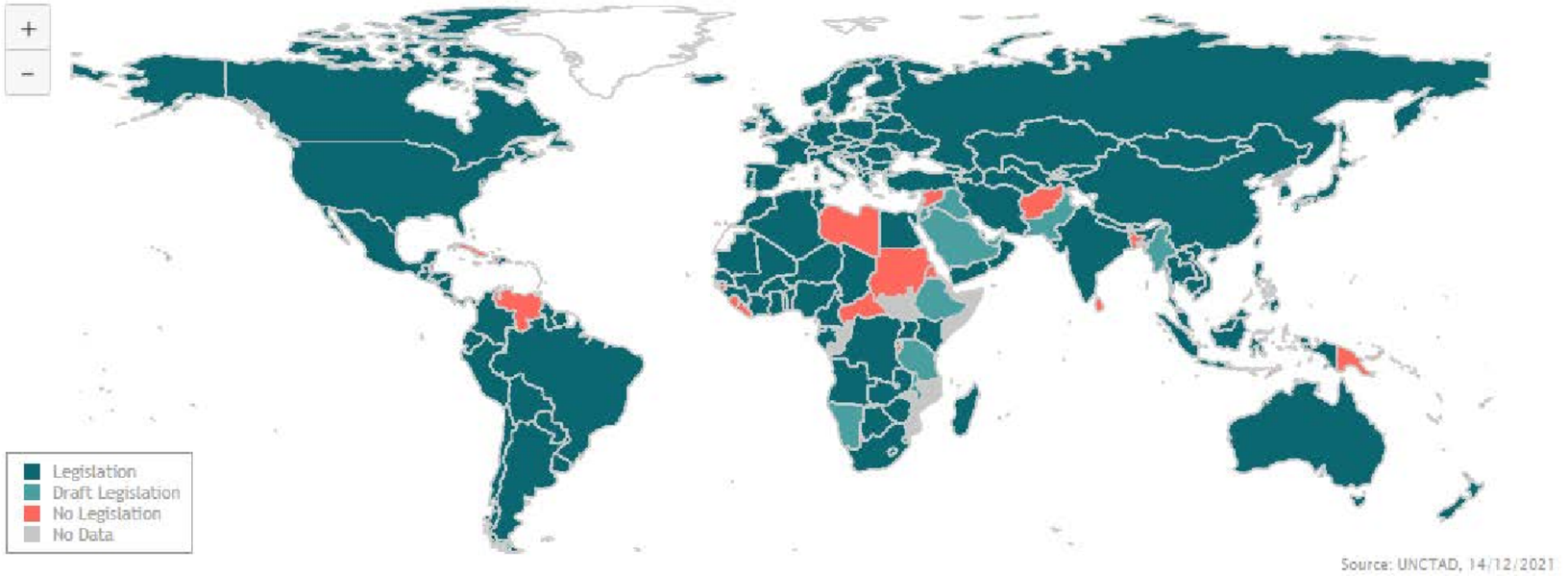
Data regulation is increasingly important

User data protection has gained **a great deal of attention** around the world.

Many countries have put in place **legislation** to regulate the collection and use of personal data, such as the well-known European Union (EU) General Data Protection Regulation (GDPR).

Infringements of user privacy could result in **large penalties**, e.g., “a fine of up to €20 million, or 4% of the firm’s worldwide annual revenue” set by GDPR.

Post-GDPR Era



71%	9%	15%	5%
COUNTRIES WITH **LEGISLATION**	COUNTRIES WITH **DRAFT LEGISLATION**	COUNTRIES WITH **NO LEGISLATION**	COUNTRIES WITH **NO DATA**

Source: https://unctad.org/page/data-protection-and-privacy-legislation-worldwide

Evolution of Android privacy data protection

Android 6	Android 9	Android 10	Android 11	Android 12	Android 13
Runtime Permissions	Restricted access to logs	MAC address randomization	Package visibility	Microphone and camera indicators	Runtime permission for notifications
Access Hardware Identifier (e.g., Wi-Fi/bluetooth MAC) needs LOCATION permission	Restricted access to phone numbers	Restriction on non-resettable device identifiers	Restrictions on /sdcard/Android/data	Permission package visibility	New runtime permission for nearby Wi-Fi devices
	Restricted access to Wi-Fi location and connection information	Restrictions on direct access to configured Wi-Fi networks	Add READ_PHONE_NUMBERS permission	Clipboard access notifications	Use of body sensors in the background requires new permission
		Some telephony, Bluetooth, Wi-Fi APIs require FINE location permission	Auto-reset permissions from unused apps	Add BLUETOOTH_SCAN, BLUETOOTH_ADVERTISE, and BLUETOOTH_CONNECT permissions	Permission required for advertising ID(GAID)
		Add ACCESS_BACKGROUND_LOCATION permission		Support restricting apps from obtaining advertising ID (GAID)	
		Protection of USB device serial number			

Android 6: Runtime permissions

Runtime permissions have been added since Android 6, and runtime permissions are required to obtain sensitive information such as device unique identifiers and location information, and use services such as Camera.

Runtime Permissions

This release introduces a new permissions model, where users can now directly manage app permissions at runtime. This model gives users improved visibility and control over permissions, while streamlining the installation and auto-update processes for app developers. Users can grant or revoke permissions individually for installed apps.

On your apps that target Android 6.0 (API level 23) or higher, make sure to check for and request permissions at runtime. To determine if your app has been granted a permission, call the new `checkSelfPermission()` method. To request a permission, call the new `requestPermissions()` method. Even if your app is not targeting Android 6.0 (API level 23), you should test your app under the new permissions model.

For details on supporting the new permissions model in your app, see Working with System Permissions. For tips on how to assess the impact on your app, see Permissions Usage Notes.

Android 10: Device unique identifier restriction

Starting from Android 10, Google restricts the acquisition of **device unique identifiers**, and apps can no longer obtain device unique identifiers such as **IMEI/SN/IMSI/ICCID**.

Restriction on non-resettable device identifiers

Starting in Android 10, apps must have the `READ_PRIVILEGED_PHONE_STATE` privileged permission in order to access the device's non-resettable identifiers, which include both IMEI and serial number.

Caution: Third-party apps installed from the Google Play Store cannot declare privileged permissions.

Affected methods include the following:

- `Build`
  - `getSerial()`
- `TelephonyManager`
  - `getImei()`
  - `getDeviceId()`
  - `getMeid()`
  - `getSimSerialNumber()`
  - `getSubscriberId()`

If your app doesn't have the permission and you try asking for information about non-resettable iden s anyway, the platform's response varies based on target SDK version:

- If your app targets Android 10 or higher, a `SecurityException` occurs.
- If your app targets Android 9 (API level 28) or lower, the method returns `null` or placeholder data if the app has the `READ_PHONE_STATE` permission. Otherwise, a `SecurityException` occurs.

Android 12: GAID restriction

Starting from Android 12, for **GAID** (**Google advertising ID**), users can prohibit the App from obtaining GAID through the limit tracking settings.

public String getId ()

Retrieves the advertising ID.

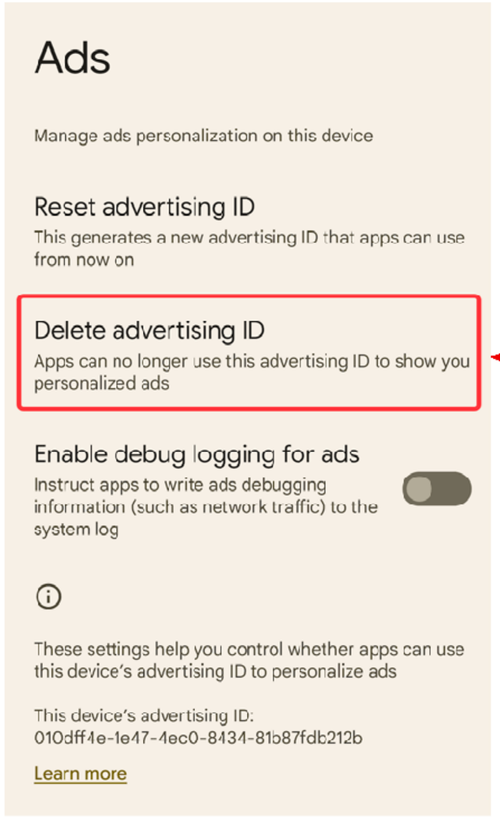
Starting from late 2021, on Android 12 devices, when `isLimitAdTrackingEnabled()` is `true`, the returned value of this API will be `00000000-0000-0000-0000-000000000000` regardless of the app's target SDK level.

In early 2022, this change will be applied to all the devices that support Google Play services.

Apps with target API level set to 33 (Android 13) or later must declare the normal permission `com.google.android.gms.permission.AD_ID` as below in the AndroidManifest.xml in order to use this API.

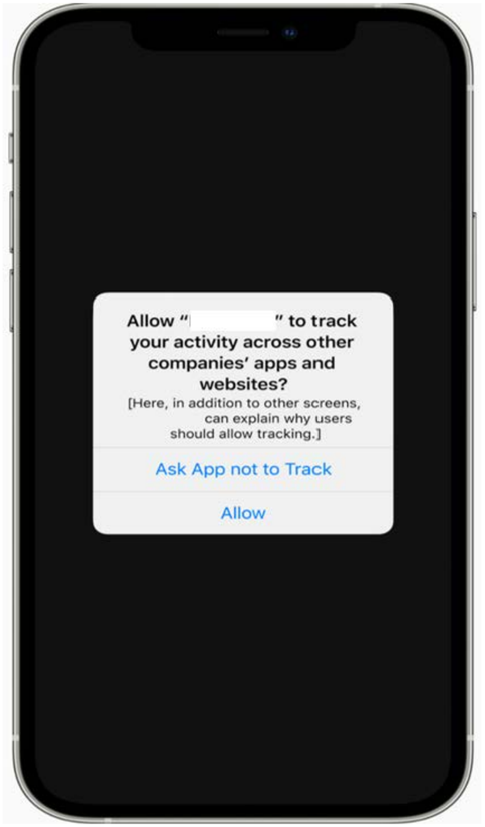
- This permission will be granted when the app is installed.
- If this permission is not declared, the returned value will be `00000000-0000-0000-0000-000000000000` starting early 2022.
- Until then, to help developers, a warning line is logged if the permission is missing when the app targets API level 33 (Android 13) or higher.
- This warning line is under the `Log` tag `AdvertisingIdSettings`.

```
<uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
```



iOS Identifier for Advertisers (IDFA)

BTW, for iOS, starting from iOS 14.5, if the app wants to obtain **IDFA** (equivalent to Android GAID), it must be **manually authorized by the user**.



Research questions

Are these measures adequate to protect user privacy?

Manually authorizing to obtain IDFA since iOS 14 has caused disputes, but, **is it a storm in a teacup**?

Research questions

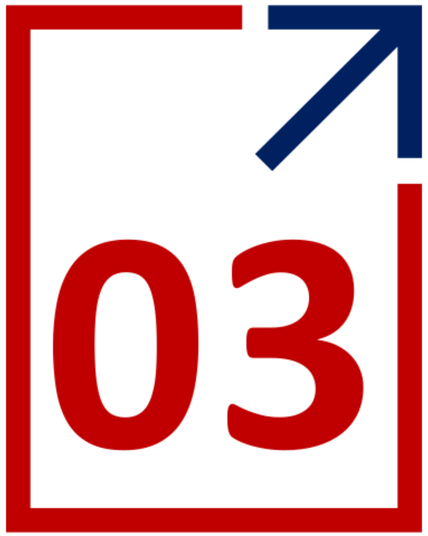
Are these measures adequate to protect user privacy?

Manually authorizing to obtain IDFA since iOS 14 has caused disputes, but, **is it a storm in a teacup**?

Explained: Why [redacted] thinks Apple's iOS 14 privacy push will have a severe impact on business

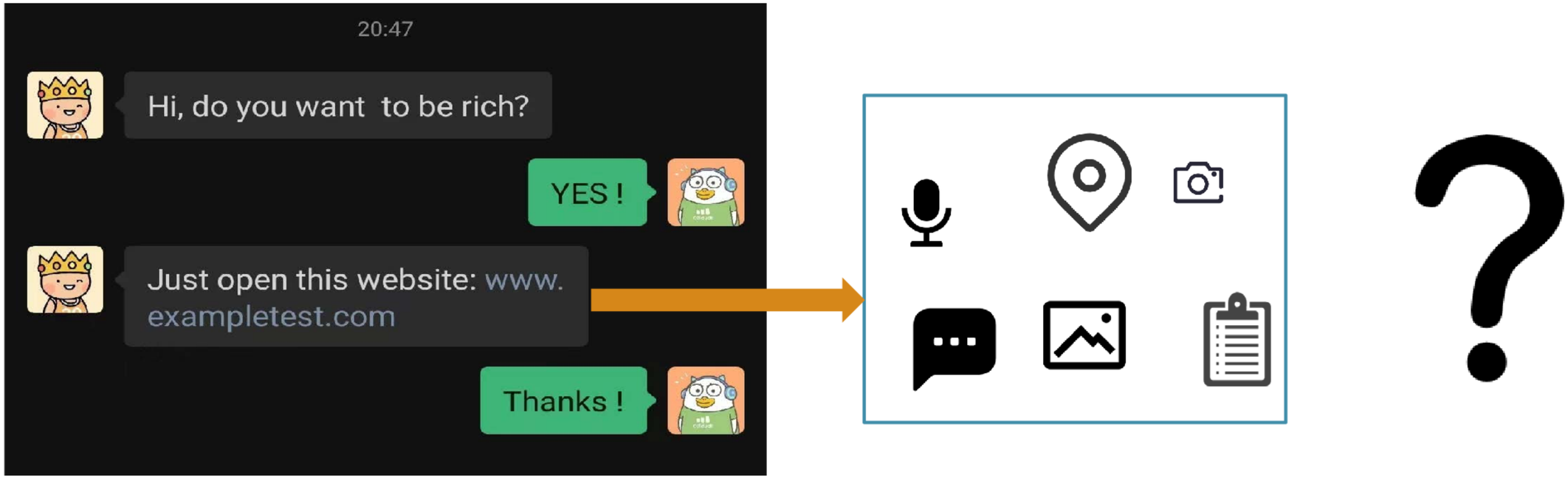
DIGITAL INFORMATION WORLD

[redacted] begins a ruckus against upcoming in-app changes to Apple's IDFA in iOS 14



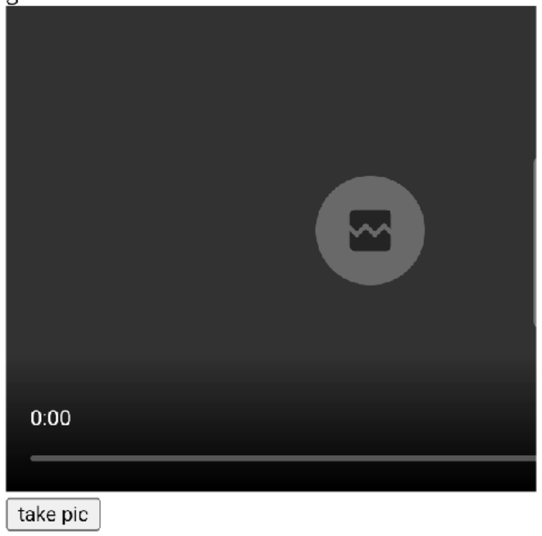
Our work

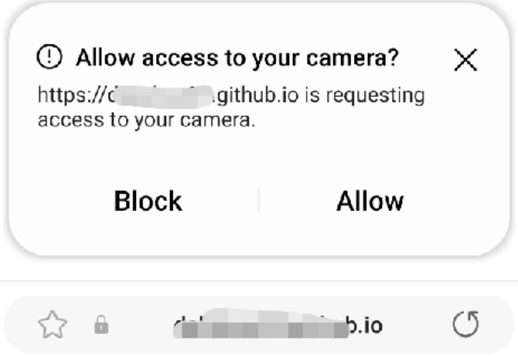
The Dangers of Unrestricted Access to Privacy Information Using Webview



The Dangers of Unrestricted Access to Privacy Information Using Webview

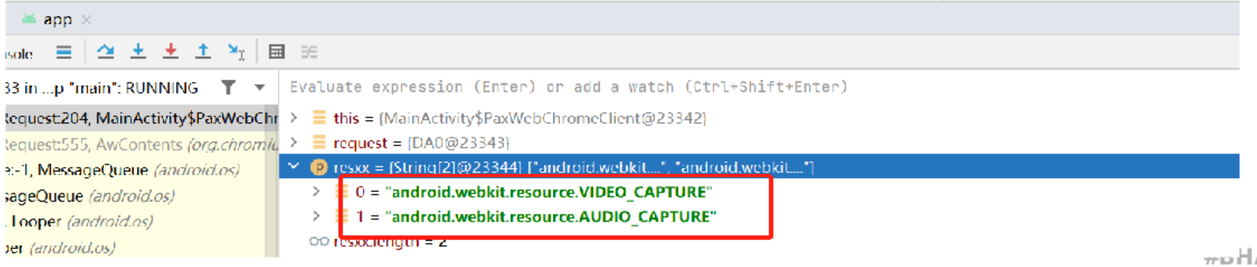
get camera



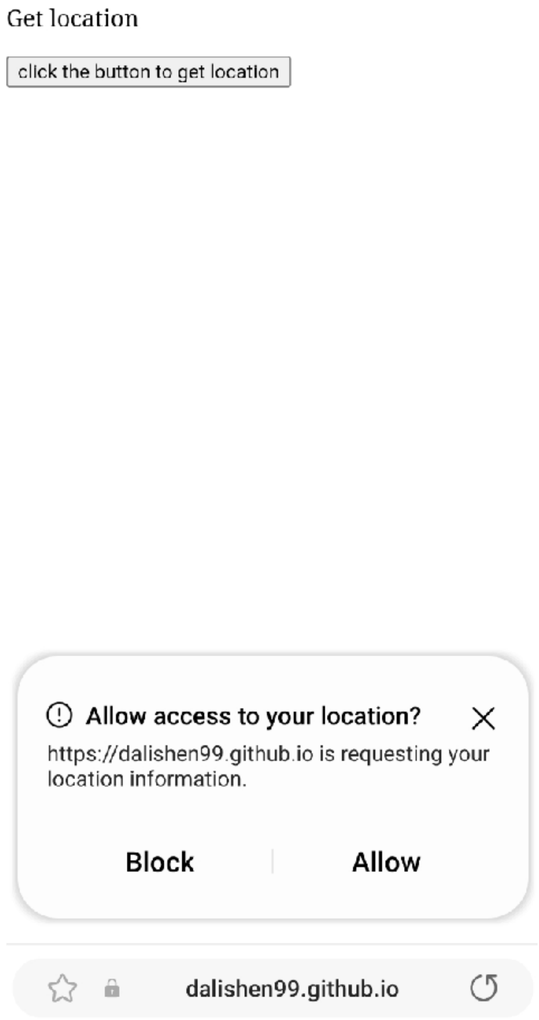


webView.setWebChromeClient(chromeClient);

```
public void onPermissionRequest(PermissionRequest request) {   request: DA0@23343
    String[] resxx = request.getResources();   resxx: ["android.webkit....", "android.webkit...."]   request: DA0@23343
    for (int i = 0; i < resxx.length; i++) {
        Log.e( tag: "proyx", resxx[i]);   resxx: ["android.webkit....", "android.webkit...."]
    }
    //super.onPermissionRequest(request);
    AlertDialog.Builder normalDialog = new AlertDialog.Builder( context: MainActivity.this);
    //normalDialog.setIcon(R.drawable.buttom_yello);
    normalDialog.setTitle("Open Camera");
    normalDialog.setMessage("Open Camera");
    normalDialog.setPositiveButton( text: "OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Log.e( tag: "proyx111", request.getClass().toString());
            Toast.makeText(MyApplication.getContext(), text: "OK, take a picture now", Toast.LENGTH_LONG).show();
            request.grant(request.getResources());
            Log.e( tag: "proyx111", request.toString());
        }
    });
    normalDialog.setNegativeButton( text: "no", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(MyApplication.getContext(), text: "you cancel open camera", Toast.LENGTH_LONG).show();
        }
    });
    normalDialog.show();
}
```



The Dangers of Unrestricted Access to Privacy Information Using Webview




```
webSettings.setDatabaseEnabled(true);
String dir = this.getApplicationContext().getDir("database",
Context.MODE_PRIVATE).getPath();
webSettings.setGeolocationEnabled(true);
webSettings.setGeolocationDatabasePath(dir);

@Override
public void onGeolocationPermissionsShowPrompt(String origin,
GeolocationPermissions.Callback callback) {
   callback.invoke(origin, true, false);
   super.onGeolocationPermissionsShowPrompt(origin, callback);
}
```

The Dangers of Unrestricted Access to Privacy Information Using Webview

Searching for *onGeolocationPermissionsShowPrompt* and *onPermissionRequest* function in github limit on .java or .kt file.

We got 1127 result back (Limited to Github search ability), and among them, 639 are positive cases.

The Dangers of Unrestricted Access to Privacy Information Using Webview

react-native-webview / react-native-webview Public

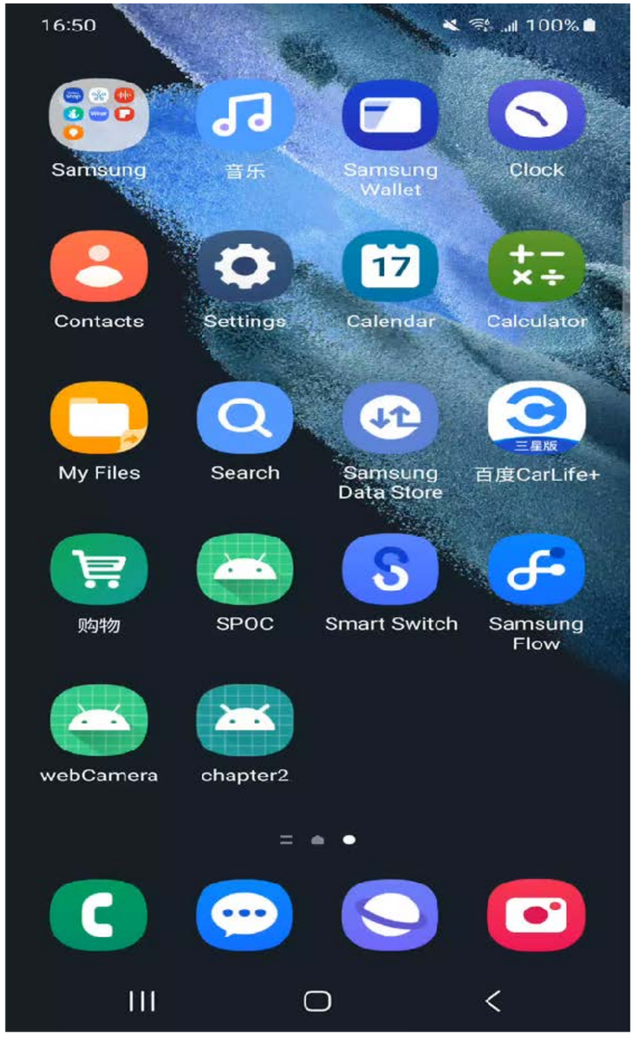
```
    }

    if (permission.equals(PermissionRequest.RESOURCE_PROTECTED_MEDIA_ID)) {
      if (granted && grantedPermissions != null) {
        grantedPermissions.add(PermissionRequest.RESOURCE_PROTECTED_MEDIA_ID);
      }
      shouldAnswerToPermissionRequest = true;
    }
  }

  if (shouldAnswerToPermissionRequest
    && permissionRequest != null
    && grantedPermissions != null) {
    permissionRequest.grant(grantedPermissions.toArray(new String[0]));
    permissionRequest = null;
    grantedPermissions = null;
  }
```

dalishen99 commented last month Author

I make a simple poc to get location for this:
https://dalishen99.github.io/location.html



https://github.com/react-native-webview/react-native-webview/issues/2903

Complex and confusing AD id

- Most users and even developers don't know the existence of AD ids.
- Hard to set even for domain experts, due to the complex UIs
- GAID is designed to be resettable, but resetting it is not much meaningful, as it is still there and can be used to track users during a particular period of time, for example, **cross-tracking the user in two apps is still possible**

App Set ID

Starting with Android 12 devices, Google Play will zero out the advertising ID when a user opts out of personalization in their Android Settings. Google Play has also introduced the App Set ID, which offers a privacy-friendly way to correlate usage or actions across a set of apps owned by the same organization.

IMA version 3.25.1 or higher includes the App Set ID SDK by default. App Set ID is essential to support non ads use-cases such as analytics and fraud prevention, when the advertising ID is zeroed out. For more information on the App Set ID see this Android developer guide.

- OAID is an AD id on Android OEM devices in China. Apps can get two AD ids in serval models of Android devices, i.e., GAID and OAID.
- Since OAID is not a feature in AOSP, there are more ways to bypass auditing on many Android phones to get OAID.

Our findings on these two advertising IDs

	OAID			GAID			
brand	OAID settings	restriction tracing is allowed	user authorization required	GAID exists	GAID settings	restriction tracing is allowed	user authorization required
A	Yes-8	Yes	No	No	-	-	-
B	Yes-3	Yes	No	No	-	-	-
C	Yes-5	Yes	No	Yes	Yes-3	Yes	No
D	Yes-4	Yes（but not work）	No	Yes	Yes-5	Yes	No
E	Yes-4	Yes（but not work）	No	Yes	Yes-5	Yes	No
F	Yes-4	Yes（but not work）	No	Yes	Yes	No UI	No
G	Yes-4	Yes	Yes	Yes	Yes-5	Yes（but not work）	No
H	Yes-5	Yes	No	Yes	No	No UI	No

Advertising IDs have actually become permanent or long-lasting!

Ways to Get Sensitive Data

01 Official channels provided by AOSP

02 Java reflection

03 Call in native code

04 Call directly through Binder

05 Call via vulnerabilities

06 Hidden channels

Ways to Get Sensitive Data

- Official channels provided by AOSP:

  Most are implemented through various Manager APIs

  Eg: TelephonyManager.getImei/getDeviceId…

- Java reflection：

  In this way static scanning can be bypassed

  eg: `telephonyMgr.getClass().getMethod("getImei", int.class).invoke(telephonyMgr, slotId);`

Ways to Get Sensitive Data

- Call in native code:

  difficult to analyze

  eg : jmethodID getDeviceId = ((*env)->GetMethodID(env, TelephoneManager_Cls, "getDeviceId", "()Ljava/lang/String;"));

  jobject imei= (*env)->CallObjectMethod(env, telephonymanager, getDeviceId);

- Call directly through Binder：

  difficult to analyze

  eg: IBinder mRemote = (IBinder) Class.forName("android.os.ServiceManager").getMethod("getService", String.class).invoke(null, "phone");

  mRemote.transact(144, _data, _reply, 0)

  String imei = _reply.readString();

Ways to Get Sensitive Data

- Call via vulnerabilities:

  Many OEM manufacturers add their own APIs. This may be error-prone.

  Those APIs may be vulnerable, leading to exploitable vulnerabilities.

  Such vulnerabilities are challenging to detect, as they are specific to the particular OEM.

Eg: CVE-2021-25344

```
@Override // com.samsung.android.knox.custom.IKnoxCustomManager
public String getSerialNumber() {
    return (String) this.mInjector.binderWithCleanCallingIdentity((FunctionalUtils.ThrowingSupplier) :
}

static /* synthetic */ String lambda$getSerialNumber$4() throws Exception {
    String serial = "ril.serialnumber";
    try {
        String rilSerial = SystemProperties.get(serial);
        if (TextUtils.isEmpty(rilSerial) || "00000000000".equals(rilSerial)) {
            serial = "ro.boot.serialno";
        }
        return SystemProperties.get(serial);
    } catch (Exception e) {
        Log.e(TAG, "getSerialNumber() failed - persistence problem " + e);
        return "";
    }
}
```

Ways to Get Sensitive Data

- Hiden channels:

① Get CPU SN:

/sys/devices/soc0/serial_number

```
[redacted] $ cat  /sys/devices/soc0/serial_number
2479336860
```

② Get IMSI/ICCID/Phone Number (A-201311522, won't fix): **target sdk <30** & without any permission

getContentResolver().query("content://telephony/siminfo/", null, null, null, null);

Google believes that all apps have an sdk version higher than 30, but this is not the case in third-party app stores!

Hook String constructor

Hook native String constructor to detect sensitive data

```
var artSymbol =
if (artSymbol)
    for (let i
        const e
        if ((el
            Int
        });
    }
    // hook
    if ((el
        Int
const envAddr = ptr(Java.vm.tryGetEnv().handle);
const envPointAddr = envAddr.readPointer();
const envPointAddr167 = envPointAddr.add((167) * Process.pointerSize);
const newStringUtfAddr = envPointAddr167.readPointer();
Interceptor.attach(newStringUtfAddr, {
    onEnter(args) {
    },
    onLeave(retval) {
        if (retval == 0x0) {
            return;
        }
        let ret = Java.vm.getEnv().getStringUtfChars(retval, null).readCString();
        isSensitiveInfoInString(ret,"jni::newStringUtf");
    }
});
const envPointAddr163 = envPointAddr.add((163) * Process.pointerSize);
const newStringAddr = envPointAddr163.readPointer();
Interceptor.attach(newStringAddr, {
    onEnter(args) {
    },
    onLeave(retval) {
        if (retval == 0x0) {
            return;
        }
        let ret = Java.vm.getEnv().getStringUtfChars(retval, null).readCString();
        isSensitiveInfoInString(ret,"jni::newString");
    }
});
```

Hook String constructor

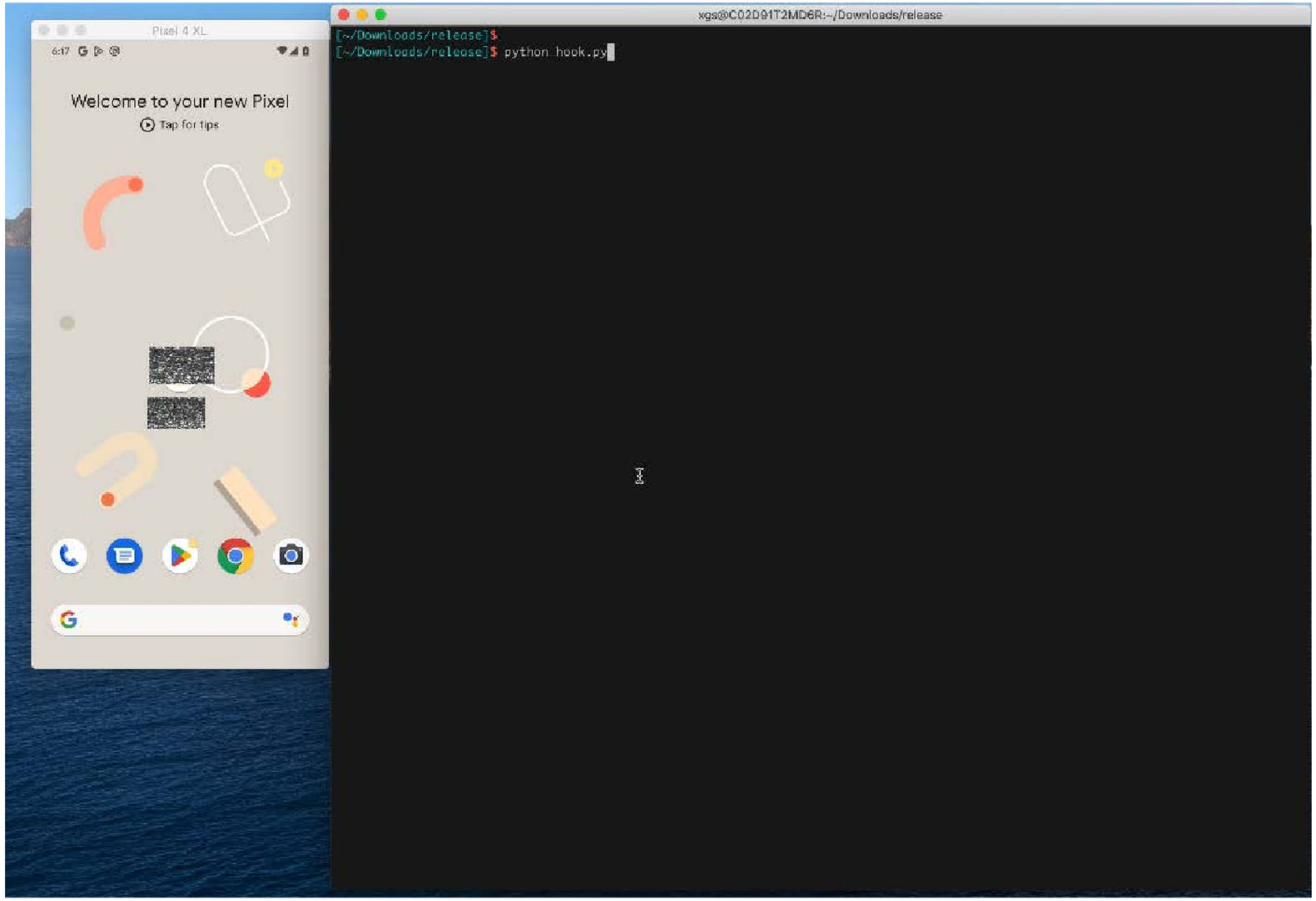
Advantage：

- No need to pay attention to the way the app calls sensitive data.
- Even if 0-day or n-day is used, it can be detected

Disadvantages：

- A large number of Strings are hooked, and the app runs stuck

Hook String


Pixel 4 XL
6:17
Welcome to your new Pixel
Tap for tips
xgs@C02D91T2MD6R:~/Downloads/release
[~/Downloads/release]$
[~/Downloads/release]$ python hook.py

Hook String constructor - results

Starting from Android 10, third-party apps are no longer allowed to obtain the **unique identifier of the device**.

Restriction on non-resettable device identifiers

Starting in Android 10, apps must have the `READ_PRIVILEGED_PHONE_STATE` privileged permission in order to access the device's non-resettable identifiers, which include both IMEI and serial number.

Caution: Third-party apps installed from the Google Play Store cannot declare privileged permissions.

Affected methods include the following:

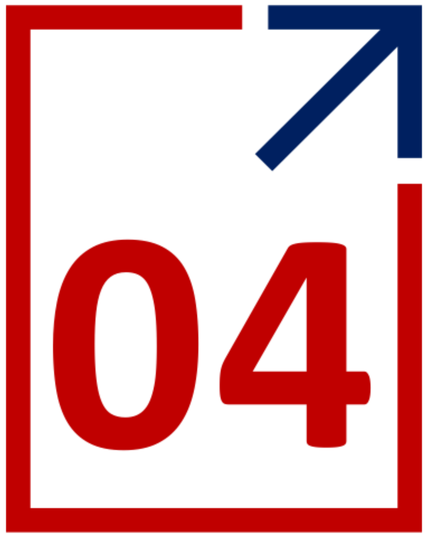
- `Build`
  - `getSerial()`
- `TelephonyManager`
  - `getImei()`
  - `getDeviceId()`
  - `getMeid()`
  - `getSimSerialNumber()`
  - `getSubscriberId()`

If your app doesn't have the permission and you try asking for information about non-resettable identifiers anyway, the platform's response varies based on target SDK version:

- If your app targets Android 10 or higher, a `SecurityException` occurs.
- If your app targets Android 9 (API level 28) or lower, the method returns `null` or placeholder data if the app has the `READ_PHONE_STATE` permission. Otherwise, a `SecurityException` occurs.

Hook String constructor - results

brand	Android version	CVE	UUID	with perms
Google	Android 10	CVE-2021-0428	ICCID	READ_PHONE_STATE
Samsung	Android 11	CVE-2021-25344	SN	without any perms
Samsung	Android 11	CVE-2021-25358	IMSI	without any perms
Samsung	Android 11	CVE-2021-25515	BSSID	without any perms
Samsung	Android 12	CVE-2022-22272	IMSI	READ_PHONE_STATE
Xiaomi	Android 11	CVE-2020-14105	SNO	without any perms



Summary

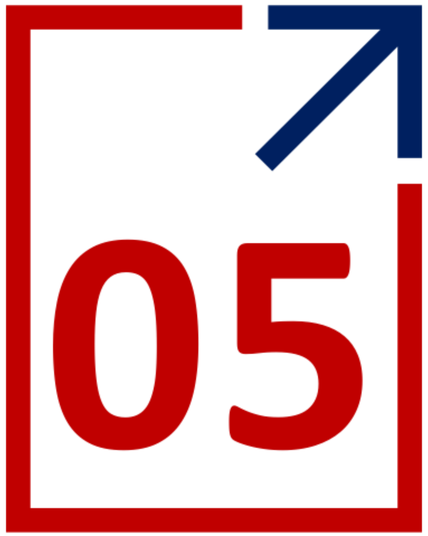
Summary & Key take-ways

- **System-level protection**: Starting from Android 10, third-party apps cannot obtain the unique identifier of the device. If this happens in an app, the app must have exploited some vulnerabilities (0-day or n-day).

- **App-level protection**: If the app's webview does not handle permissions properly, it will also be used by any URL to obtain user data.

- **The disaster of fragmentation**: Some OEMs do not strictly follow the AOSP permission policy, and many custom APIs can be used to obtain the unique identifier of the device.

- **New challenges**: The AD id becomes a persistent id to some extent. Users can be tracked continuously from the first power on until the phone is restored to factory Settings.



Q&A

twitter@cnwatcher