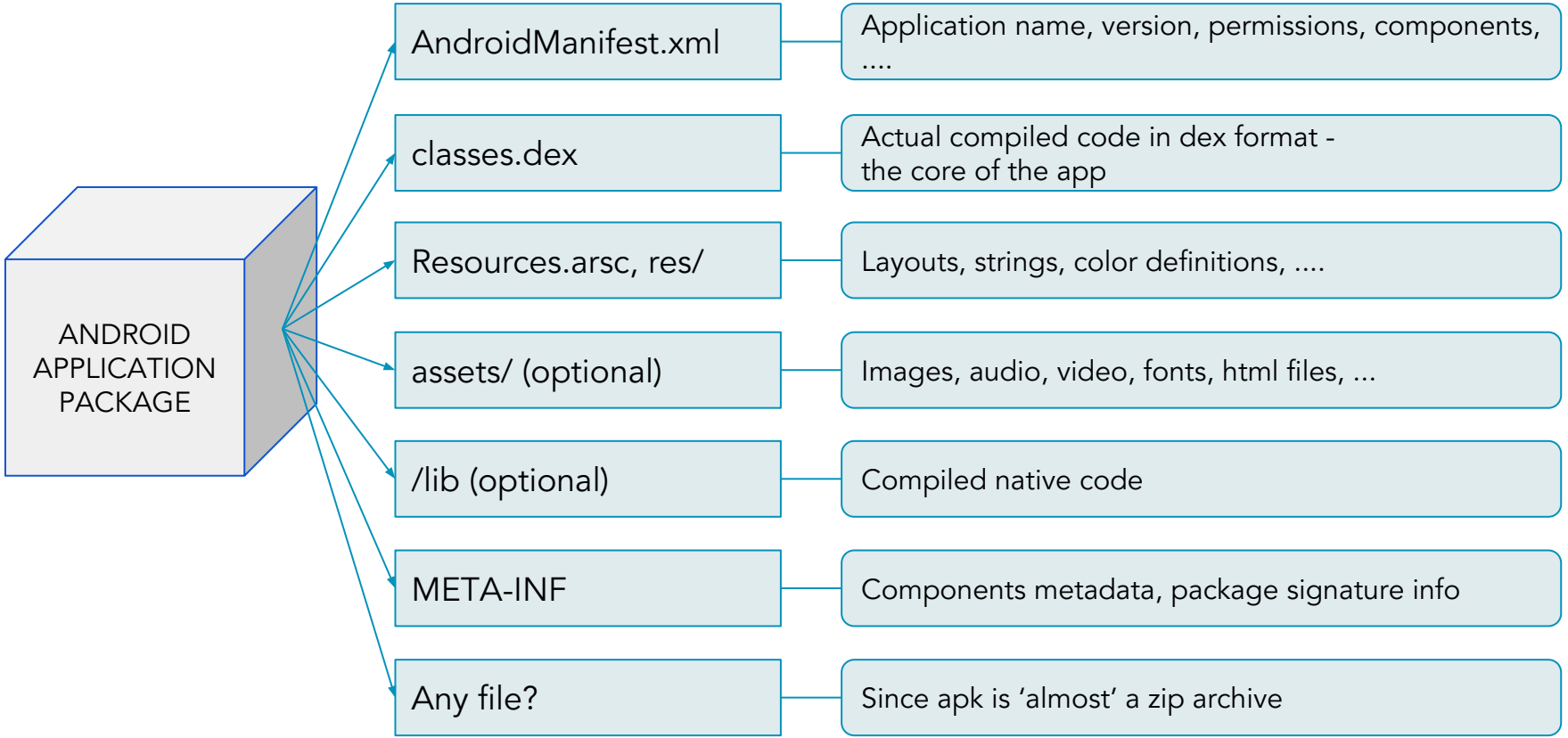
BLOWING THE COVER: HANDS-ON ANALYSIS OF HANDCRAFTED ANDROID MALWARE

Alex Reshetniak | September 26 2018

About Me

- Senior Security Researcher @ Lookout
- B.S. in Information Security
- More than 5 years working in Information Security
- Experience includes incident response, forensics, malware analysis

APK File Structure



Glossary

What I am talking about

- Malware family - a group of malicious applications that are common in code structure, functionality, and are usually associated with the same threat actor
- Obfuscation - a process of deliberately making the code of a program harder to read and understand
- Decompiler - a reverse engineering tool used to convert a compiled executable program into (pseudo) source code

Why obfuscate?

- Protect Intellectual Property (IP)
- Complicate Reverse Engineering
- Prevent tampering

Why decompile code?

- Much easier to read decompiled Java-like code than disassembled Smali instructions

Common obfuscation techniques

Often used by malware authors

- Trivial (repackaging the apk, renaming the package, resigning)
- Insertion of dead code blocks
- Insertion intermediate operations
- Encrypting/Encoding strings
- Encrypting payloads - dex files or native libraries
- Usage of reflection

```
if(v18 % 8 == 0) {
    int v16 = v11 ^ v2[0];
    int v15 = v10 ^ v2[1];
    int v14 = v9 ^ v2[2];
    v13 = v2[3] ^ v12;
    int v17;
    for(v17 = 4; v17 < 36; v17 += 4) {
        v21 = v3_1[v16 & 0xFF] ^ v4[v15 >> 8 & 0xFF] ^ v5_1[v14 >> 16 & 0xFF] ^ v6[v13 >>> 24] ^ v2[v17];
        v22 = v3_1[v15 & 0xFF] ^ v4[v14 >> 8 & 0xFF] ^ v5_1[v13 >> 16 & 0xFF] ^ v6[v16 >>> 24] ^ v2[v17 + 1];
        v23 = v3_1[v14 & 0xFF] ^ v4[v13 >> 8 & 0xFF] ^ v5_1[v16 >> 16 & 0xFF] ^ v6[v15 >>> 24] ^ v2[v17 + 2];
        v13 = v3_1[v13 & 0xFF] ^ v4[v16 >> 8 & 0xFF] ^ v5_1[v15 >> 16 & 0xFF] ^ v6[v14 >>> 24] ^ v2[v17 + 3];
        v17 += 4;
        v16 = v3_1[v21 & 0xFF] ^ v4[v22 >> 8 & 0xFF] ^ v5_1[v23 >> 16 & 0xFF] ^ v6[v13 >>> 24] ^ v2[v17];
        v15 = v2[v17 + 1] ^ (v3_1[v22 & 0xFF] ^ v4[v23 >> 8 & 0xFF] ^ v5_1[v13 >> 16 & 0xFF] ^ v6[v21 >>> 24]
        v14 = v3_1[v23 & 0xFF] ^ v4[v13 >> 8 & 0xFF] ^ v5_1[v21 >> 16 & 0xFF] ^ v6[v22 >>> 24] ^ v2[v17 + 2];
        v13 = v3_1[v13 & 0xFF] ^ v4[v21 >> 8 & 0xFF] ^ v5_1[v22 >> 16 & 0xFF] ^ v6[v23 >>> 24] ^ v2[v17 + 3];
    }

    v21 = v3_1[v16 & 0xFF] ^ v4[v15 >> 8 & 0xFF] ^ v5_1[v14 >> 16 & 0xFF] ^ v6[v13 >>> 24] ^ v2[v17];
    v22 = v3_1[v15 & 0xFF] ^ v4[v14 >> 8 & 0xFF] ^ v5_1[v13 >> 16 & 0xFF] ^ v6[v16 >>> 24] ^ v2[v17 + 1];
    v23 = v3_1[v14 & 0xFF] ^ v4[v13 >> 8 & 0xFF] ^ v5_1[v16 >> 16 & 0xFF] ^ v6[v15 >>> 24] ^ v2[v17 + 2];
    v13 = v3_1[v13 & 0xFF] ^ v4[v16 >> 8 & 0xFF] ^ v5_1[v15 >> 16 & 0xFF] ^ v6[v14 >>> 24] ^ v2[v17 + 3];
    v14 = v17 + 4;
    v11 = v2[v14] ^ (v7[v21 & 0xFF] & 0xFF ^ (v7[v22 >> 8 & 0xFF] & 0xFF) << 8 ^ (v7[v23 >> 16 & 0xFF] & 0xFF
    v10 = v2[v14 + 1] ^ (v7[v22 & 0xFF] & 0xFF ^ (v7[v23 >> 8 & 0xFF] & 0xFF) << 8 ^ (v7[v13 >> 16 & 0xFF] &
    v9 = v2[v14 + 2] ^ (v7[v23 & 0xFF] & 0xFF ^ (v7[v13 >> 8 & 0xFF] & 0xFF) << 8 ^ (v7[v21 >> 16 & 0xFF] & 0
```

```
this.bjove = new int[]{0x4DCC, 5719, 0x47BD, 28, 80, 13};
this.ieqduv = 17;
this.xgzmajvkt = false;
if(!this.xgzmajvkt || true != this.mfdtx) {
    int[] v1_1 = new int[5];
    int v2 = 2;
    v0 = this.mfdtx ? 1 : 0;
    int v3 = v0 + 81 + this.bjove.length + 0x19F2;
    v0 = this.lvutl ? 1 : 0;
    v3 += v0;
    int v4 = this.bhtsrfwvr;
    v0 = this.madbxm ? 1 : 0;
    v1_1[v2] = v0 + v4 - 51 + v3 - (this.ieqduv * 17 + this.ieqduv + this.bjove.length);
}
else {
    this.lvutl(this.madbxm, this.mfdtx, this.bhtsrfwvr, 46, false);
    v0 = this.xgzmajvkt ? 1 : 0;
    v1 = 7796 - v0;
    v0 = this.xfiykfspk ? 1 : 0;
    v1 = v0 + v1 + 20881 - (0xEA15 + this.bhtsrfwvr);
    v0 = this.lvutl ? 1 : 0;
    v1 -= v0 - 0x4420;
    v0 = this.xgzmajvkt ? 1 : 0;
    this.bhtsrfwvr = v0 * 91 - 92 + v1;
}

v0 = this.bhtsrfwvr != 0 ? 1 : 0;
if(v0 != 1) {
    v0 = true == this.xfiykfspk ? 1 : 0;
    v0 = v0 != 1 ? 1 : 0;
    v1 = this.bjove.length > 0 ? 1 : 0;
```

Sample 1

Cosiloon

Cosiloon malware family

Quick reference info

Type: Adware/App Dropper

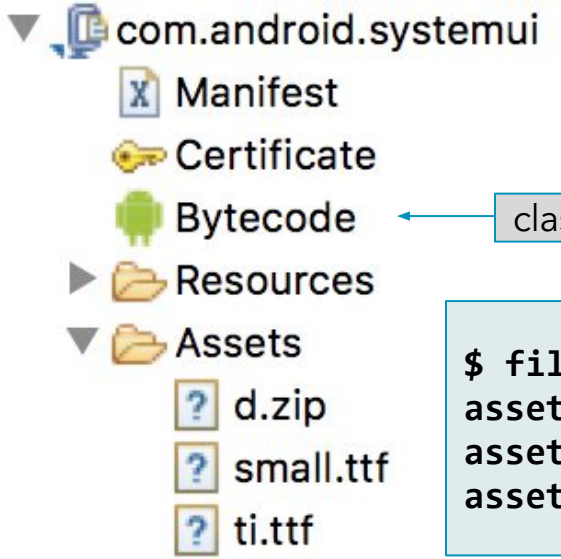
What it does:

Shows advertisement on the device screen.

Can silently download and install additional applications.

Cosiloon - opened in JEB decompiler

The apk content suggests there might be some hidden executable code



classes.dex

```
$ file assets/*
assets/d.zip:          data
assets/small.ttf:      data
assets/ti.ttf:         data
```

SHA256:	fe2a8a18c5859b0665a17b50de5837f0fc38af39d43e9b32e68192edf1cf5d6e
File name:	d.zip
Detection ratio:	1 / 59
Analysis date:	2018-07-14 15:16:37 UTC (1 month, 3 weeks ago)

SHA256:	4497d3b41694ddb21dea51842d419e29047335e3089202e7dabc959ee2d7ab02
File name:	ti.ttf
Detection ratio:	1 / 59
Analysis date:	2018-02-10 21:29:37 UTC (6 months, 3 weeks ago)

SHA256:	8bea608404c0218649509d5df4a7de2d17198b30edb43
File name:	small.ttf
Detection ratio:	0 / 57
Analysis date:	2018-02-10 21:30:27 UTC (6 months, 3 weeks ago)

Cosiloon payload decoding

Decoding d.zip file with Java

```
$ file d.zip_out
d.zip_out: Java archive data (JAR)
```

```
public class Main {

    private static int magicInt = 47;

    public static void main(String[] args) throws IOException {
    public static long decode_1(InputStream inputStream, OutputStream outputStream) throws IOException {
        return decode_2(inputStream, outputStream, new byte[0x400]);
    }
    public static long decode_2(InputStream inputStream, OutputStream outputStream, byte[] byteArr) throws IOException {
        long i = 0;
        if(inputStream != null && outputStream != null) {
            if(verify_arr(byteArr)) {
                byteArr = new byte[0x400];
            }
            while(true) {
                int k = inputStream.read(byteArr);
                if(-1 == k) {
                    return i;
                }
                decode_3(byteArr);
                outputStream.write(byteArr, 0, k);
                i += ((long)k);
            }
        }
        return i;
    }
    public static void decode_3(byte[] byteArr) {
        for(int i = 0; i < byteArr.length; ++i) {
            byteArr[i] = ((byte)(((byte)(byteArr[i] ^ 47))));
        }
    }
    public static boolean verify_arr(byte[] byteArr) {
        if(byteArr != null && byteArr.length != 0) {
            return false;
        }
        return true;
    }
}
```

```
public static void decode_3(byte[] byteArr) {
    for(int i = 0; i < byteArr.length; ++i) {
        byteArr[i] = ((byte)(((byte)(byteArr[i] ^ 47))));
    }
}
```

SHA256:	68aa20d143f71fd6c07210e9214b3937073d3f7cc79d4aaac8badc50225b5d62
File name:	46adf4efbf287a262a2517716bc22b3a.virus
Detection ratio:	13 / 58
Analysis date:	2018-07-02 05:28:22 UTC (2 months ago)

Cosiloon payload decoding

Decoding other obfuscated files

```
public static void decode_asset(InputStream inputStream, OutputStream outputStream) {
    int v0 = 0x400;
    try {
        byte[] bytes_read = new byte[v0];
        byte[] bytes_out = new byte[inputStream.available()];
        v0 = 0;
    label_6:
        int v6 = inputStream.read(bytes_read);
        if(v6 >= 0) {
            int v2 = 0;
            while(true) {
                if(v2 >= v6) {
                    goto label_6;
                }

                bytes_out[v0] = ((byte)(((byte)(bytes_read[v2] ^ 0x79))));
                ++v2;
                ++v0;
            }
        }

        outputStream.write(bytes_out);
    }
```

```
try {
    new g("2").a(c.a(arg4, c.a("c21hbGwudHRm")), new FileOutputStream(v0_1));
}
```

Base64.decode(“c21hbGwudHRm”) => small.ttf

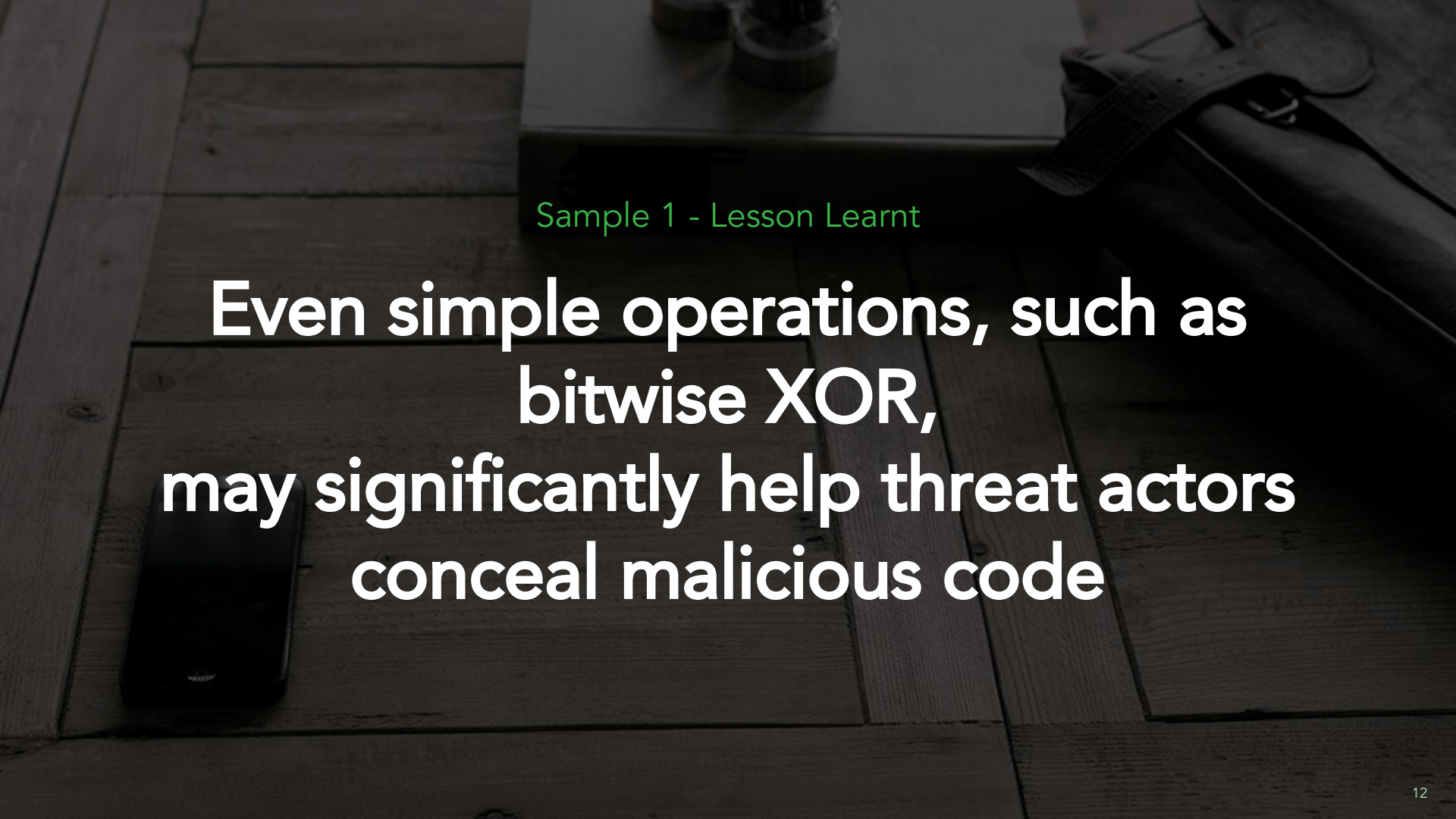
```
public void decrypt_payload(InputStream inputStream, OutputStream outputStream) {
    Cipher cipher = Cipher.getInstance(c.b());
    cipher.init(2, this.key);
    CipherOutputStream cipherOutStream = new CipherOutputStream(outputStream, cipher);
    byte[] bytes_res = new byte[0x400];
    while(true) {
        int i = inputStream.read(bytes_res);
        if(i < 0) {
            break;
        }

        cipherOutStream.write(bytes_res, 0, i);
    }

    cipherOutStream.close();
    outputStream.close();
    inputStream.close();
}
```

Cosiloon - decoded files

Original files in assets folder

Decoded files

```
$ file *
d.zip:          data
small.ttf:      data
ti.ttf:         data
d.zip_out:      Java archive data (JAR)
small.ttf_out:  Java archive data (JAR)
ti.ttf_out:     Java archive data (JAR)
```



Sample 1 - Lesson Learnt

Even simple operations, such as bitwise XOR, may significantly help threat actors conceal malicious code

Sample 2

DressCode

DressCode malware family

Quick reference info

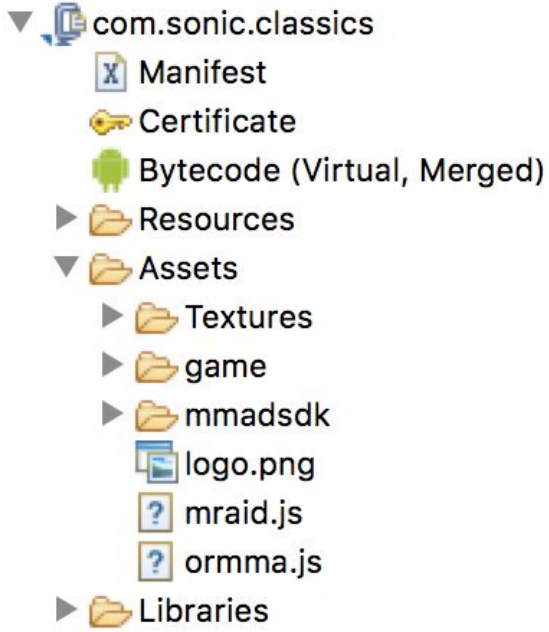
Type: ClickFraud

What it does:

Turns user's device into a proxy, using SOCKS protocol. Performs 'clicks' on advertisements on behalf of the user to generate revenue.

DressCode - all the files are of known types

Two Javascript files and an image, nothing suspicious. Well...is it?



On basic principles of RGB model and steganography

About colors, bytes, and bits

rgb(28, 69, 135)

```
R: 28 => 00011100
G: 69 => 01000101
B: 135 => 10000111
```

128	64	32	16	8	4	2	1
0	0	0	1	1	1	0	0
16+8+4=28							

Changing the least significant bit

128	64	32	16	8	4	2	1
0	0	0	1	1	1	0	**1**
16+8+4**+1**=**29**							

Changing the most significant bit

128	64	32	16	8	4	2	1
1	0	0	1	1	1	0	0
128+16+8+4=**156**							

Insignificance of the least significant bits

Results of manipulation with 2 least significant bits of the color

```
rgb(100, 100, 100)
0x64, 0x64, 0x64

0x64 == 0110 0100
```

```
rgb(103, 103, 103)
0x67, 0x67, 0x67

0x67 == 0110 0111
```

Understanding the decoding routine

Extracting the payload

Load Image from assets -> toRBitmap -> fromBase63 => classes.dex

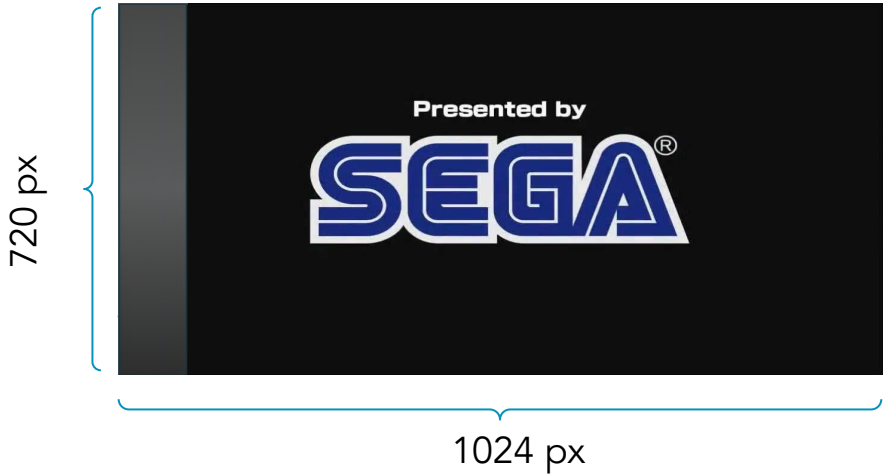
```
public static byte[] fromBase63(ArrayList arg5) {
    byte[] v0 = new byte[arg5.size() / 2];
    int v1;
    for(v1 = 0; v1 < arg5.size(); v1 += 2) {
        v0[v1 / 2] = ((byte)(arg5.get(v1 + 1).byteValue() + (arg5.get(v1).byteValue() << 4)));
    }

    return v0;
```

```
public static ArrayList toRBitmap(Bitmap arg6, ArrayList arg7) {
    int v1;
    for(v1 = 0; v1 < arg6.getWidth(); ++v1) {
        int v2 = 0;
        while(true) {
            if(v2 < arg6.getHeight()) {
                int v3 = arg6.getPixel(v1, v2);
                int v0 = v3 & 3 | (v3 & 0x300) >>> 6 | (0x30000 & v3) >>> 12;
                if(v0 != 16) {
                    arg7.add(Byte.valueOf(((byte)v0)));
                    ++v2;
                    continue;
                }
            }
            else {
                break;
            }

            return arg7;
        }
    }

    return null;
}
```

```
$ file *
logo.png:     PNG image data, 1280 x 720, 8-bit/color RGBA, non-interlaced
logo.png_out: Dalvik dex file version 035
```

Sample 2 - DressCode

About 10% of the image is used for storing the code

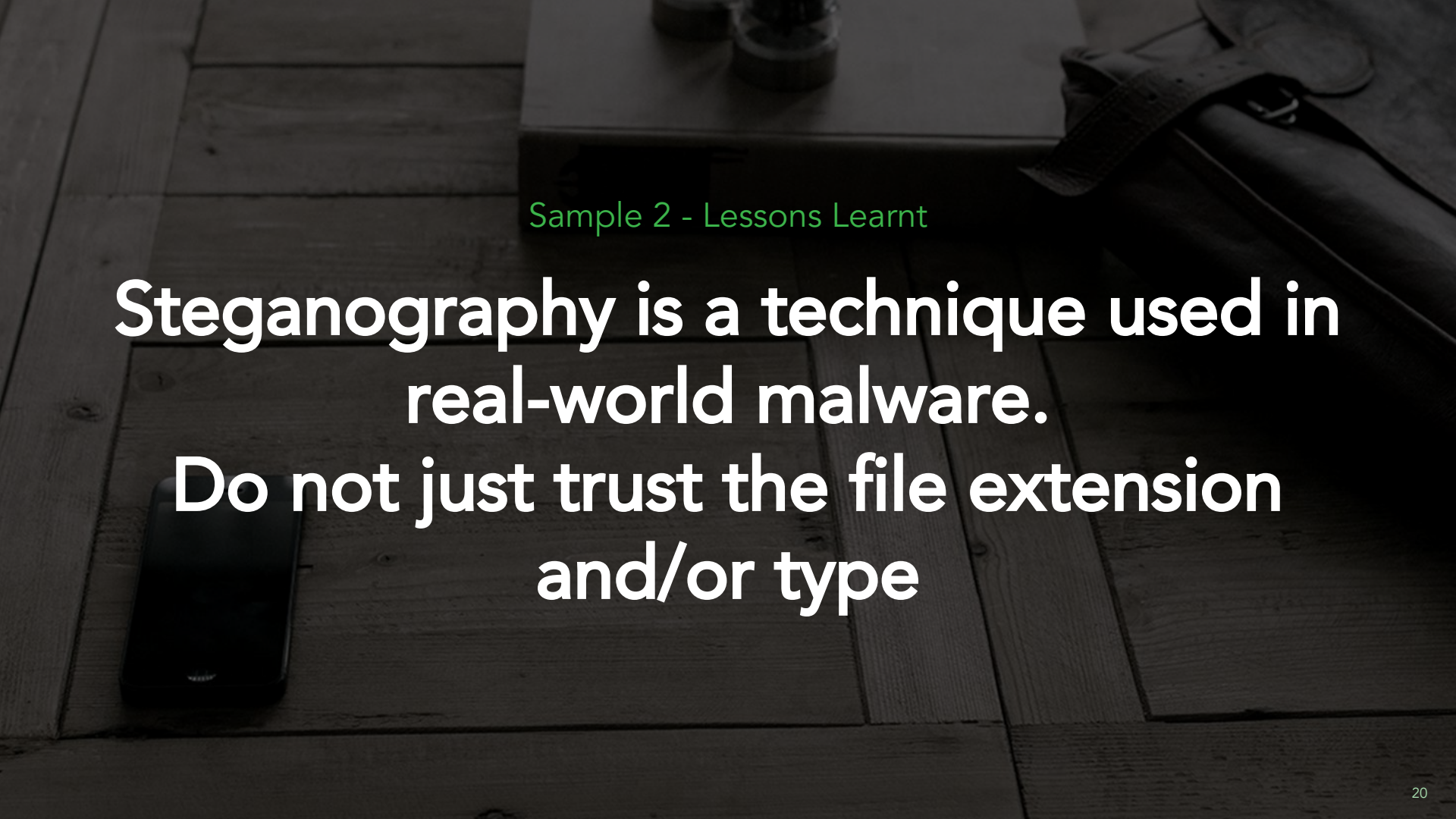


Payload size: 35840 bytes

Required # of pixels:
35864 x 2 = 71728

# of pixels in the image:
1024 x 720 = 737280

% of the picture taken by payload:
71728 / 737280 x 100% ≈ 10%



Sample 2 - Lessons Learnt

Steganography is a technique used in real-world malware.
Do not just trust the file extension and/or type

Sample 3

Xafecopy

Xafecopy malware family

Quick reference info

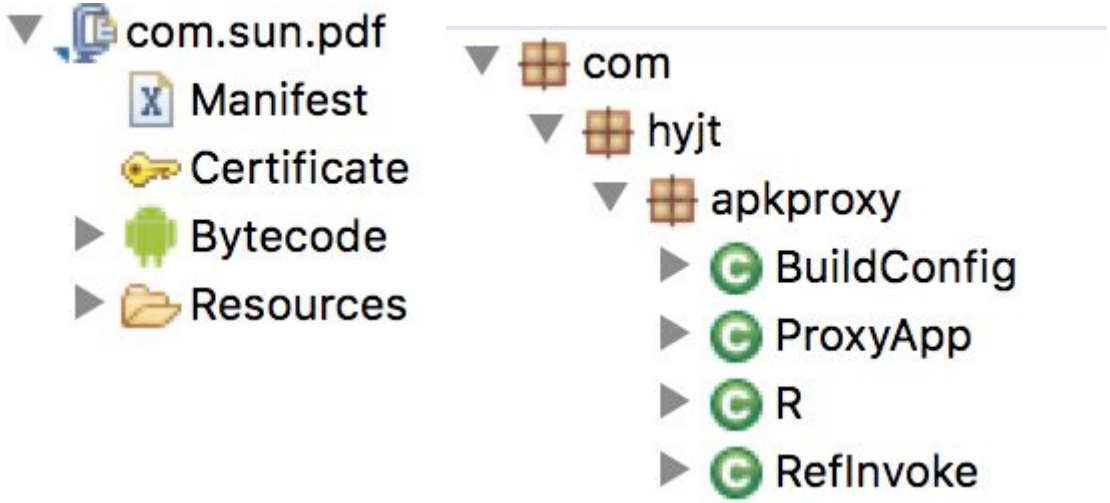
Type: Chargeware

What it does:

It will silently visit specially crafted URLs and attempt to subscribe the user for paid services.

Xafecopy in decompiler

Ok, I know it must be somewhere in the file……which file?




```
$ ls -lh classes.dex
... 3.6M .... classes.dex
```

Xafecopy and its main method

Self-explanatory method names

```
protected void attachBaseContext(Context arg19) {
    super.attachBaseContext(arg19);
    try {
        File v10 = this.getDir("odex", 0);
        File v4 = this.getDir("dex", 0);
        this.odexDir = v10.getAbsolutePath();
        this.libPath = this.odexDir;
        this.apkPath = v4.getAbsolutePath() + "/p.apk";
        File v5 = new File(this.apkPath);
        if(!v5.exists()) {
            v5.createNewFile();
Log.i("demo", "dex size:" + this.splitPayLoadFromDex(this.readDexFileFromApk(), v5));
            ProxyApp.unPackSOFromApk(this.apkPath, this.libPath);
        }
```

Xafecopy payload revealed

Split payload from dex???

```
private byte[] decrypt(byte[] arg3) {
    int v0;
    for(v0 = 0; v0 < arg3.length; ++v0) {
        arg3[v0] = ((byte)(arg3[v0] ^ 0xC7));
    }

    return arg3;
}
```

```
private int splitPayLoadFromDex(byte[] bytes_in, File file_out) throws IO
    byte[] bytes_out;
    int dex_length = bytes_in.length;
    byte[] length_storage = new byte[4];
    System.arraycopy(bytes_in, dex_length - 4, length_storage, 0, 4);  //   arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
    int pld_length = new DataInputStream(new ByteArrayInputSt    am(length_storage)).readInt();
    if(pld_length < dex_length) {
        System.out.println(Integer.toHexString(pld_length));
        bytes_out = new byte[pld_length];
        System.arraycopy(bytes_in, dex_length - 4 - pld_length, bytes_out, 0, pld_length);
    }

    bytes_out = this.decrypt(bytes_out);
    try {
        FileOutputStream v4 = new FileOutputStream(file_out, false);
        v4.write(bytes_out);
        v4.flush();
        v4.close();
```

Read the last 4 bytes from the classes.dex file

Copy the specified number of bytes from the end of the classes.dex file into a new file

Xafecopy - manual payload extraction

Read the last 4 bytes of the classes.dex file ->

Copy the specified number of bytes into a new file ->

Perform an XOR operation on the bytes of the new file ->

Get the hidden APK file!

95	93	E9	94	81	97	8C	C6	C5	D3	C7	D3	C7	CF	CF	CF
C7	34	67	78	8D	D4	83	1D	56	3A	C4	C7	C7	F5	C3	C7
C7	D6	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	D9
5E	D7	C7	8A	82	93	86	EA	8E	89	81	E8	84	82	95	93
E9	95	94	86	97	8C	C2	C1	C7	C7	C7	C7	90	C6	90	C6
3D	B9	C7	C7	9D	5A	D7	C7	C7	C7	00	11	1C	6A		

`0x00111C6A == 1121386`

.dex | apk_from_dex_encoded

lit As: Hex | Run Script | Run Template

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
97	8C	C4	C3	D3	C7	CF	CF	CF	C7	34	67	78	8D	1D	7D	—ŒÄÃÓÇÏÏÏÇ4gx..}
63	69	06	CE	C7	C7	06	CB	C7	C7	DB	C7	C3	C7	A6	B4	ci.ÎÇÇ.ËÇÇÛÇÃÇ¦´
B4	A2	B3	B4	E8	A1	AE	A9	A3	A5	B2	B3	B3	A8	A9	F5	´¢³´è¡®©£¥²³³¨©õ
F7	F6	F1	F6	F5	F5	F1	E9	AD	B4	39	0D	C7	C7	D2	51	÷öñöõõñé-´9.ÇÇÒQ
F2	65	2B	CF	CB	C3	68	25	BB	C5	F4	C2	DC	4B	5E	5E	òe+ÏËÃh%»ÅôÂÜK^^
F4	F4	49	5E	88	78	28	D8	87	8E	8C	12	E2	0D	C7	A0	ôôI^^x(Ø‡ŽŒ.â.Ç
54	81	A4	77	CF	DC	0B	30	5F	A0	E9	FD	61	68	73	10	T.¤wÏÜ.0_ éýahs.

Binary Xor:

Treat Data As: Signed Byte

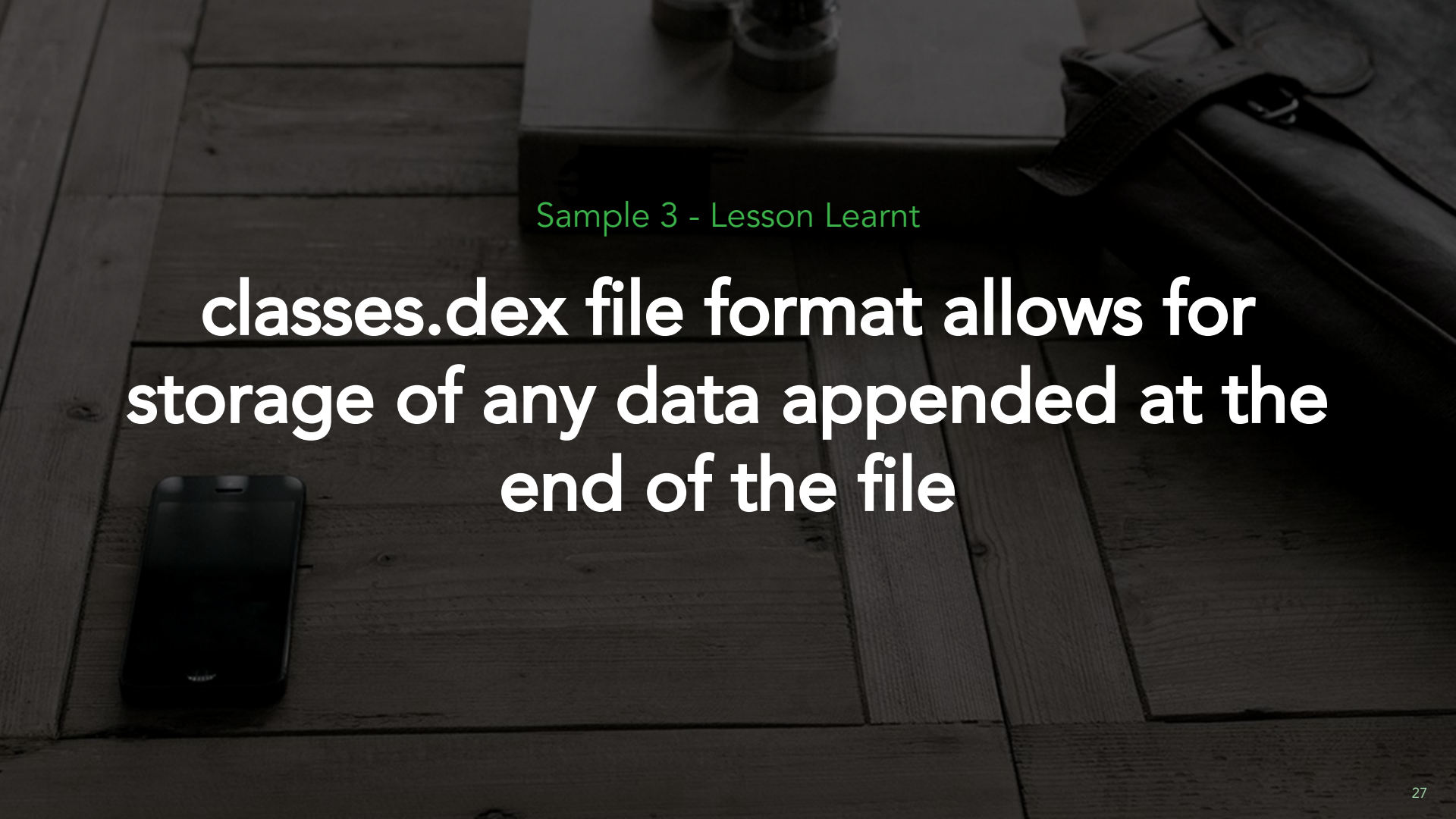
Operand: C7

Decimal / Hex

PK - magic bytes for a zip archive

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
50	4B	03	04	14	00	08	08	08	00	F3	A0	BF	4A	DA	BA	PK.....ó ¿JÚ°
A4	AE	C1	09	00	00	C1	0C	00	00	1C	00	04	00	61	73	¤®Á...Á.....as
73	65	74	73	2F	66	69	6E	64	62	75	74	74	6F	6E	32	sets/findbutton2
30	31	36	31	32	32	36	2E	6A	73	FE	CA	00	00	15	96	0161226.jsþÊ...–
35	A2	EC	08	0C	04	AF	E2	7C	02	33	05	1B	8C	99	99	5¢ì...¯â|.3..Œ™™
33	33	8E	99	4F	BF	EF	1F	40	49	4B	D5	25	CA	00	67	33Ž™O¿ï.@IKÕ%Ê.g
93	46	63	B0	08	1B	CC	F7	98	67	2E	3A	A6	AF	B4	D7	"Fc°..Ì÷˜g.:¦¯´×
7B	E0	37	C8	16	B9	96	9A	96	10	63	F4	3B	DE	4E	72	{à7È.¹–š–.cô;ÞNr
54	D7	23	71	16	FE	9C	25	A9	FA	C0	B6	96	D1	64	83	T×#q.þœ%©úÀ¶–Ñdƒ
47	0B	69	A7	43	F8	3C	0E	5D	C5	E7	2A	8D	37	57	75	G.i§Cø<.]Åç*.7Wu
DB	92	1C	D2	D7	4F	29	45	87	B8	33	55	53	15	27	DB	Û'.Ò×O)E‡¸3US.'Û
D7	AD	38	30	3A	C4	23	32	C0	A7	92	3A	9F	53	5D	26	×-80:Ä#2À§':ŸS]&
82	57	F5	33	89	E9	D2	8B	7E	FA	73	F0	65	AB	6F	0B	‚Wõ3‰éÒ‹~úsðe«o.

```
$ file *
xafecopy.apk:      Java archive data (JAR)
xafecopy.apk_out: Java archive data (JAR)
```



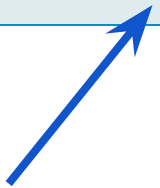
Sample 3 - Lesson Learnt

classes.dex file format allows for storage of any data appended at the end of the file

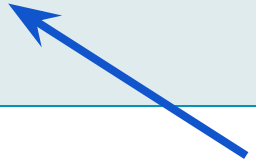
How does the code gets called?

Reflection is the answer

```
Class class = new DexClassLoader(this.getFilesDir().getPath() + File.separator +
"module.dex", this.getApplicationInfo().dataDir, null,
this.getClass().getClassLoader()).loadClass("com.appstatistics.Main");

class.getMethod("run").invoke(class.newInstance());
```



Method Name



Class Name

Putting it all together

- There is a number of known obfuscation methods used by threat actors today
- File extension is not a reliable indication of file type
- Steganography is not just theory - it is used in real malware
- There are always more ways to hide malicious code than where we expect it to reside

What's next?

Check Lookout website soon for a blog post on DressCode malware family evolution

https://blog.lookout.com/

(or just follow my LinkedIn, I'll make sure to share the link ;)

https://www.linkedin.com/in/areshetniak/

SHA1 sums of the reviewed samples for your reversing pleasure:

Sample 1 - 6c0da50bbf0524df35ffea87788e4bb8f276a6b4

Sample 2 - e8d2d6ee35a54ee6328f55d2dccbce3c213690d6

Sample 3 - e0f5f0816a1e41785e7b44cf4ac46bff6d557312

EVERYTHING IS OK