SUBEX

Android Info-Stealer Threat Report

Date: 16/07/2021
Sanjuktasree Chatterjee

Android applications are now providing various functions in one single application. These possess some security risks to the users. Android is a very popular operating system for most users. Cybercriminals are actively targeting this platform and the applications to conduct the attack.

On `15th July 2021`, we found a zero-day Android package file in our honeypot which is highly malicious. This file is a variant of a Chinese camera application '`beauty camera`' which is found to be a malicious info-stealer that can disable the system's antivirus and steal data from the device and may install other applications to crash the user's device.

File Hash: `c459dd321bedded775a4235cdee980bc`

Technical Analysis:

The malware file is bundled with 3 classes.dex file (Fig:1) in an Android package file. The classes.dex files contain few information about malicious activity.

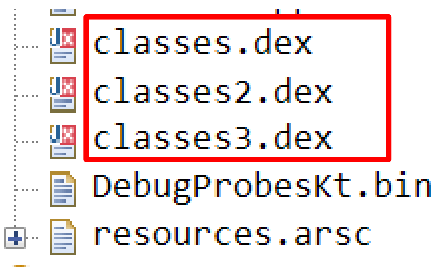

Fig: 1

The package name shows that the file is '`beauty camera`' which is a Chinese application but has some malicious characteristics which show that the attacker uses the source code of the main application to lure the victim. There are multiple encoded strings that has been seen during the analysis of the android source code(Fig:2). These has been encoded with `Base64`.

Fig: 2

There are too many permissions given to the application which are not required for a normal camera application. "ACCESS_COARSE_LOCATION" will check the location of the user. There are NETWORK_STATE, WIFI_STATE, INTERNET permissions which are used to check and create remote connections to install other apps and deliver information to the CnC server. READ_PHONE_STATE permission helps to gather more information about the device and data. "CLEAR_APP_CACHE", "WRITE_EXTERNAL_STORAGE", "SYSTEM_ALERT_WINDOW",

“EXPAND_STATUS_BAR” are some permissions that are usually not required for a camera application. These seems to be malicious characteristics for this sample.

```
<uses-sdk android:minSdkVersion="23" android:targetSdkVersion="28"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.EXPAND_STATUS_BAR"/>
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.SYSTEM_OVERLAY_WINDOW"/>
<uses-permission android:name="android.permission.CLEAR_APP_CACHE"/>
```

Fig: 3

The permissions related to the device power, system boot process and WRITE_SETTINGS (Fig:4) leads to change the security settings and can also do system crash. DISABLE_KEYGUARD helps to disable the security configuration.

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.DEVICE_POWER"/>
<uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
```

Fig: 4

The application checks for user authentication services that uses authentication to open an account from the user’s device.

```
public class AuthenticationService extends Service {
    public AccountAuthenticator mAuthenticator;

    public class AccountAuthenticator extends AbstractAccountAuthenticator {
        @Override // android.accounts.AbstractAccountAuthenticator
        public Bundle addAccount(AccountAuthenticatorResponse accountAuthenticatorResponse,
            return null;
        }
```

Fig: 5

The actions may be performed remotely by using the codes API (Fig:6) calls.

```
public static void write(RemoteActionCompat remoteActionCompat, VersionedParcel versionedParcel) {
    androidx.core.app.RemoteActionCompatParcelizer.write(remoteActionCompat, versionedParcel);
}
```

Fig: 6

Multiple network communication has been made with the malicious domain to install malicious files and to transfer the information to the attacker.

```
[string@0000a97f] http://
[string@0000a980] http://127.0.0.1
[string@0000a981] http://211.151.146.65:8080/wlantest/shanghai_sun/mock_ad_server_intersitial_video.json
[string@0000a982] http://configapi-api.glqa.jpushoa.com/v1/status
[string@0000a983] http://m.baidu.com
[string@0000a984] http://mobads.baidu.com/ads/index.htm
[string@0000a985] http://mobads.baidu.com/ads/pa/
[string@0000a986] http://mobads.baidu.com/cpro/ui/mads.php
[string@0000a987] http://schemas.android.com/apk/res/android
[string@0000a988] http://sf1-ttcdn-tos.pstatp.com/obj/ttfe/adfe/union_endcard/Lark20190725-175511.png
[string@0000a989] http://union.baidu.com/
```

Fig: 7

When the application runs in the victim's system it collects data about all background processes that are running on that and if antivirus runs in background, it kills all the process related to that.

```
public static Set<String> getAllBackgroundProcesses() {
    List<ActivityManager.RunningAppProcessInfo> runningAppProcesses = ((ActivityManager) Utils.getApp().
    HashSet hashSet = new HashSet();
    if (runningAppProcesses != null) {
        for (ActivityManager.RunningAppProcessInfo runningAppProcessInfo : runningAppProcesses) {
            Collections.addAll(hashSet, runningAppProcessInfo.pkgList);
        }
    }
    return hashSet;
}

public static Set<String> killAllBackgroundProcesses() {
    ActivityManager activityManager = (ActivityManager) Utils.getApp().getSystemService("activity");
    List<ActivityManager.RunningAppProcessInfo> runningAppProcesses = activityManager.getRunningAppProce
    HashSet hashSet = new HashSet();
    if (runningAppProcesses == null) {
        return hashSet;
```

Fig: 8

The sample also collects Wi-Fi related information (Fig: 9) to establish remote connection with attackers' server.

```
WifiInfo connectionInfo = wifiManager.getConnectionInfo();
NetworkInfo networkInfo = connectivityManager.getNetworkInfo(1);
if (connectionInfo == null || networkInfo == null || !networkInfo.isConnected() || (
    return "{}";
}
```

Fig: 9

All the external files and directories related data in the devices are gathered by the sample (Fig: 10).

```
public static File[] getExternalFilesDirs(Context context, String str) {
    if (Build.VERSION.SDK_INT >= 19) {
        return context.getExternalFilesDirs(str);
    }
    return new File[]{context.getExternalFilesDir(str)};
}

public static File[] getExternalCacheDirs(Context context) {
    if (Build.VERSION.SDK_INT >= 19) {
        return context.getExternalCacheDirs();
    }
    return new File[]{context.getExternalCacheDir()};
}
```

Fig: 10

The service information like version, name of the service running is being collected and stored in external location.

```
public static <T> T getSystemService(Context context, Class<T> cls) {
    if (Build.VERSION.SDK_INT >= 23) {
        return (T) context.getSystemService(cls);
    }
    String systemServiceName = getSystemServiceName(context, cls);
    if (systemServiceName != null) {
        return (T) context.getSystemService(systemServiceName);
    }
    return null;
}
```

Fig: 11

IOCS:

Malicious URLs:

http://m.baidu.com
http://union.baidu.com/
http://sf1-ttcdn-tos.pstatp.com/obj/ttfe/adfe/union_endcard/Lark20190725-175511.png

MITRE Techniques:

Install Insecure or Malicious Configuration(T1478)
Masquerade as Legitimate Application(T1444)
Access Sensitive Data in Device Logs(T1413)
Deliver Malicious App via Other Means (T1476)
Generate Fraudulent Advertising Revenue(T1472)

Subex Secure Protection

Subex Secure detects the malware as "`SS_Gen_Android_InfoStealer_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, analyzed, 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.