SUBEX

Android – BlackRock Malware Threat Report

Date: 06-07-2021

K. Narahari

Overview:

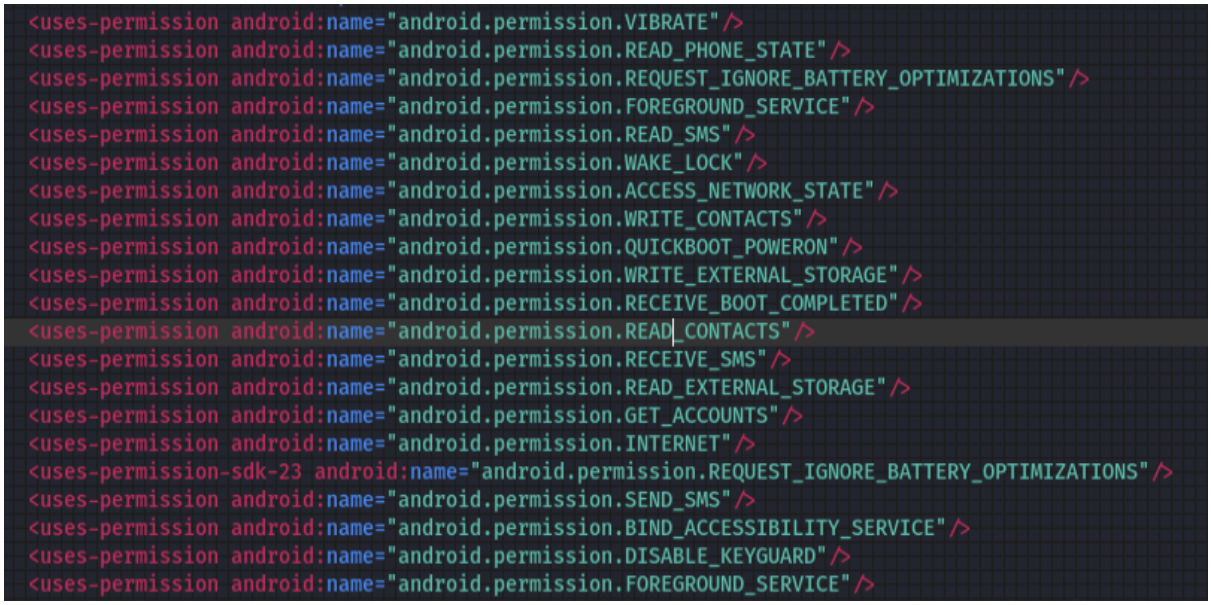
In the present scenario, smartphones with advanced computing capabilities and cross-platform applications are user-friendly with the clients and customers. The attackers are interested in smartphones as there is a large scope for them to perform malicious activities.

In our Honeypot, we found a malicious Blackrock Android Trojan Malware which can steal information like passwords and credit-card information from about 377 smartphone applications including Amazon, Facebook, Gmail, and Tinder. Since these are very popular applications, the threat posed by the BlackRock Android Malware is quite high.

File Hash: **d9a961119f96ed632a2542d97b3a0ae2**

Technical Analysis:

We have downloaded the malware sample and reverse-engineered the apk then performed static analysis and went through the manifest.xml file which is the primary file to begin the analysis.



From the permission list used by this apk, we can observe that there are plenty of permissions given which are dangerous and used by the malware to perform illegal and malicious tasks.

“READ_SMS, WRITE_SMS, SEND_SMS, RECEIVE_SMS” is used to steal the OTP, verification codes, message related (hiding, forwarding, intercepting). With the broadcast receiver, they are broadcasting the SMS and this malicious app will have listeners to capture and it will be sent to the attacker.

```
public void getAddAttendeeToContact(Context context, final QMAttendee qMAttendee, final QMCallback<Intent> qMCallback) {
    RuntimePermissionUtility.requestIfNeeded(context, getLocaler(), "android.permission.WRITE_CONTACTS", new QMCallback<Boolean>(
        /* class com.quickmobile.conference.attendees.QMAttendeesComponent.AnonymousClass1 */

        public void done(Boolean bool, Exception exc) {
            if (bool.booleanValue()) {
                String phone = qMAttendee.getPhone();
                String email = qMAttendee.getEmail();
                String title = qMAttendee.getTitle();
                String company = qMAttendee.getCompany();
                Intent intent = new Intent("android.intent.action.INSERT");
                intent.setType("vnd.android.cursor.dir/contact");
```

From the above screenshot, we can observe the attacker is declaring the permission inside the java method at runtime. Using the intents, the attacker is passing the contact information. Along with that, attacker is taking the phone number, email ID, title, and company if it is stored in the contact file.

```
PUuAqQuTaTyKuIjZfRgNkTcNyAz() {
    this.value = new AtomicReference<>();
    ReentrantReadWriteLock reentrantReadWriteLock = new ReentrantReadWriteLock();
    this.lock = reentrantReadWriteLock;
    this.readLock = reentrantReadWriteLock.readLock();
    this.writeLock = this.lock.writeLock();
```

ReadWriteLock is implemented by ReentrantReadWriteLock package. This is generally implemented to stop the deadlock situation, but here attacker is using this as an advantage. When the attacker is stealing the data from the victim, then victim cannot use the resources as this lock has been implemented. So, other threads requesting readLocks must wait till the write Lock is released. Hence, until the attacker releases the lock, the user will not have any control.

```
private LineBarVisualizer mVisualizer;
private boolean permissionToRecordAccepted = true;
private String[] permissions = {"android.permission.RECORD_AUDIO"};
```

From the malicious application, attacker is recording the audio from the victim device and storing in local device then passing the recorded audio files to the attacker server.

```
public Intent getShareIntent(Context context, Uri uri, String str) {
    int i = 0;
    String string = getLocaler().getString(L.LABEL_PHOTO_SHARE_SUBJECT, new Object[]{getQMQuickEvent().getAppShortName()});
    Intent intent = new Intent("android.intent.action.SEND");
    intent.setType("image/jpeg");
    intent.putExtra("android.intent.extra.STREAM", uri);
    intent.putExtra("android.intent.extra.TITLE", string);
    intent.putExtra("android.intent.extra.SUBJECT", string);
    intent.putExtra("title", string);
    intent.putExtra("description", str);
    intent.addFlags(1);
```

Using the intents, attacker is sharing the images to the other applications. android.intent.action.**SEND** is commonly used to share an image but can be used to share any type of binary content, along with that Uri, title, subject is also attached with the images.

```
(context.checkCallingOrSelfPermission("android.permission.GET_ACCOUNTS") == 0) {
 Pattern pattern = Patterns.EMAIL_ADDRESS;
 Account[] accounts = AccountManager.get(context).getAccounts();
```

Android.permission.GET_ACCOUNTS is used to get access to the existing logged-in accounts from the device. It allows this malicious app to get the list of accounts known by the phone. This may include any accounts created by applications you have installed. With this permission, the attacker can take control of all the accounts on the victim device.

```
if (ActivityCompat.checkSelfPermission(this.mContext, "android.permission.ACCESS_FINE_LOCATION") == 0 || ActivityCompat.checkSelfPerm
    this.mMap.setMyLocationEnabled(true);
    this.mMap.addMarker(position);
    this.mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, (float) this.mContext.getResources().getInteger(2131427346)));
    return;
}
return;
```

In order to receive the location updates from the network provider or GPS provider, you must request the user's permission by declaring the android.permission.ACCESS_FINE_LOCATION in the manifest file or declaring through source code. Without these permissions, we cannot access the location of the device. In this malicious application attacker using this permission to track the exact location of the victim.

```
public static final String VIDEOS_ALLOW_ROTATION = "videoAllowRotation";
public static final String VIDEOS_CURRENT_POSITION = "videoCurrentPosition";
public static final String VIDEOS_DISALLOW_DEVICE_PLAYER = "videoAllowDevicePlayer";
public static String VIDEOS_SHOW_CONTROLS = "videoShowControlsInd";
public static String VIDEOS_URI = "videoURI";
```

These above-mentioned strings are used by the application to take control of the camera related settings. Inside, the methods the attacker will be using these strings to take photos, videos from the camera and send it back to the attacker server. These strings are used for alignment purpose for videos to play or record.

```
public void setupFragment(View view) {
    super.setupFragment(view);
    this.webView = (WebView) view.findViewById(2131362671);
    this.webView.getSettings().setSupportMultipleWindows(true);
    this.webView.setWebChromeClient(new WebChromeClient() {
```

WebView is used to embed the internal browser inside an application. Here this application uses WebView with JavaScript enabled to execute the malicious scripts which is triggered by

the browser. They have made Chrome as default browser to steal login credentials or cookies which might be available from the browser.

```
public void onDownloadStart(String str, String str2, String str3, String str4, long j) {
    QLBuilder with = QL.with(QMWebFragment.this.qmQuickEvent.getQMContext(), this);
    with.d("WebView file download started from primary method with mime type " + str4);
    QMWebFragment.this.downloadFile(str, str4);
```

After loading the browser using the WebView, the attacker is downloading the files from the WebView client. It might be a malicious file that can be also embedded malware or any other kind of threat actor.

IOCs:

https://pdftron.s3.amazonaws.com/downloads/android/pdftron.pfx
http:// th02.deviantart.net/fs71/PRE/f/2014/103/a/2a12fb3ad0f8863164bb57034-d7e5xxg.png

MITRE Techniques:

Broadcast receivers (T1402)
Access Contact List (T1432)
Masquerade as Legitimate Application (T1444)
Access Sensitive Data in Device Logs (T1413)
Execution (TA0041)

Subex Secure Protection

Subex Secure detects the android sample as “SS_Gen_Andro_BlackRockTrojan_A”.

Our Honeypot Network

This report has been prepared from the threat intelligence gathered by our honeypot network. This honeypot network is today operational in 62 cities across the world. These cities have at least one of the following attributes:

- Are landing centers for submarine cables
- Are internet traffic hotspots
- House multiple IoT projects with a high number of connected endpoints
- House multiple connected critical infrastructure projects

- Have academic and research centers focusing on IoT
- Have the potential to host multiple IoT projects across domains in the future.

Over 3.5 million attacks a day is being registered across this network of individual honeypots. These attacks are studied, analysed, categorized, and marked according to a threat rank index, a priority assessment framework that we have developed within Subex. The honeypot network includes over 4000 physical and virtual devices covering over 400 device architectures and varied connectivity mediums globally. These devices are grouped based on the sectors they belong to for purposes of understanding sectoral attacks. Thus, a layered flow of threat intelligence is made possible.