OPCDE_2018 challenge write-up

خطوات حل التحدي

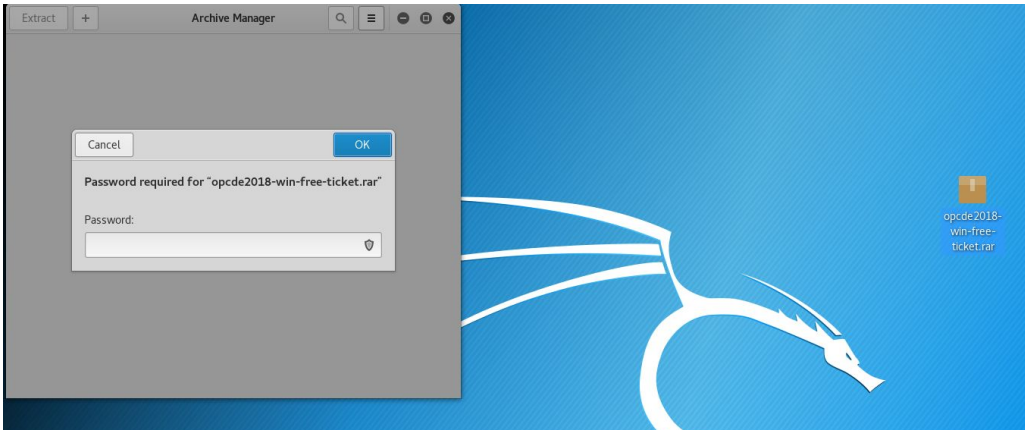
By @sasukeOurad

عرض أستاذنا المهندس محمد الدوب هذا التحدي في مجال الهندسة العكسية لتطبيقات الأندرويد.

هذا الشرح يفترض استخدامك لنسخة لينكس (كالي أو غيرها) مع معرفة بسيطة بالمجال بشكل عام. بالإمكان حل هذا التحدي باستخدام نظم تشغيل مختلفة (مثل ماك ونظام التشغيل الآخر ^_^)

وجب تنبيهه أن "باسم الله" تكتب بالألف عندما لا تكون البسملة كاملة :)

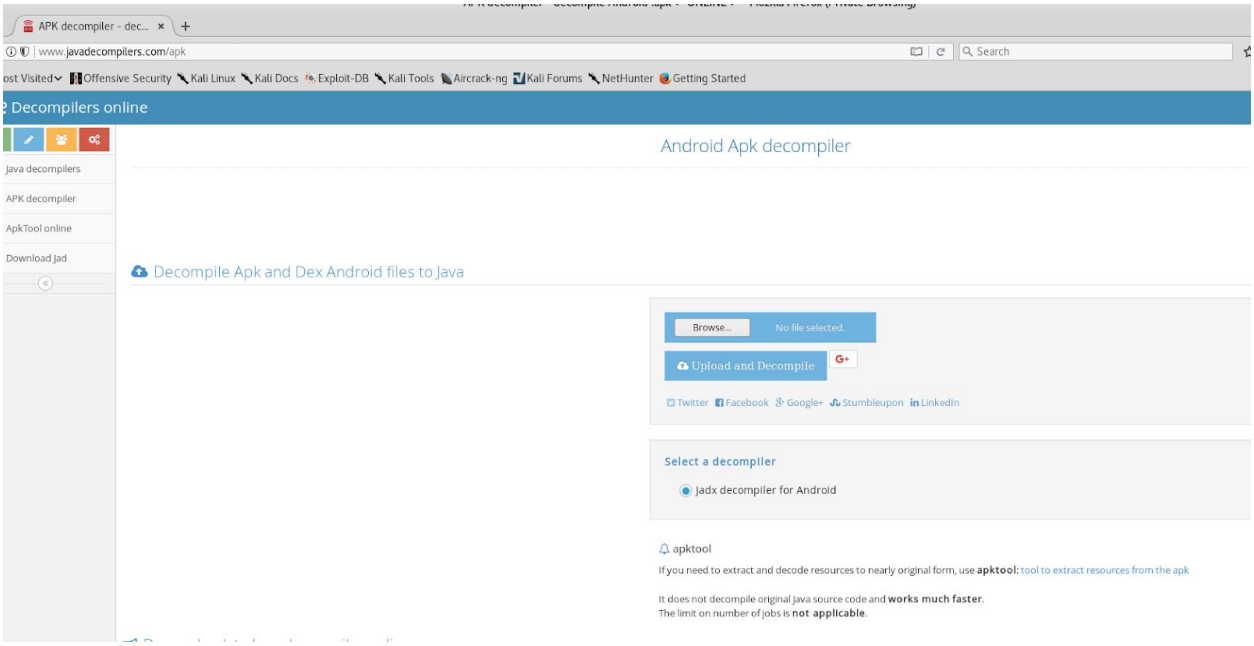
عند تحميل الملف المضغوط بصيغة rar ، بالإمكان فك الملف بالضغط عليه بالطريقة الاعتيادية. كلمة السر المطلوبة حينها هي opcde2018 كما قال أستاذنا محمد.



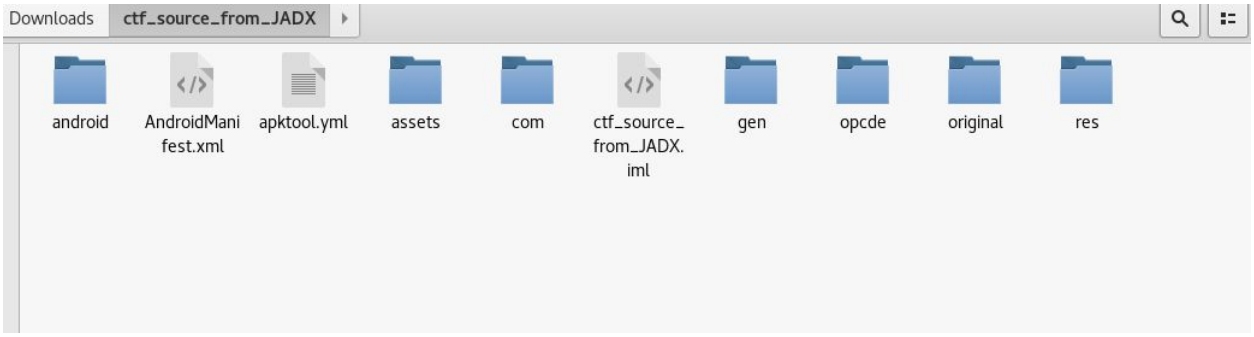
بفك الملف المضغوط، نحصل على تطبيق خاص بأندرويد باسم **ctf.apk .** فكرة التحدي الأساسي هي الهندسة العكسية لتطبيقات الأندرويد. إحدى ميزات تطبيقات الأندرويد لمحلل التطبيقات هي اعتمادها على لغة java بشكل أساسي. يسهل عكس وتحليل تطبيقات الjava في العادة. لذلك تتوفر عدة طرق لاستخراج الكود الأساسي لتطبيق الأندرويد. الطريقة المستخدمة في هذا التقرير ليست الوحيدة وقد لا تكون الأفضل، ولكن تم استعمالها للسرعة.

http://www.javadecompilers.com/apk

هو أحد المواقع المتميزة في استخراج كود تطبيق الأندرويد. كل ما عليك فعله هو تسليم ملف الapk واستلام الكود مضغوطاً



بعد تنزيل الملف المضغوط بصيغة zip وفك الملف .. نتحصل على الملفات التالية الخاصة بتطبيق الأندرويد اللازم تحليله.



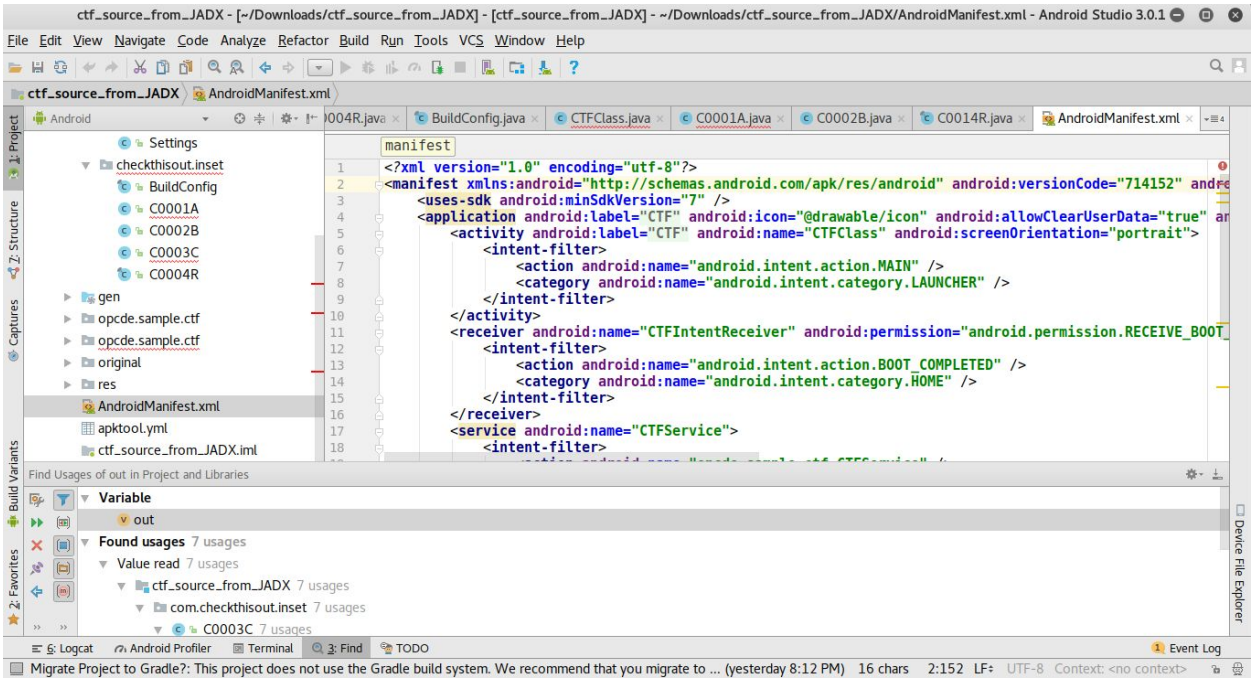
الخطوة القادمة تلتزم عمل شيئين في نفس الوقت .. التحليل اليدوي والأوتوماتيكي للملفات بالإضافة إلى تشغيل البرنامج في بيئة محكمة الإغلاق.

التحليل اليدوي (الساكن)

هذه الخطوة تستدعي قراءة شاملة وعامة للكود .. في رأيي المتواضع، عند التعامل مع ملفات الأندرويد يفضل فتح الملفات في برنامج يعطيك نظرة المصمم أو المبرمج لهذا البرنامج. لذلك أنصح باستخدام android studio أو أي برنامج تفضله لبرمجة تطبيقات الأندرويد. في android studio اتجه إلى

File > New > import project

واختر الملف الذي يحتوي على الكود الذي نزلناه في الخطوة السابقة
(ثم next next next) ^_^ لتحصل على شيء مشابه للصورة التالية



تبدأ رحلة البحث من ملف AndroidManifest.xml . يحتوي هذا الملف على تفاصيل تطبيق الأندرويد من حيث الاسم والصلاحيات ونقطة انطلاق التطبيق Main activity. في هذا التحدي، نقطة انطلاق التطبيق هي ملف opcde.sample.ctf.CTFClass
ملفٌ آخر مثيرٌ للاهتمام في هذا التحدي هو res/values/strings.xml
بسبب احتوائه على النصوص المستخدمة في هذا التطبيق، يمكن الخروج منها ببعض المعلومات المفيدة لاحقاً. على سبيل المثال، info_first_use يشرح أن هذا التحدي تم اقتباسه من برنامج ضار حقيقي، ويحذر من تنزيل البرامج الضارة أو الغير موثوق بها (كلام سليم).

```
tring name="service_start">Service is starting...</string>
tring name="service_idle">It looks like i have nothing to do :-)</string>
tring name="db_no_boot_data">Oops, found no data on boot. Using default data.</string>
tring name="service_shutdown">Service finnished</string>
tring name="init">First app init</string>
tring name="cache_size_error">Cache size is illegal. Using default value of 128kb.</string>
tring name="menu_info">Info</string>
tring name="menu_exit">Exit</string>
tring name="info_first_use">This CTF is based on a piece of malware, never install it on a real device! Try to find the flag!</string>
tring name="info_button">Try to find the flag!.</string>
tring name="ok_label">Ok</string>
tring name="license">License</string>
tring name="cache_size_info">Sorry, this cache size is not allowed. Please use a value between 128 up to 8192 Kb.</string>
tring name="license_button">Ok</string>
tring name="no_devices">No SD-Cards found!</string>
```

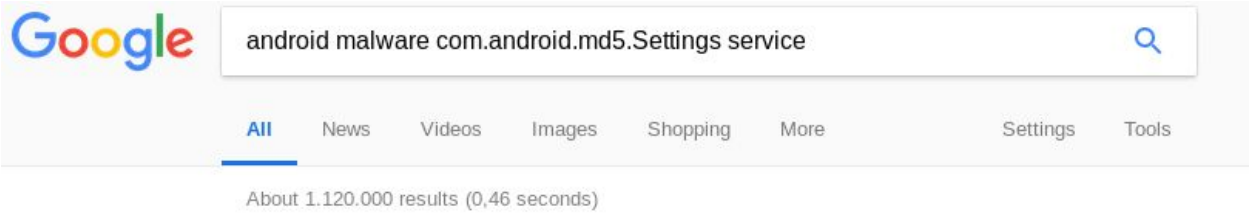
نقطة أخرى مُلاحظة أثناء قراءة AndroidManifest.xml هي وجود android service باسم:
com.android.md5.Settings
الandroid service هو برنامج يعمل في خلفية النظام بحيث لا تتم ملاحظته من قبل المستخدم.

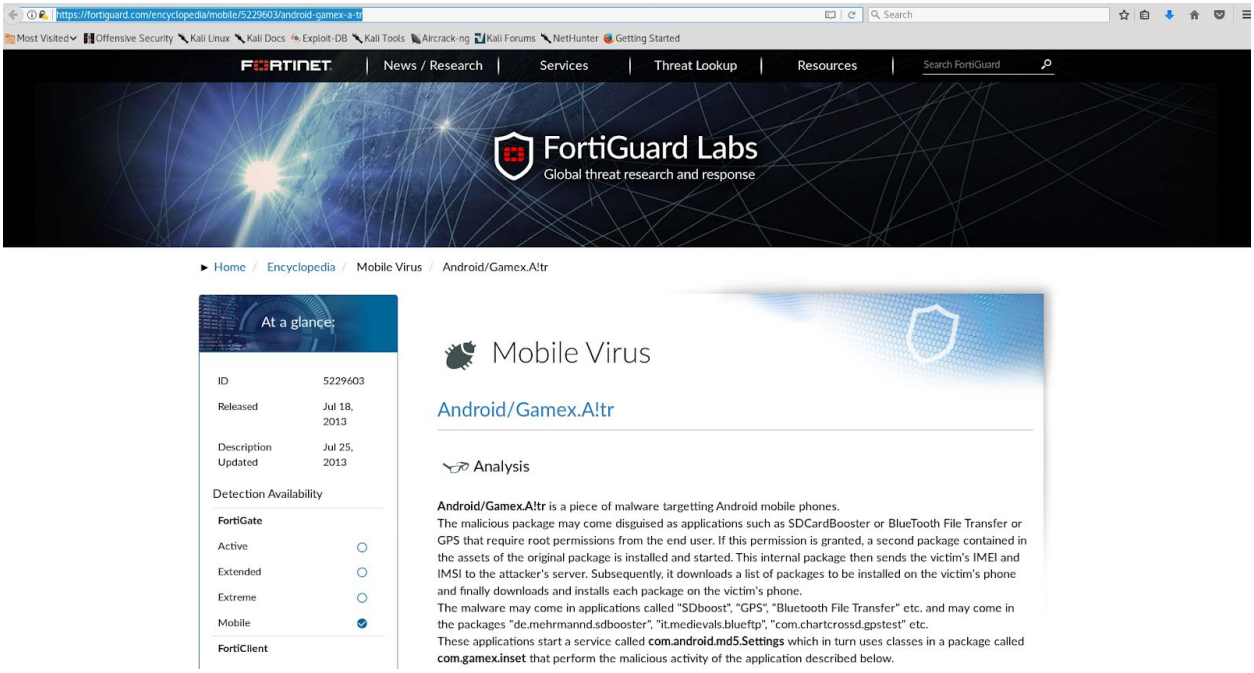
```
        <service android:name="com.android.md5.Settings" android:enabled="true" />
    </application>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

تشغيل البرنامج في بيئة محكمة الإغلاق.

أحد أهم وأخطر الخطوات (على المحلل أو الباحث) أثناء تحليل البرامج الضارة. كما ذُكر سابقاً، لا ينصح باستخدام هذا الأسلوب إلا باتزان وحكمة لاجتناب تضرر أجهزة الباحث. من الممكن تجهيز أجهزة افتراضية باتصال مموه بالانترنت لدراسة سلوك البرنامج.
في هذا التحدي، لم تكن الحاجة ملحة لاستخدام هذا الأسلوب. ولكنه المفضل لي فلِمَ لا ^_^
أنصح باستخدام genymotion كـ تطبيق للهاتف الافتراضي على الحاسوب، فصل الاتصال بالانترنت وربطه عن طريق proxy عند الحاجة كـ burpsuite. توفر genymotion هواتف بـroot مما يمنحك التحكم الكامل في الهاتف. بعد ذلك، من الضروري تنصيب xposed للتحليل الديناميكي للهاتف وكل ما يعمل عليه والتحكم في البرامج والنظام. يمكنك xposed من وضع proxy داخلي وإخفاء حصولك على root من البرامج المختلفة.
أكرر، بالإمكان حل هذا التحدي بدون اللجوء لهذه التقنية. ولكن لم لا ^^
بتشغيل البرنامج، نحصل على هذا الشكل الجميل:

فور تشغيل البرنامج يقوم بطلب الحصول على أعلى الصلاحيات root. باستخدام المعلومات التي تم تجميعها إلى الآن، يمكن ببحث صغير على الموقع المفضل للجميع google العثور على البرنامج الضار الذي يحاكيه هذا التحدي:





البرنامج الضار يسمى Gamex.A!tr وطبقاً للوصف فإن صورة الشعار تحتوي على برنامج أندرويد آخر بصيغة مشفرة، يقوم البرنامج بفك الشفرة فور حصوله على صلاحيات root.

هممم لم لا نلقي نظرة على شعار التطبيق الجميل؟؟؟؟

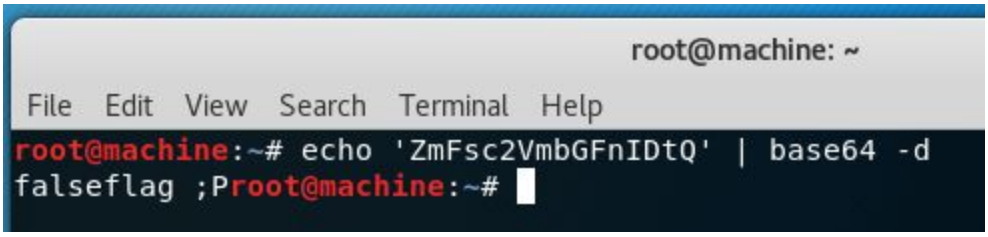
لم أجد أي شيء مختلف في الشعار 💔 ... ولكن، صورة بمسمى assets/ctf.png أثارت اهتمامي. عند محاولة فتح الصورة نفاجأ بأنها لا تعمل. بعد البحث عن وظيفتها في android studio عن طريق ضغطة يمنى بالفأرة ثم find usages ، نجد أنه تم استخدامها في com/checkthisout/inset/C0003C.java

```
try {
    in = this.context.getAssets().open( fileName: "ctf.png");
    if (file.exists()) {
        file.delete();
    }
    file.createNewFile();
    FileOutputStream out = new FileOutputStream(file);
    try {
        byte[] buffer = new byte[1000];
        byte[] buf = new byte[1000];
        while (true) {
            int read = in.read(buffer);
            if (read == -1) {
                break;
            }
            for (int i = 0; i < read; i++) {
                buf[i] = (byte) (buffer[i] ^ 18);
            }
            out.write(buf,  off: 0, read);
        }
```

ما يقوم به هذا الجزء من البرنامج هو أخذ byte من الصورة وعمل xor مع العدد 18 وتكرار ذلك حتى تنتهي الصورة. لنقم بعمل مشابه، لنأمل الحصول على الصورة الأصلية ، عفواً سأستخدم python ^_^

ملحوظة: يحتوي com/checkthisout/inset/C0003C.java على رموز بصيغة base64 وعند تحويلها على أمل الحصول على السر نفاجأ بمزحة من مصمم التصميم >.<

```
new File(this.f1p).delete();
this.context.sendBroadcast(new Intent( action: "ZmFsc2VmbGFnIDtQ"));
this.context.stopService(new Intent(this.context, Settings.class));
```

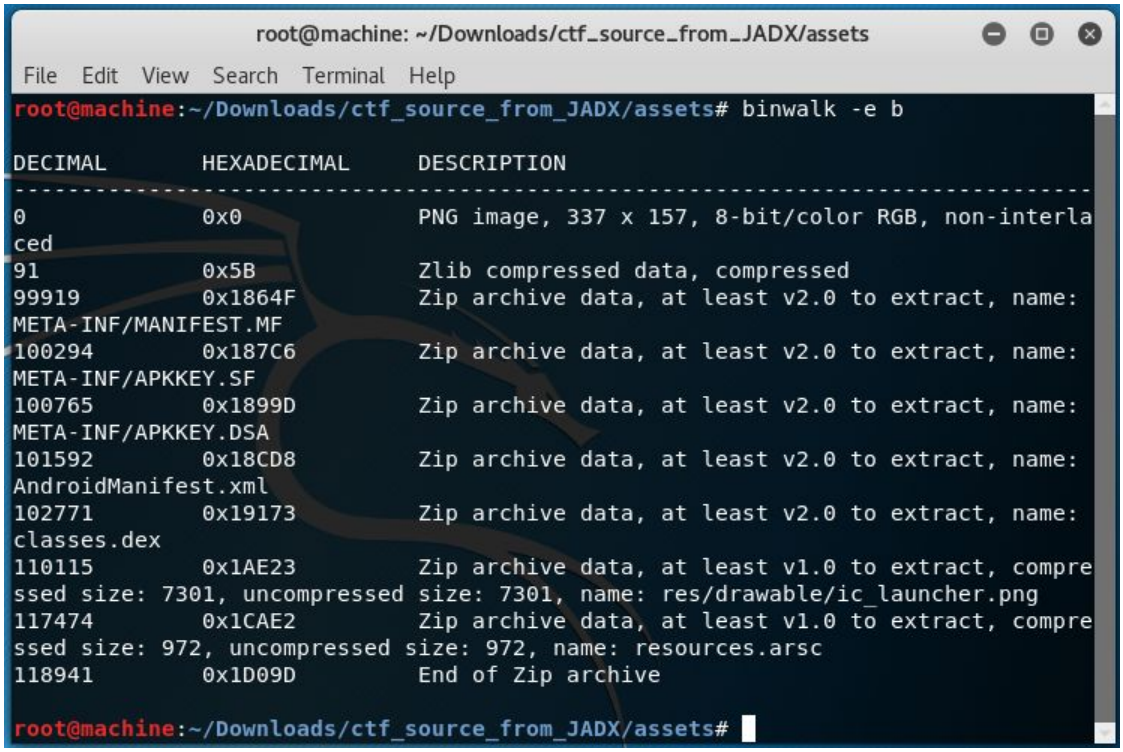


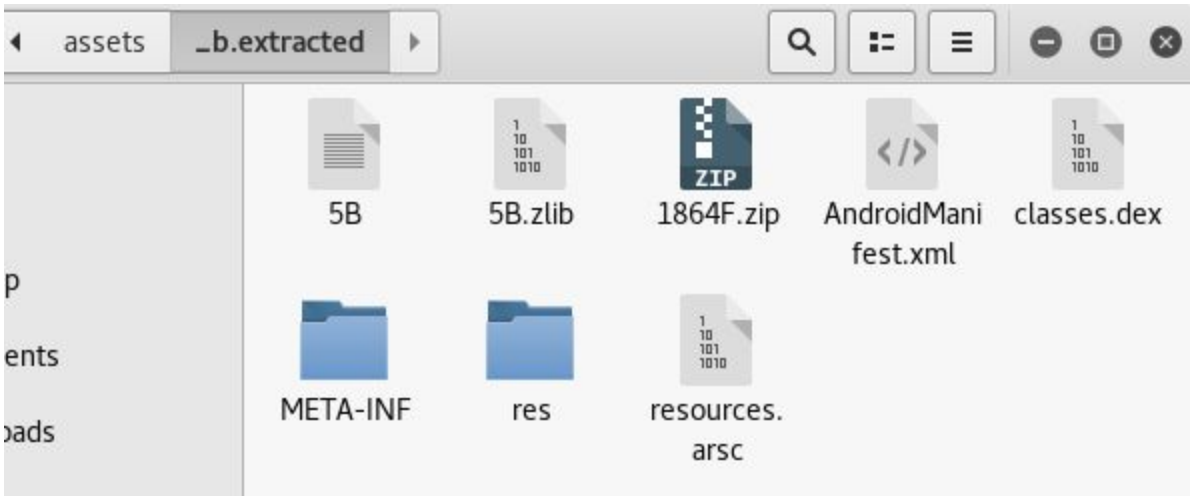
عدنا: البرنامج الصغير التالي يقوم بعمل الخطوة المذكورة مسبقاً: أخذ byte من الصورة وعمل xor مع العدد 18 وتكرار ذلك حتى تنتهي الصورة. لاحظ أن العدد 18 بنظام decimal هو نفسه 0x12 في نظام الـhexadecimal.

الملف الناتج سمّي b إبداعاً وابتكاراً D: .. إذا كانت حساباتي صحيحة فستكون الصورة الأساسية خلف ctf.png

وهي كذلك ^_^ .. الآن وبناءً على قراءتنا السابقة للبرنامج الضار Gamex.A!tr نظن بأن هذه الصورة ليست صورةً عادية بل تحتوي على برنامج ضار آخر مخبأ بداخلها.

للبحث بداخل الملفات المخبأة، أنصح دائماً بالابتداء بـbinwalk وهو برنامج ممتاز للبحث عن البرامج أو الملفات المدمجة داخل ملفات أخرى. نستخدم binwalk -e b لاستخراج أي ملفات أخرى متواجدة داخل الصورة. ويالللمفاجأة ~_~

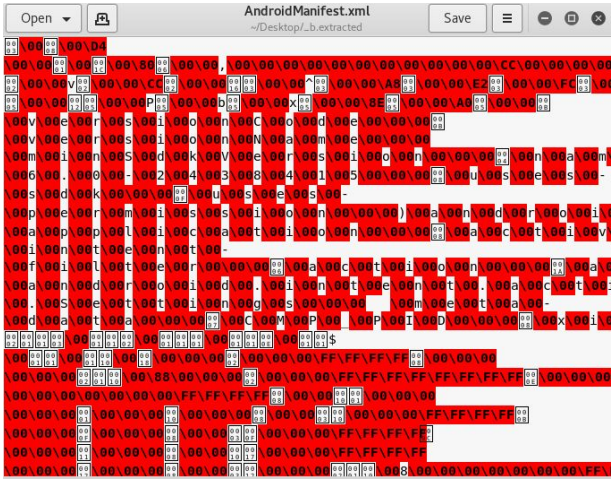




كل هذه الملفات كانت مخفاة بداخل الصورة البريئة ^_^.

من النظرة المبدأية فإن أهم الملفات الموجودة هنا هي AndroidManifest.xml و classes.dex

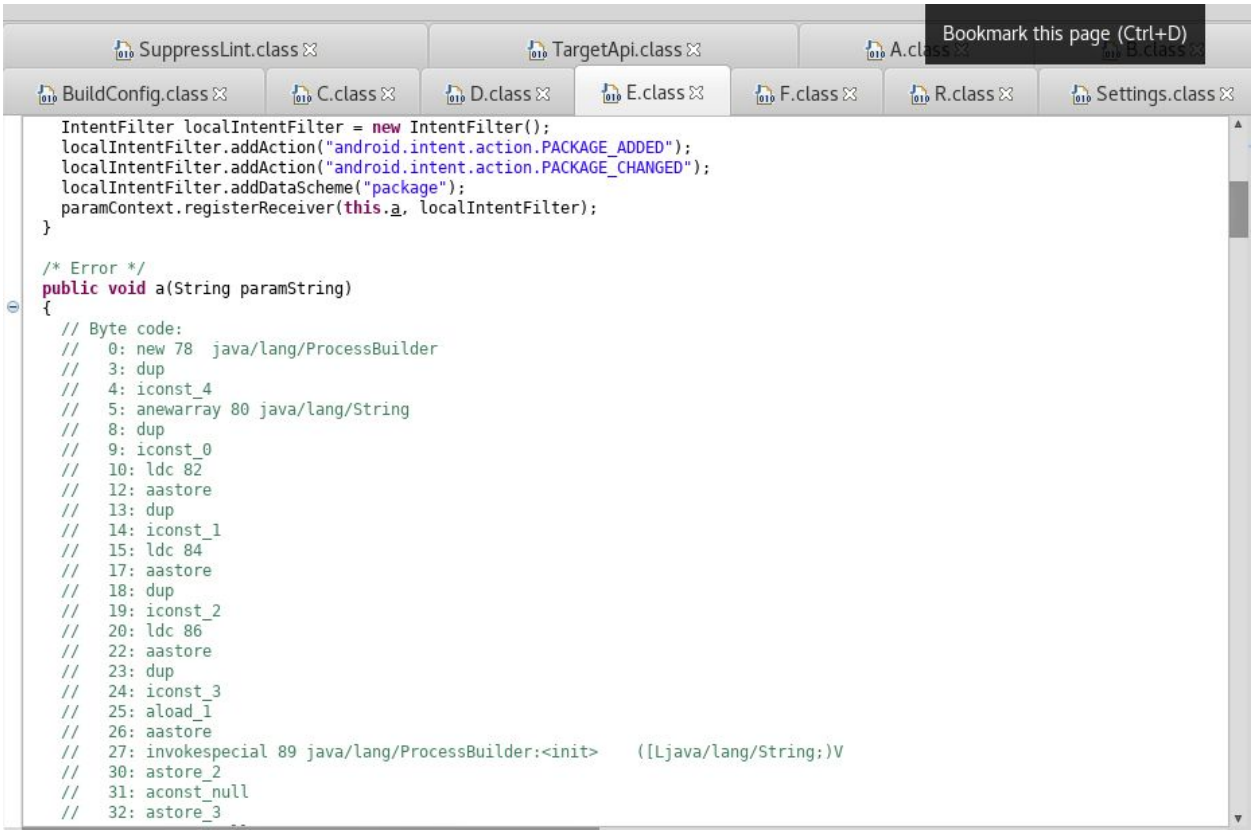
وجود هذين الملفين يدل على وجود تطبيق أندرويد. المشكلة هي وجودهما في صورة غير قابلة للقراءة.



لكل مشكلة حل .. لتنقية AndroidManifest.xml بالإمكان استعمال

https://github.com/ytsutano/axmldec

وللحصول على ملفات الأندرويد في classes.dex يمكن استعمال dex2jar لتحويل الملف إلى كودات جاهزة للقراءة ثم قراءة الملفات باستخدام android-studio كما فعلنا مسبقاً أو jd-gui. المشكلة هي أن dex2jar لم يكن موفقاً في تحويل جميع ملفات classes.dex إلى كودات قابلة للقراءة. كما يتضح في الشكل التالي، فإنه لم يوفق في التحويل وأصدر error في ملف E.class



وبالتالي تمت الاستعانة بصديقنا القديم

http://www.javadecompilers.com/apk

الذي قام بالمهمة كما يجب.

لنقم بفحص الملفين فحصاً أولياً، ولنبدأ بالملفات التي فشل dex2jar بقراءتها. لا شك أنها تحتوي سراً

ملحوظة أخرى: يتواجد سر مزيف آخر في هذه الملفات للقضاء على الآمال ولكن لا داعي لذكره، لنتجاهله ^_^

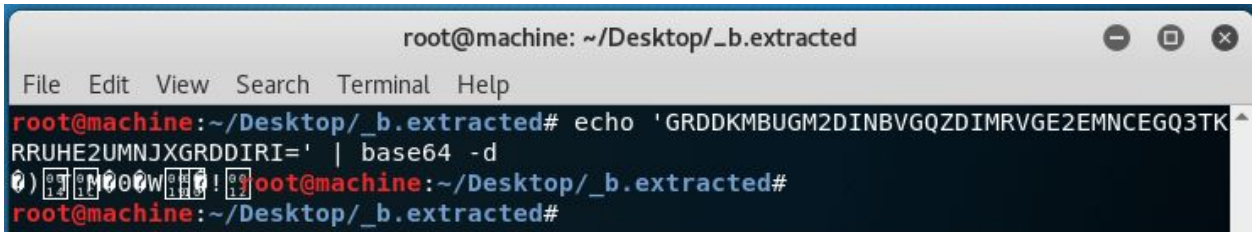
الملف المسمى E.class عن طريق dex2jar تغيرت تسميته إلى C0005E.java وهو شيء طبيعي عند اختلاف الأدوات المستخدمة في الهندسة العكسية.

C0005E.java
~/Desktop/_b.extracted/crap/classes.dex_source_from_JADX/com/android/setting

```
L extends BroadcastReceiver {
) {


void onReceive(Context context, Intent intent) {
((intent.getAction().equals("android.intent.action.PACKAGE_ADDED") ||
n().equals("android.intent.action.PACKAGE_CHANGED")) && "com.android.update".equals(intent.getDataString().substr
 context.sendBroadcast(new Intent("akjgikurhnfjghfkj"));
 new File(C0005E.this.f6p).delete();
 context.stopService(new Intent(context, Settings.class));



5E(Context context) {
 = context;
ilter filter = new IntentFilter();
addAction("android.intent.action.PACKAGE_ADDED");
addAction("android.intent.action.PACKAGE_CHANGED");
addDataScheme("package");
.registerReceiver(this.f4a, filter);


 run() {
tion e;
le th;
le = new File(this.f6p);
ream in = null;
outStream fileOutputStream = null;

= this.f5b.getAssets().open("GRDDKMBUGM2DINBVGQZDIMRVGE2EMNCEGQ3TKRRUHE2UMNJXGRDDIRI=");
(file.exists()) {
 file.delete();
```

حصلنا على هذا الرمز الجديد. فلنر إن كان السر المختبئ خلف base64

GRDDKMBUGM2DINBVGQZDIMRVGE2EMNCEGQ3TKRRUHE2UMNJXGRDDIRI=



ليس سوى رموز بلا معنى >.< ما الحل .. هل وصلنا باباً مسدوداً ؟؟

هو كذلك .. إن ظننت أن العالم لا يحتوي على شيء سوى base64 .. ألم تسأل نفسك كيف وصلنا إلىbase64 ؟ ألم نمر بتقنيات ذات رقم أصغر؟ أليس من الأولى الخروج من الصندوق وتجربة base32 ؟؟

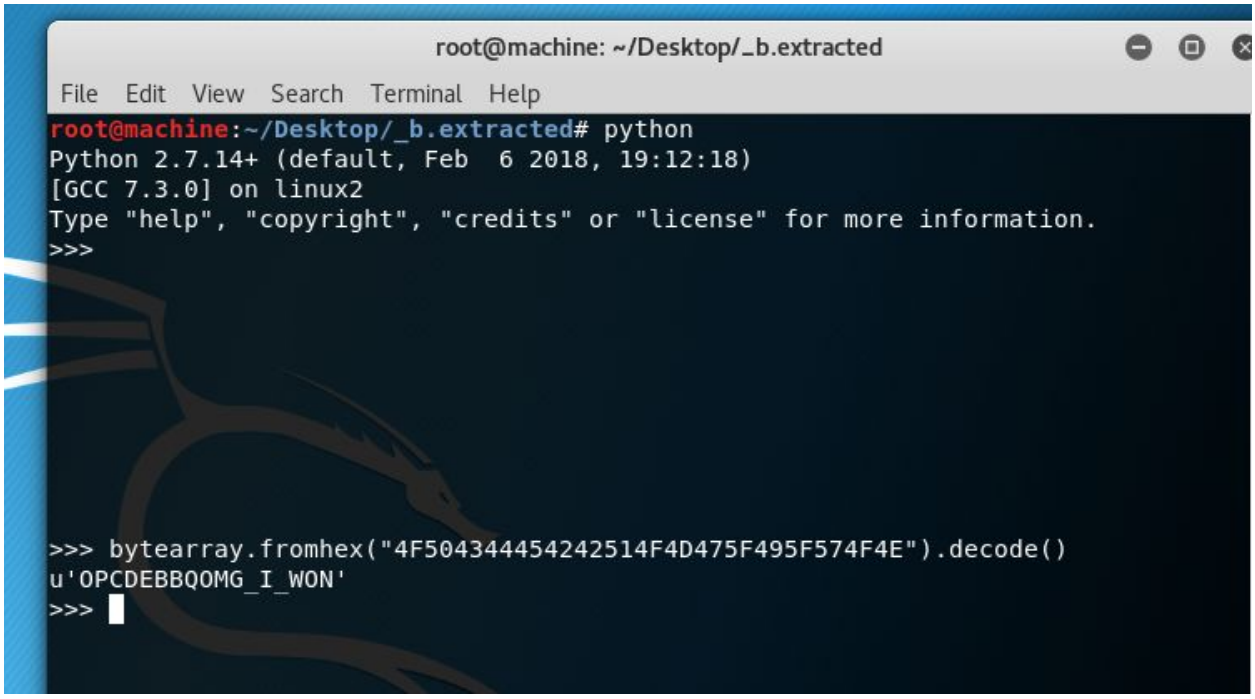
بتجربة base32 نكتشف الآتي:

```
root@machine:~/Desktop/_b.extracted#
root@machine:~/Desktop/_b.extracted# echo 'GRDDKMBUGM2DINBVGQZDIMRVGE2EMNCEGQ3TK
RRUHE2UMNJXGRDDIRI=' | base32 -d
4F504344454242514F4D475F495F574F4Eroot@machine:~/Desktop/_b.extracted#
root@machine:~/Desktop/_b.extracted#
```

أهاااا

4F504344454242514F4D475F495F574F4E

ألا يبدو ذلك كنظام ترميز hexadecimal ؟
فلنر .. لنحوله إلى حروف ASCII



وهو كذلك **OPCDEBBQOMG_I_WON**

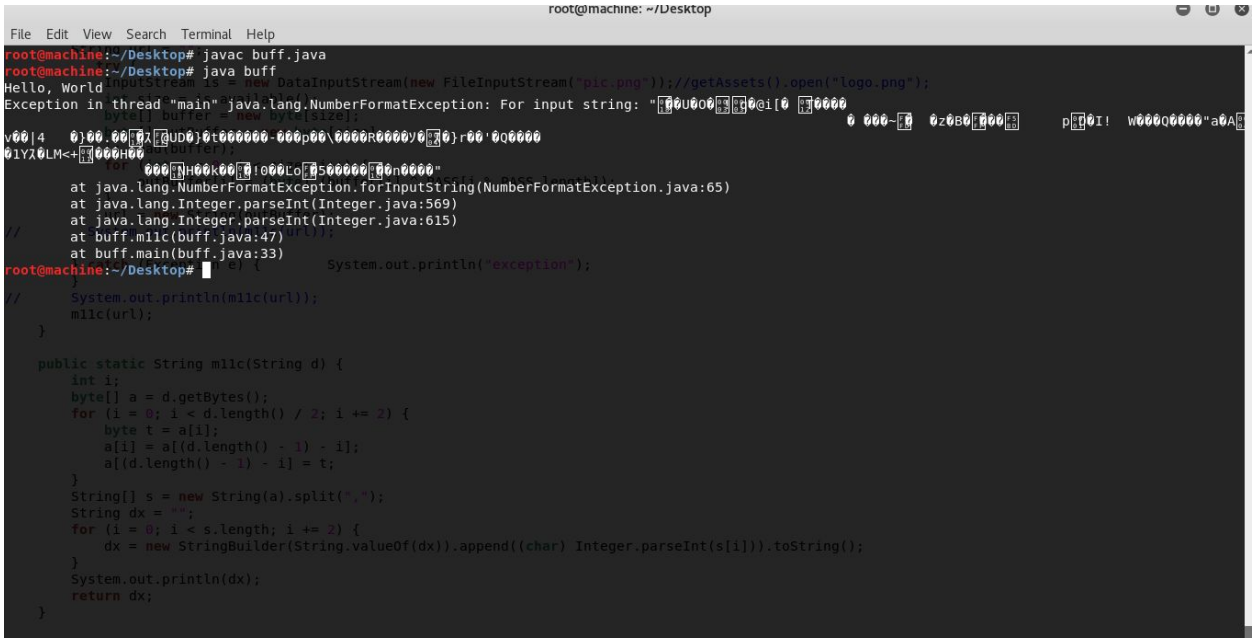
حصلنا على السر بحمدالله ^_^ شكراً أستاذنا محمد على هذا التحدي الجميل ^_^

طبعاً لم تنتهي الحكاية ها هنا كما ظننت .. أستاذنا محمد شرح ووضح أن الفائز سيجد رابطاً يدخل فيه اسمه وايميله وحسابه وهذا الملف:

لكن بتسرعي المعهود نسيت وأضعت وقتاً ثميناً في الاحتفال. ثم عدت خائب الوفاض أبحث عن الرابط..
أولاً انجذبت للكود المسمى geturl لأن "الكتاب يبان من العنوان" ولكن الصورة "logo.png" ليست موجودة من الأساس

```
private String getUrl() {
    String url = "";
    try {
        InputStream is = getAssets().open("logo.png");
        int size = is.available();
        byte[] buffer = new byte[size];
        byte[] outBuffer = new byte[size];
        is.read(buffer);
        for (int i = 0; i < size; i++) {
            outBuffer[i] = (byte) (buffer[i] ^ PASS[i % PASS.length]);
        }
        url = new String(outBuffer);
    } catch (Exception e) {
    }
    return m11c(url);
}

public String m11c(String d) {
    int i;
    byte[] a = d.getBytes();
    for (i = 0; i < d.length() / 2; i += 2) {
        byte t = a[i];
        a[i] = a[(d.length() - 1) - i];
        a[(d.length() - 1) - i] = t;
    }
    String[] s = new String(a).split( s: ",");
    String dx = "";
```

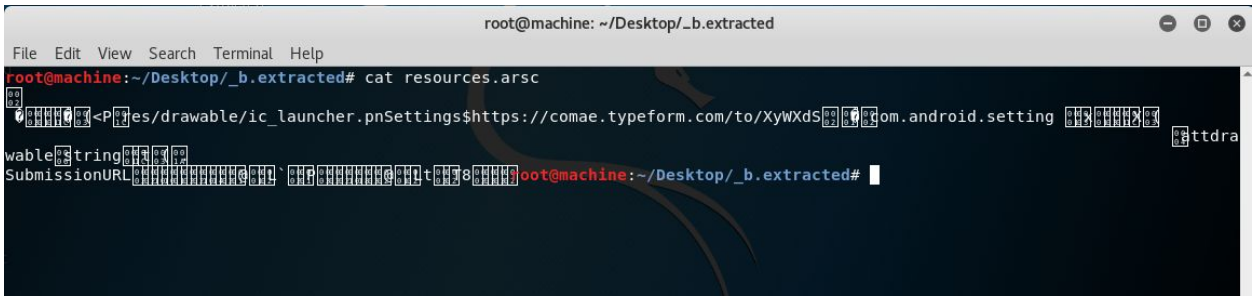
ثم حاولت إعادة تصميم الكود بـjava ... ولكن منذ متى طاوعتنا هذه اللغة 💔 .. بل أثرتنا بالاعتراض والexceptions.



ثم لمعت عيناي حينما رأيت DSA وظننت أني وجدتُ مفتاح التشفير الذي سيجعل حياتي أسهل لكنها تعقدت أكثر.

لأجد الحلّ في النهاية هو البساطة .. ما أغفل عنه دائماً عند تعقيدي للأمور .. resources.arc هو ملف خاص بتطبيقات الأندرويد ويحتوي عادةً على معلومات مفيدة للتطبيق والمحللين على حدّ سواء .. قراءة بسيطة للملف تنبيك عن رابط مخفي هنا وهناك:
https://comae.typeform.com/to/XyWXdS

وهو كذلك ^_^
شكرا مرة أخرى .. أبدعت أستاذنا



أرجو أنكم استفدتم .. ولأي ملاحظات أو أسئلة تفضلوا.. دعواتكم ^_^

@sasukeOurad