Android VAPT Report

(com.getsmartapp)

SMARTAPP

The report has been prepared by Security Team (Times Internet).

Application Summary

Application Name	com.getsmartapp
Application Namespace	com.getsmartapp
Version	2.9.3
Audit Date	2016-04-15 06:01
Application SHA1 Hash	cb69b75a95fbab24d1d7aae66edd81fc73ccdc16
Application MD5 Hash	06a83354c59b806f2d0543904c074092

Audit Summary

Unprotected Services	**Passed**
Improper Content Provider Permissions	**Passed**
Application Debug Enabled	**Passed**
Improper Custom Permission	**Passed**
Broken Trust Manager for SSL	**High Risk**
Broken hostnameverifier for SSL	**High Risk**
Insecure SSLSocketFactory Implemented	**Passed**

Hostname Verifier Allows all hostname	**Passed**
App extends WebViewClient	**Passed**
Unused Permissions	**Low Risk**
Remote Code Execution Through JavascriptInterface	**High Risk**
Insufficient Transport Layer Protection	**High Risk**
Derived Crypto Keys	**High Risk**
Application Logs	**Medium Risk**
Encryption SALT in NDK	**Medium Risk**

Priority Level	Number of failed test cases
High Risk	5
Medium Risk	1
Low Risk	1

Report Summary

Sno	Threat	Threat Level
1	Broken Trust Manager for SSL	High Risk
2	Broken hostname verifier for SSL	High Risk
3	Unused Permissions	Low Risk
4	Remote Code Execution Through JavaScript Interface	High Risk
5	Information in Shared Preference	Medium Risk
6	Insufficient Transport Layer Protection	High Risk
7	Derived Crypto Keys	High Risk
8	Application Logs	Medium Risk
9	Encryption SALT Stored on client in NDK	Medium Risk

TRUST MANAGER FOR SSL/TLS IS NOT PROPERLY CONFIGURED

SMARTAPP

Risk Rating:

HIGH

Android apps that use SSL/TLS protocols for secure communication should properly verify server certificates. The basic verification includes:

- verify that the subject (CN) of X.509 certificate and the URL matches
- verify that the certificate is signed by the trusted CA
- verify that the signature is correct
- verify that the certificate is not expired

A developer has the freedom to customize their SSL implementation. The developer should properly use SSL as appropriate to the intent of the app and the environment the apps are used in. If the SSL is not correctly used, a user's sensitive data may leak via the vulnerable SSL communication channel.

On Android, using HttpURLConnection is recommended for HTTP client implementation.

RISK ASSESMENT

Custom TrustManager is implemented in class io.fabric.sdk.android.services.network.mImplements naive certificate check.
This TrustManager breaks certificate validation!Referenced in method io.fabric.sdk.android.services.network.k->a

NONCOMPILANT CODE EXAMPLE

The following code implements a custom MySSLSocketFactory class that inherits javax.net.ssl.SSLContext:

```
public void emptyTrustManager() throws IOException, KeyManagementException, NoSuchAlgorithmException {     TrustManager tm = new
X509TrustManager() {

       @Override
       public void checkClientTrusted(X509Certificate[] chain,
             String authType) throws CertificateException {
          // Do nothing -> accept any certificates        }
       @Override
       public void checkServerTrusted(X509Certificate[] chain,
             String authType) throws CertificateException {
          // Do nothing -> accept any certificates        }

       @Override
       public X509Certificate[] getAcceptedIssuers() {           return null;
       }
    };
}
```

In the example above, checkClientTrusted()and checkServerTrusted() are overriden to make a blank implementation so that SSLSocketFactory does not verify the SSL certificate. The MySSLSocketFactory class is used to create an instance of HttpClient in another part of the application.

COMPILANT SOLUTION

The compliant solution is given below which shows how to solve the issue with the non compliant code:

```
public void checkCertTrusted(X509Certificate[] chain, String authType, boolean isServer)     throws CertificateException
{
    try {        if (isServer)
          appTrustManager.checkServerTrusted(chain, authType);        else
          appTrustManager.checkClientTrusted(chain, authType);
    } catch (CertificateException ae) {
       // if the cert is stored in our appTrustManager, we i gnore expiredness        if
(isExpiredException(ae)) {
          Log.i("log", "accepting expired certificate from keystore");          return;
       }
       if (isCertKnown(chain[0])) {
          Log.i("log", "accepting cert already stored in keystore");          return;
       }       try {
          if (defaultTrustManager == null)
             throw ae;
          Log.d("log", "trying defaultTrustManager");          if (isServer)
             defaultTrustManager.checkServerTrusted(chain, authType);          else
             defaultTrustManager.checkClientTrusted(chain, authType);
       } catch (CertificateException e) {
          Log.d("log", "defaultTrustManager failed: " + e);          interactCert(chain, authType, e);
       }
    }}

public void checkTrustManager() throws IOException, KeyManagementException, NoSuchAlgorithmException {     TrustManager tm = new
X509TrustManager() {

       @Override
       public void checkClientTrusted(X509Certificate[] chain,             String authType)
throws CertificateException {          checkCertTrusted(chain, authType, false);       }

       @Override
       public void checkServerTrusted(X509Certificate[] chain,             String authType)
throws CertificateException {          checkCertTrusted(chain, authType, false);
```

```
        }

        @Override
        public  X509Certificate[]  getAcceptedIssuers()  {            return
defaultTrustManager.getAcceptedIssuers();
        }
    };
}
```

The method checkCertTrusted() gives an idea about how to solve this issue. If a custom keystore is being used to load certificates then we need to define and do a failsafe load of those certificates.

BUSINESS IMPLICATION

In the event that a user (anonymous or verified) is able to execute overprivileged functionality, the business may experience the following impacts:

- Reputational Damage
- Fraud
- Information Theft

HOSTNAME VERIFIER FOR SSL/TLS IS NOT PROPERLY CONFIGURED

Risk Rating:

HIGH

SMARTAPP

Allowing All Hostnames: The app does not verify if the certificate is issued for the URL the client is connecting to. For example, when a client connects to example.com, it will accept a server certificate issued for some-other-domain.com.

On Android, using HttpURLConnection is recommended for HTTP client implementation.

RISK ASSESMENT

Implements naive hostname verification. This HostnameVerifier breaks certificate validation!

Custom HostnameVerifiers is implemented in class com.squareup.okhttp.internal.tls.OkHostnameVerifierImplements naive hostname verification. This HostnameVerifier breaks certificate validation!

NONCOMPILANT CODE EXAMPLE

The following code inherits javax.net.ssl.SSLContext:

```
public void HostnameVerifier() {
    HostnameVerifier hv = new HostnameVerifier() {
        @Override
        public boolean verify(String hostname, SSLSession session) {
            // Always return true -> Accespt any host names          return true;
        }
    };
}
```

The hostname verifier instead of verifying the hostname always returns true without checking the contents

COMPILANT SOLUTION

The code example shows how to verify hostname using a wrapHostnameVerifier which also checks for custom loaded certificates

```
public HostnameVerifier wrapHostnameVerifier(final HostnameVerifier defaultVerifier) {     if
(defaultVerifier == null)
        throw new IllegalArgumentException("The default verifier may not be null");

    return new SecuringHostnameVerifier(defaultVerifier);
}
```

```
class SecuringHostnameVerifier implements HostnameVerifier {     private HostnameVerifier
defaultVerifier;
    public MemorizingHostnameVerifier(HostnameVerifier wrapped) {        defaultVerifier =
wrapped;     }

    @Override
    public boolean verify(String hostname, SSLSession session) {
        Log.d("log", "hostname verifier for " + hostname + ", trying default verifier first");
        // if the default verifier accepts the hostname, we are d one        if
(defaultVerifier.verify(hostname, session)) {
            Log.d("log", "default verifier accepted " + hostname);           return true;        }
        // otherwise, we check if the hostname is an alias f or this cert in our keystore        try {
            X509Certificate cert = (X509Certificate)session.getPeerCertificates()[0];           if
(cert.equals(appKeyStore.getCertificate(hostname.toLowerCase(Locale.US)))) {               Log.d("log", "certificate for " +
hostname + " is in keystore. accepting.");              return true;           } else {
                Log.d("log", "server " + hostname + " provided wrong certificate.");              return false;
            }
        } catch (Exception e) {
            e.printStackTrace();           return false;
        }
    }
}
```

BUSINESS IMPLICATION

In the event that a user (anonymous or verified) is able to execute overprivileged functionality, the business may experience the following impacts:

- Reputational Damage
- Fraud
- Information Theft

DO NOT ASK FOR PERMISSIONS THAT MAY NOT BE USED

SMARTAPP

Risk Rating:

LOW

An app might request a user for certain permissions, like access to SD card, contacts, social profiles, etc. which has not actually been invoked while the scan was running. We list out all these permissions so that you can optimize your app and follow proper compliance checks.

RISK ASSESMENT

Application seems to be using extra permissions which are not really needed:

- 'GET_TASKS'
- 'PROCESS_OUTGOING_CALLS'
- 'WRITE_EXTERNAL_STORAGE'\
- 'RECEIVE'
- 'USE_CREDENTIALS'
- 'READ_CALL_LOG'
- 'RECEIVE_BOOT_COMPLETED'
- 'C2D_MESSAGE'
- 'RECEIVE_ADM_MESSAGE'
- 'UA_DATA'
- 'READ_SMS'
- 'RECEIVE_SMS'

COMPILANT SOLUTION

Do not request for permissions beyond what your app actually needs

BUSINESS IMPLICATION

In the event that a user (anonymous or verified) is able to execute overprivileged functionality, the business may experience the following impacts:

- Reputational Damage
- Fraud
- Information Theft

DO NOT PROVIDE ADDJAVASCRIPTINTERFACE METHOD ACCESS IN A WEBVIEW WHICH COULD CONTAIN UNTRUSTED CONTENT. (API LEVEL JELLY_BEAN OR BELOW)

SMARTAPP

Risk Rating:

HIGH

For API level JELLY_BEAN or below, allowing an app to use the addJavascriptInterface method with untrusted content in a WebView leaves the app vulnerable to scripting attacks using reflection to access public methods from JavaScript. Untrusted content examples include content from any HTTP URL (as opposed to HTTPS) and user-provided content. The method addJavascriptInterface(Object, String) is called from the android.webkit.WebView class. Sensitive data and app control should not be exposed to scripting attacks.

RISK ASSESMENT

Application seems to use JavascriptInterface, using that, an attacker can do a Remote Code Execution on your application and steal sensitive informations.

com/getsmartapp/screens/ProcessPaymentActivity;->onCreate

com/getsmartapp/screens/ProcessPaymentActivity;->onCreate

com/getsmartapp/screens/ProcessWalletPaymentActivity;->onCreate

com/getsmartapp/screens/ProcessWalletPaymentActivity;->onCreate

NONCOMPILANT CODE EXAMPLE

This noncompliant code example shows an application that calls the addJavascriptInterface () method, and hence is not secure for API level JELLY_BEAN and lower.

```
WebView webView = new WebView(this); setContentView(webView);...
class JsObject { private String sensitiveInformation;

...
public String toString() {     return
sensitiveInformation;
}

}
webView.addJavascriptInterface(new JsObject(), "injectedObject"); webView.loadData("", "text/html",
null); webView.loadUrl("http://www.example.com");
```

JavaScript can now control the host. Java reflection could be used to access any of the public methods of an injected object, using the permissions of the app.

COMPILANT SOLUTION

1. Compliant code could refrain from calling the addJavascriptInterface() method.

```
WebView  webView  =  new  WebView(this);
setContentView(webView);
```

2. Another compliant solution is to specify in the app's manifest that the app is only forAPI levels JELLY_BEAN_MR1 and above. For these API levels, only public methods that are annotated with JavascriptInterface can be accessed from JavaScript. API level 17 is JELLY_BEAN_MR1.

```
<manifest>
<uses-sdk  android:minSdkVersion="17"  /> ...


</manifest>
```

INFORMATION IN SHARED PREFERENCE

SMARTAPP

Risk Rating:

MEDIUM

Storing critical information should be avoided as much as possible.

RISK ASSESMENT

Shared Preferences are stored as a file in the filesystem on the device. They are, by default, stored within the app's data directory with filesystem premissions set that only allow the UID that the specific application runs with to access them. So, they are private in so much as Linux file permissions restrict access to them, the same as on any Linux/Unix system.

Anyone with root level access to the device will be able to see them, as root has access to everything on the filesystem. Also, any application that runs with the same UID as the creating app would be able to access them (this is not usually done and you need to take specific action to make two apps runs with the same UID, so this is probably not a big concern). Finally, if someone was able to mount your device's filesystem without using the installed Android OS, they could also bypass the permissions that restrict access.

If you're concerned about such access to your preferences (or any data written by your application), then you will want to encrypt it. If you are that concerned about them, you're going to need to figure out exactly how much protection is necessary for the level of risk you see. There is a very extensive discussion about this in Application Security for the Android Platform, just published in December 2011 (disclaimer: I'm the author of this book).

BUSINESS IMPLICATION

Broken cryptography will result in the following:

- MITM
- Tampered Requests
- False Belief of Security

SOLUTION

- Use Android Keystore to store them.
- Limit the Shared Prefrence mode to Private, to avoid data to be accessed by Third party apps

INSUFFICIENT TRANSPORT LAYER PROTECTION

Risk Rating:

HIGH

SMARTAPP

Insufficient transport layer protection issues happen when the data is sent from the mobile app to the server over unsecure channels. Whether the data is transmitted through the carrier network or through WiFi, it will end up through the Internet either way before it could reach the remote server. There are several ways where unprotected data transmitted over the network could be sniffed; things like routers, proxies, cell towers, are some of the few ways data could be sniffed while in transit.

RISK ASSESMENT

Insufficient transport layer protection issues happen when the data is sent from the mobile app to the server over unsecure channels. Whether the data is transmitted through the carrier network or through WiFi, it will end up through the Internet either way before it could reach the remote server. There are several ways where unprotected data transmitted over the network could be sniffed; things like routers, proxies, cell towers, are some of the few ways data could be sniffed while in transit.

This application opens a Socket and connects to it which might be insecured, which is defined at Lcom/squareup/okhttp/internal/http/RouteSelector;>nextInetSocketAddress

This application opens a Socket and connects to it which might be insecured, which is defined at Lcom/squareup/okhttp/internal/http/RouteSelector;>nextProxy

This application opens a Socket and connects to it which might be insecured, which is defined at Lcom/squareup/okhttp/internal/http/RouteSelector;>resetNextInetSocketAddress

This application opens a Socket and connects to it which might be insecured, which is defined at Lcom/squareup/okhttp/internal/http/SocketConnector;>connectRawSocket

GENERAL BEST PRACTICES

- Assume that the network layer is not secure and may potentially be hostile andeavesdropping.
- Enforce the use of SSL/TLS for all transport channels in which sensitive information,session tokens, or other sensitive data is going to be communicated to a backend API or web service.
- Remember to account for outside entities like 3rd party analytics companies, socialnetworks, etc, and use their SSL versions even when an application runs a routine via the browser/webkit. Mixed SSL sessions should be avoided and may expose the user?s session ID.
- Use strong, industry standard encryption algorithms and appropriate key lengths.
- Use certificates signed by a trusted CA provider.
- Never allow self-signed certificates, and consider certificate pinning for security conscious applications.
- Do not disable or ignore SSL chain verification.
- Only establish a secure connection after verifying the identity of the endpoint serverwith trusted certificates in the key chain.
- Alert users through the UI if an invalid certificate is detected.
- Do not send sensitive data over alternate channels, such as SMS, MMS, ornotifications.

NONCOMPLIANT CODE EXAMPLE

Using HTTP with SSL or TLS to connect to internet, or without a proper certificate the connection can be easily eavesdropped by attacker without the knowledge of the user.

```
String link = "http://www.google.com";
URL url = new URL(link);

HttpURLConnection conn = (HttpURLConnection) url.openConnection(); conn.connect();

InputStream is = conn.getInputStream();
BufferedReader reader =new BufferedReader(new InputStreamReader(is, "UTF-8")); String webPage =
"",data="";
while ((data = reader.readLine()) != null){
   webPage += data + "\n"; }
```

One can do a MITM attack and the use wouldn't even know that someone is eavesdropping it.

COMPLIANT CODE EXAMPLE

Never use HTTP url to download data. create a valid HTTPS request through which only sensitive data can be downloaded.

BUSINESS IMPLICATION

The violation of a user's confidentiality may result in:

- Identity theft
- Fraud
- Reputational Damage

DO NOT USE DEFAULT ANDROID CRYPTOGRAPHIC SECURITY PROVIDER ENCRYPTION AND FOLLOW THE RECOMMENDED PRACTICES

Risk Rating:

HIGH

SMARTAPP

The predominant Android cryptographic security provider API defaults to using an insecure AES encryption method: ECB block cipher mode for AES encryption. Android's default cryptographic security provider (since version 2.1) is BouncyCastle.

NOTE: Java also chose ECB as a default value when only the AES encryption method is chosen. So, this rule also applies to Java, but for Java's different default cryptographic security provider. Oracle Java's default cryptographic security provider is SunJCE.

Default behaviors of cryptographic libraries used in Android systems often do not use recommended practices. For example, the predominant Android Java security provider API defaults to using an insecure AES encryption method: ECB block cipher mode for AES encryption. Extensive app testing by [Egele 2013] has shown that the following 6 rules are often not followed, resulting in 88% of apps with cryptographic APIs on Google Play making at least one mistake.

Six common cryptography rules they tested:

1. Do not use ECB mode for encryption.
2. Do not use a non-random IV for CBC encryption.
3. Do not use constant encryption keys.
4. Do not use constant salts for PBE.
5. Do not use fewer than 1,000 iterations for PBE.
6. Do not use static seeds to seed SecureRandom(·).

RISK ASSESMENT

Traces of Crypto Keys which might be intermediate keys used when loading the different libraries.

AES was found to be implemented at Lcom/freshdesk/mobihelp/e/ag;->ag. This kind of Algorithm shouldn't be used, try using HMAC or other Algorithms

AES was found to be implemented at Lcom/freshdesk/mobihelp/e/ag;->ah. This kind of Algorithm shouldn't be used, try using HMAC or other Algorithms

AES/ECB/PKCS7Padding was found to be implemented at Lio/fabric/sdk/android/services/common/CommonUtils;->a. This kind of Algorithm shouldn't be used, try using HMAC or other Algorithms

AES was found to be implemented at Lcom/freshdesk/mobihelp/e/ag;->cy. This kind of Algorithm shouldn't be used, try using HMAC or other Algorithms

BUSINESS IMPLICATION

Broken cryptography will result in the following:

- Privacy Violations
- Information Theft
- Code Theft
- Intellectual Property Theft
- Reputational Damage

DO NOT LOG SENSITIVE INFORMATION

SMARTAPP

Risk Rating:

MEDIUM

Android provides capabilities for an app to output logging information and obtain log output. Applications can send information to log output using the android.util.Log class. To obtain log output, applications can execute the logcat command.

RISK ASSESMENT

Current Application was found to be writing logs to the system logs

- This application logs the message 'called' under the tag 'Application
- OnCreate: '
- This application logs the message 'mStoredCards' under the tag 'mStoredCards
- '
- This application logs the message 'Landroid/view/View;->getHeight()I' under the tag 'Landroid/view/View;->getHeight()I'
- This application logs the message 'Ripple Effect end' under the tag
- 'Landroid/graphics/Paint;->()V'
- This application logs the message 'sdndfs' under the tag 'created'
- This application logs the message 'CREATE TABLE sms_data_ref (message_id
- INTEGER PRIMARY KEY AUTOINCREMENT, sender_number TEXT NOT NULL, person_name
- TEXT NOT NULL, timestamp INTEGER NOT NULL, sms_type TEXT NOT NULL, number_type TEXT DEFAULTstd, network_type TEXT NOT NULL, is_night INTEGER NOT NULL DEFAULT 0, is_roaming INTEGER NOT NULL DEFAULT 0, sms_location_feature
- TEXT, sms_location_locality TEXT, sms_location_state TEXT,
- sms_location_country TEXT, circle_id INTEGER, country_name TEXT, is_synced INTEGER DEFAULT 0, sim_no INTEGER NOT NULL DEFAULT 1, sync_time INTEGER,
- UNIQUE (sender_number, timestamp))' under the tag 'mesg_table'
- This application logs the message 'onActivityResult' under the tag
- 'onActivityResult'
- This application logs the message 'Ljava/lang/Object;->toString()Ljava/lang/
- String;' under the tag 'Ref'
- This application logs the message 'Lretrofit/RetrofitError;->getMessage
- ()Ljava/lang/String;' under the tag 'Generate Ref Code'

- This application logs the message 'hiject API Fails' under the tag 'backend'
- This application logs the message 'oops' under the tag 'hijeck failed'
- This application logs the message 'data Data' under the tag 'data Data'
- This application logs the message 'Ljava/util/Map$Entry;->getValue()Ljava/ lang/Object;' under the tag 'datajson' This application logs the message 'done' under the tag 'stop service'
- This application logs the message 'Lcom/getsmartapp/lib/utils/SDKUtils;-
- >isStringNullEmpty(Ljava/lang/String;)Z' under the tag 'Lcom/getsmartapp/lib/ utils/SDKUtils;->isStringNullEmpty(Ljava/lang/String;)Z'
- This application logs the message 'Channel registration failed.' under the tag 'IntentReceiver'
- This application logs the message 'Lretrofit/RetrofitError;->getCause()Ljava/ lang/Throwable;' under the tag 'plans'
- This application logs the message 'Lorg/json/JSONArray;->toString()Ljava/ lang/String;' under the tag 'JSON STRING'
- This application logs the message 'Lretrofit/RetrofitError;->getCause()Ljava/ lang/Throwable;' under the tag 'plans' This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'https://getsmartapp.com/recharger-api-1.4/ wallet/payUFailure ' under the tag '1' This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'onSuccess' under the tag 'onSuccess'
- This application logs the message 'onFailure' under the tag 'onFailure'
- This application logs the message 'Ljava/lang/Object;->toString()Ljava/lang/
- String;' under the tag 'Get Ref'
- This application logs the message 'Lretrofit/RetrofitError;->getMessage
- ()Ljava/lang/String;' under the tag 'Get Ref'
- This application logs the message 'failure loading container' under the tag
- 'GTM'
- This application logs the message 'GTM' under the tag 'Logging a Callback!'
- This application logs the message 'onSizeChanged. W: %d, H: %d 2' under the
- tag 'PullToRefresh'
- This application logs the message 'isFirstItemVisible. Empty View.' under the tag 'PullToRefresh'
- This application logs the message 'isLastItemVisible. Empty View.' under the tag 'PullToRefresh'
- This application logs the message 'Ljava/lang/StringBuffer;->toString()Ljava/ lang/String;' under the tag 'Trans Hash' This application logs the message 'getCardByID' under the tag 'CARDEXCEPTION'
- This application logs the message 'getCardRankings' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'getSimpleCardRankMap' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'handleLinkedCardPair' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'insertFixedCards' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'insertRemainingCards' under the tag

- 'CARDEXCEPTION'
- This application logs the message 'resetContextCoefficient' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'getSimpleCardRankMap' under the tag
- 'CARDEXCEPTION'
- This application logs the message 'Lretrofit/RetrofitError;->getCause()Ljava/ lang/Throwable;' under the tag 'Gplus profile error'
- This application logs the message 'Lorg/json/JSONObject;->toString()Ljava/ lang/String;' under the tag 'Set_Identity'
- This application logs the message 'Lio/branch/referral/z;->a()Ljava/lang/
- String;' under the tag 'Set_Identity'
- This application logs the message 'Lretrofit/RetrofitError;->getMessage
- ()Ljava/lang/String;' under the tag 'Ref'

TO LOG OUTPUT

The android.util.Log class allows a number of possibilities:

Log.d (Debug)	Log.e (Error)	
Log.i (Info)	Log.v (Verbose)	Log.w (Warn)

EXAMPLE

```
Log.v("method", Login.TAG + ", account=" + str1);
Log.v("method", Login.TAG + ", password=" + str2);
```

TO OBTAIN LOG OUTPUT

Declare READ_LOGS permission in the manifest file so that an app can read log output:

AndroidManifest.xml:

`<uses-permission android:name="android.permission.READ_LOGS"/>` Call logcat from

an application:

```
Process mProc = Runtime.getRuntime().exec( new String[]{"logcat", "-d",

"method:V *:S$Bc`W^(B)"}); BufferedReader mReader = new

BufferedReader(   new InputStreamReader(proc.getInputStream()));
```

Prior to Android 4.0, any application with READ_LOGS permission could obtain all the other applications' log output. After Android 4.1, the specification of READ_LOGS permission has been changed. Even applications with READ_LOGS permission cannot obtain log output from other applications.

However, by connecting an Android device to a PC, log output from other applications can be obtained.

Therefore, it is important that applications do not send sensitive information to log output.

APPLICABILITY

Applications should make sure that they do not send sensitive information to log output. If the app includes a third party library, the developer should make sure that the library does not send sensitive information to log output. One common solution is for an application to declare and use a custom log class, so that log output is automatically turned on/off based on Debug/Release. Developers can use ProGuard to delete specific method calls. This assumes that the method contains no side effects.

NONCOMPLIANT CODE EXAMPLE

1. Facebook SDK for Android contained the following code which sends Facebook accesstokens to log output in plain text format.

   ```
   Log.d("Facebook-authorize", "Login Success! access_token=" +        getAccessToken() + "
   expires=" + getAccessExpires());
   ```

2. Here is another example. A weather report for Android sent a user's location data tothe log output as follows:

   ```
   I/MyWeatherReport( 6483): Re-use MyWeatherReport data I/ ( 6483): GET
   JSON:
   http://example.com/smart/repo_piece.cgi? arc=0&lat=26.209026&lon=127.650803&rad=50&dir=-
   999&lim=52&category=1000
   ```

   If a user is using Android OS 4.0 or before, other applications with READ_LOGS permission can obtain the user's location information without declaring ACCESS_FINE_LOCATIONpermission in the manifest file.

EXAMPLE PROOF OF CONCEPT

Example code of obtaining log output from a vulnerable application is as follows:

```
try {
    Process  mLogcatProc;      mLogcatProc  =  Runtime.getRuntime().exec(new

String[]

       {"logcat",  "-d",  "LoginAsyncTask:I  APIClient:I  method:V  *:S"  });      BufferedReader  reader  =  new

BufferedReader(new  InputStreamReader(         mLogcatProc.getInputStream()));

    String  line;

    String  separator  =  System.getProperty("line.separator");

    while  ((line  =  reader.readLine())  !=  null)  {


       slog.append(line);


       slog.append(separator);
    }


    Toast.makeText(this,  "Obtained  log  information",
       Toast.LENGTH_SHORT).show();

}

catch  (IOException  e)  {

//  handle  error

}

TextView  tView  =  (TextView)  findViewById(R.id.logView); tView.setText(slog);
```

DO NOT STORE ENCRYPTION SALT OR CRITICAL KEYS IN NDK

Risk Rating:

MEDIUM

SMARTAPP

The application stores key/salt, in NDK. This appears to be safe and secured from being exposed, but a sophisticated hacker can easily capture it.

RISK ASSESMENT

It can be used to execute server side critical requests, that requires hash/encryption key.

BUSINESS IMPLICATION

Broken cryptography will result in the following:

- MITM
- Tampered Requests
- False Belief of Security

SOLUTION

Encryption keys should not be stored on client side.
But, if it required, use Android Keystore to store them.

REFRENCE

https://developer.android.com/training/articles/keystore.html