CTF

طريقة تحليل واكتشاف Flag لمسابقة OPCDE

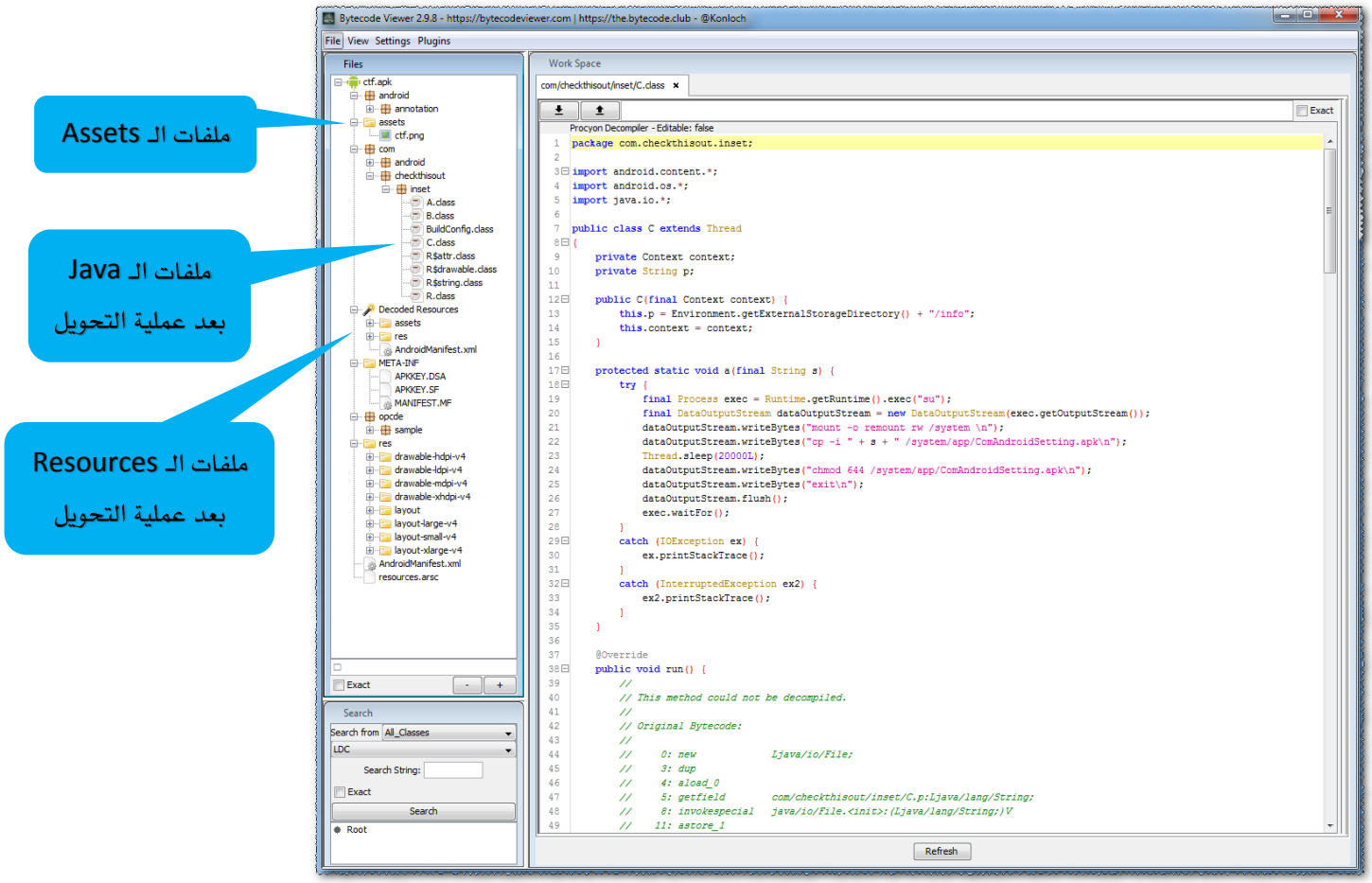
الأدوات المستخدمة في عملية التحليل Reverse Engineering

اسم الأداة	الغرض
BytecodeViewer	تحويل apk إلى لغة Java وفك ترميز الـ Resources وعرضها بشكل GUI https://github.com/Konloch/bytecode-viewer/releases
010 Editor	التعامل مع الملفات بصيغة Hexadecimal والتعرف على صيغ الملفات Templates https://www.sweetscape.com/010editor
Base32 Decoder	فك ترميز الأساس ٣٢ Base32 https://emn178.github.io/online-tools/base32_decode.html
WinRAR, WinZip, 7Zip	للتعامل مع ملفات APK والتي هي نفسها صيغة ZIP وأي من هذه الأدوات تفي بالغرض المطلوب.

تحليل ملف CTF.apk

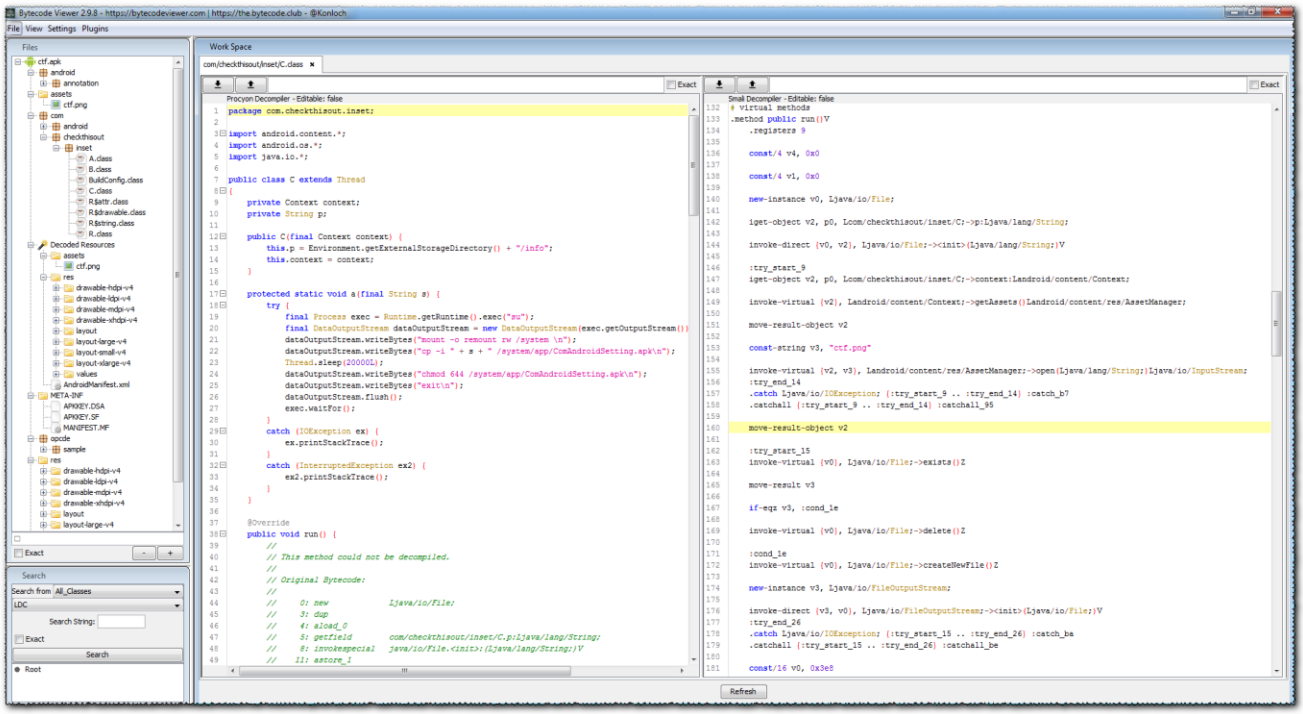
تحويل (Decompile) الملف التنفيذي للأندرويد من صيغة APK إلى ملفات اللغة الأصلية التي تم كتابة التطبيق به. وخلال هذا العملية يتم تحويل الملف التنفيذي classes.dex إلى ملفات Java قدر الإمكان وقد تواجه بعض المصاعب والتي يتم وضعها من قبل المطور بطريقة متعمدة لتعقيد وخداع المحلل ومنها وضع خدع تفشل عملية التحويل وبالتالي يصعب من عملية فهما.

في البداية يتم تنفيذ الأداة BytecodeViewer ويتم فتح الملف ctf.apk:



بعدها يتم معاينة جميع ملفات الـ Java والبحث عن الهدف المنشود وهو رابط الموقع وكذلك الـ Flag وهذه عملية تحتاج صبر وطولة بآل وتحليل متعمق لفهم وربط الملفات مع بعضها البعض.

نجد أن بعض الدوال لم يتم فك تحويلها وذلك لوجود موانع Anti-static Analysis والغرض منها تعقيد العملية على المحلل. وعلى سبيل المثال في ملف com.checkthisout.inset.C نجد أن دالة run لم يتم فك تحويلها بشكل جيد. وهي جديرة بالاهتمام وتحليلها بشكل متعمق لفهم طريقة عملها.



في هذه الحالة يتم فتح شاشة جديدة لتحويل الـ class إلى Smali لفهم الغرض من هذه الدالة run. من خلال التحليل نجد أن الدالة تقوم بالتالي:

١) فتح ملف ctf.png من دليل الـ Assets

٢) قراءة 1000 بايت وتخزينها في مصفوفة array

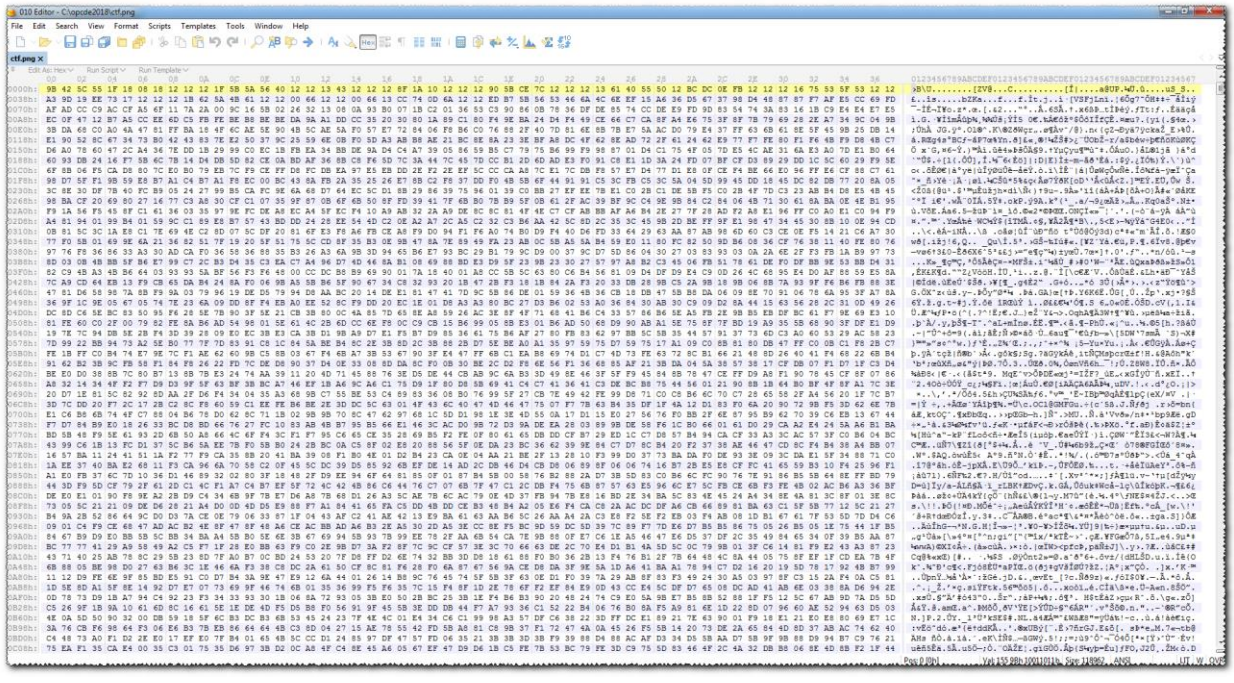
٣) تتم عملية xor لهذه المصفوفة بقيمة ثابتة لكل بايت وهي 0x12 وتتم هذه العملية لكامل الملف ctf.png

٤) ويتم تخزينها في ملف بعنوان: Environment.getExternalStorageDirectory() + "/info"

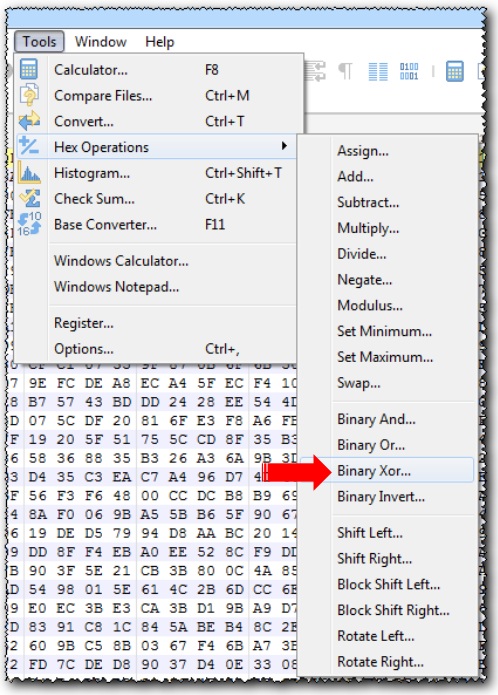
٥) يتم استخلاص ملف ctf.png من الملف ctf.apk بواسطة WinRAR أو غيرها من الأدوات

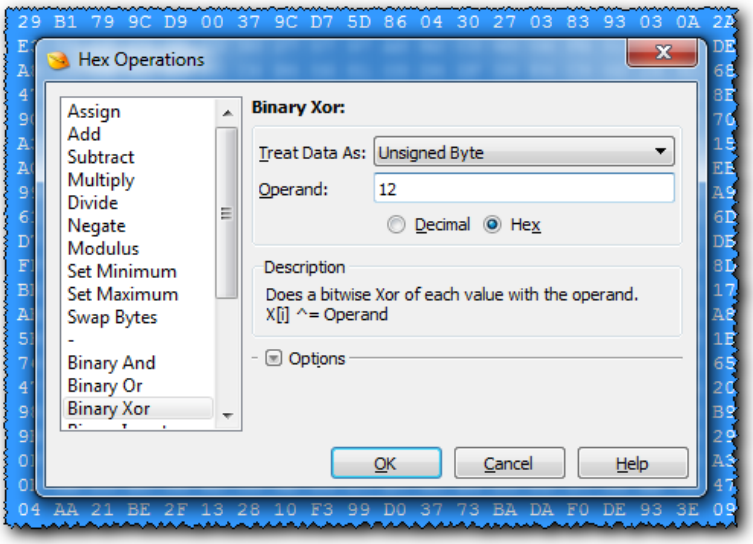
٦) يتم فتح الملف ctf.png بواسطة الأداة 010 Editor

٧) محتويات الملف ctf.png قبل عملية الـ xor

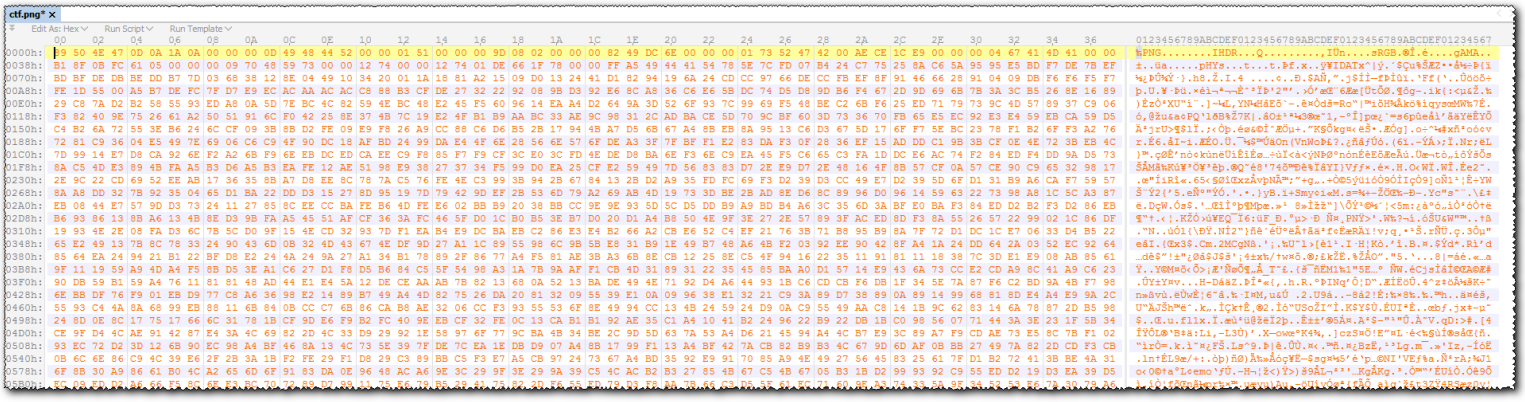


٨) يتم تحديد جميع محتويات الملف (Ctrl-A) ومن ثم استخدام خاصية xor لفك تشفير ملف ctf.png



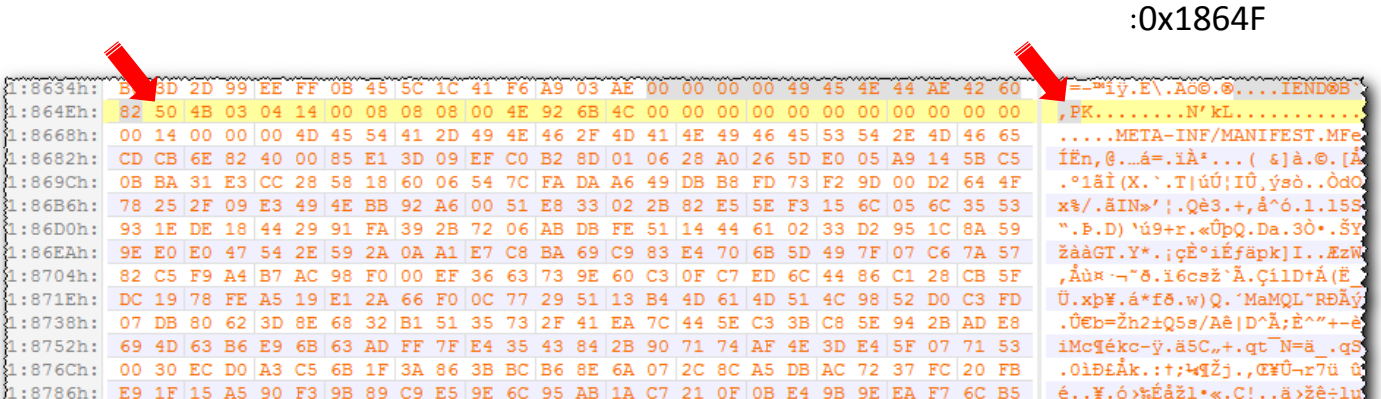


٩) الملف الناتج بعد عملية فك تشفير محتويات ملف ctf.png



١٠)ويظهر جليا أن الملف قد تم فك تشفيره بنجاح وظهور بيانات بدايات صيغة ملف PNG في بدايته

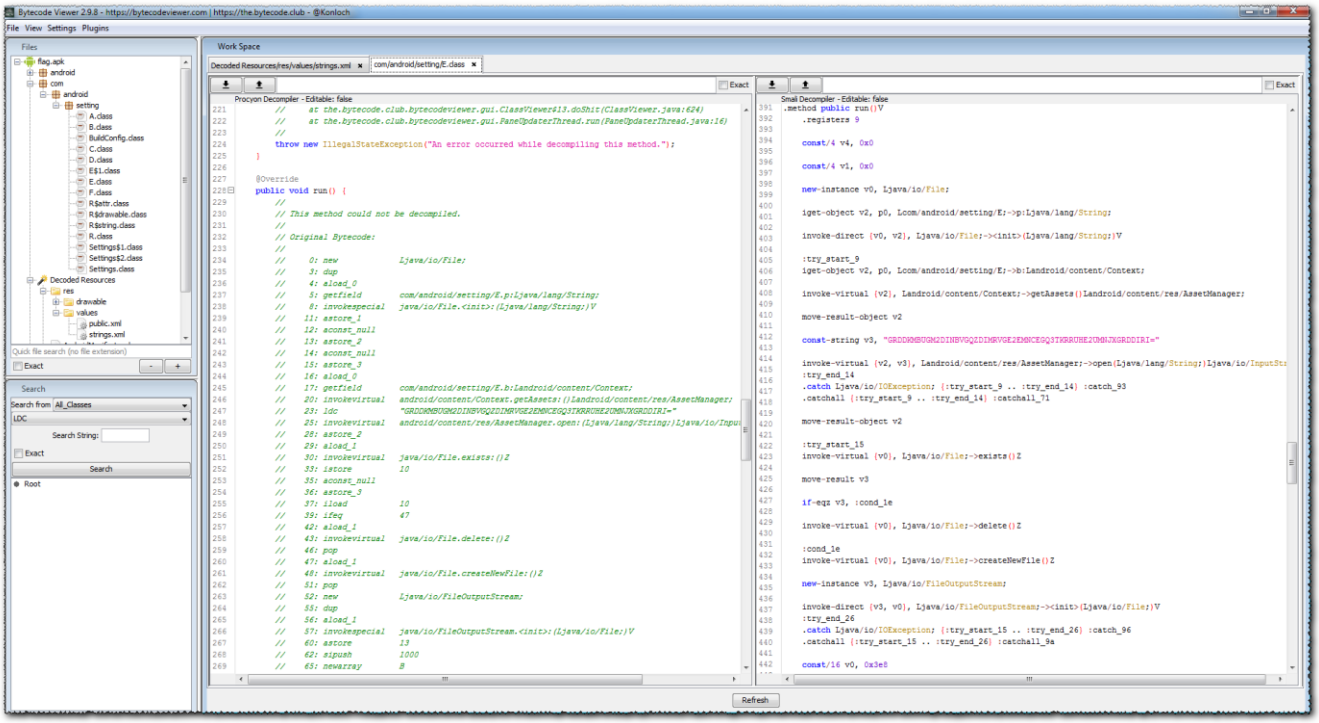
١١)بعد عملية تحليل الملف نجد أنه تم وضع ملف APK في نهاية ملف PNG وذلك لتظليل المحلل في العنوان :0x1864F



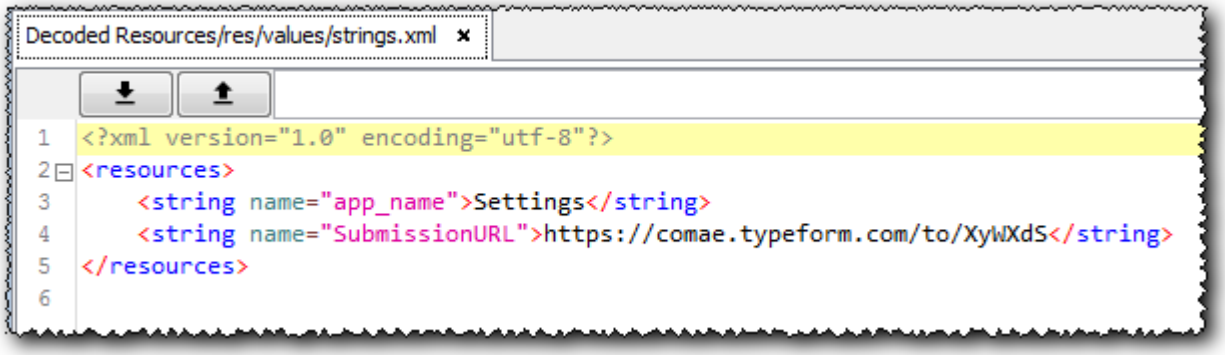
١٢)يتم تحديد ملف APK المراد تصديره إلى ملف جديد من العنوان المذكور سابقا إلى نهاية الملف ومن ثم يتم عمل Ctrl-C وفتح ملف جديد Ctrl-N باسم Flag.apk ومن ثم Ctrl-V. قد تحتاج أن تضيف بايت واحد 0x00 في أخر الملف حتى يكون ملف الـ APK سليم.

تحليل ملف Flag.apk

في البداية يتم تنفيذ الأداة BytecodeViewer ويتم فتح الملف Flag.apk:

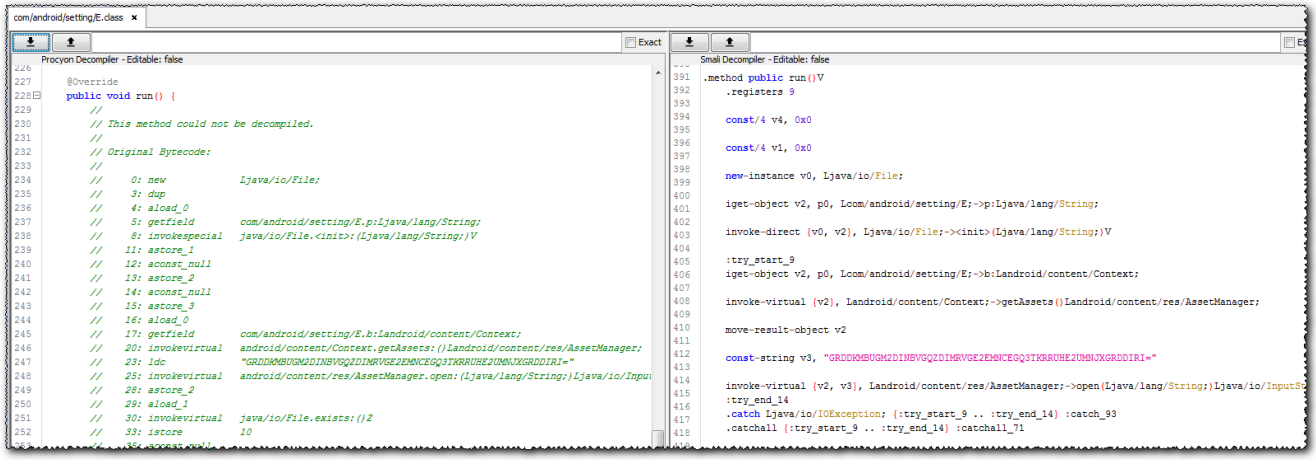


وبعد عملية فحص شاملة لمحتويات الملفات المضمنة في داخل APK نجد أن ملف strings.xml يحتوي على موقع الـCTF المطلوب:

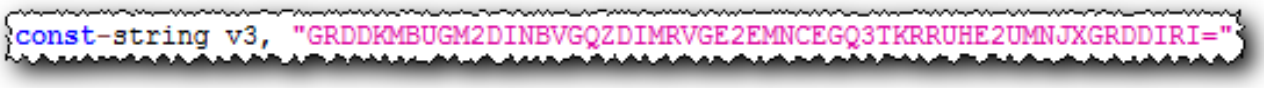


https://comae.typeform.com/to/XyWXdS

ونأتي إلى المهمة التي قد تكون الأصعب وهي اكتشاف الـ CTF، فبعد عملية بحث طويلة لكل الملفات نجد أن ملف com.android.setting.E يحتوي على دالة run قد تم وضع خدعة لتعقيد عملية تحويلها.

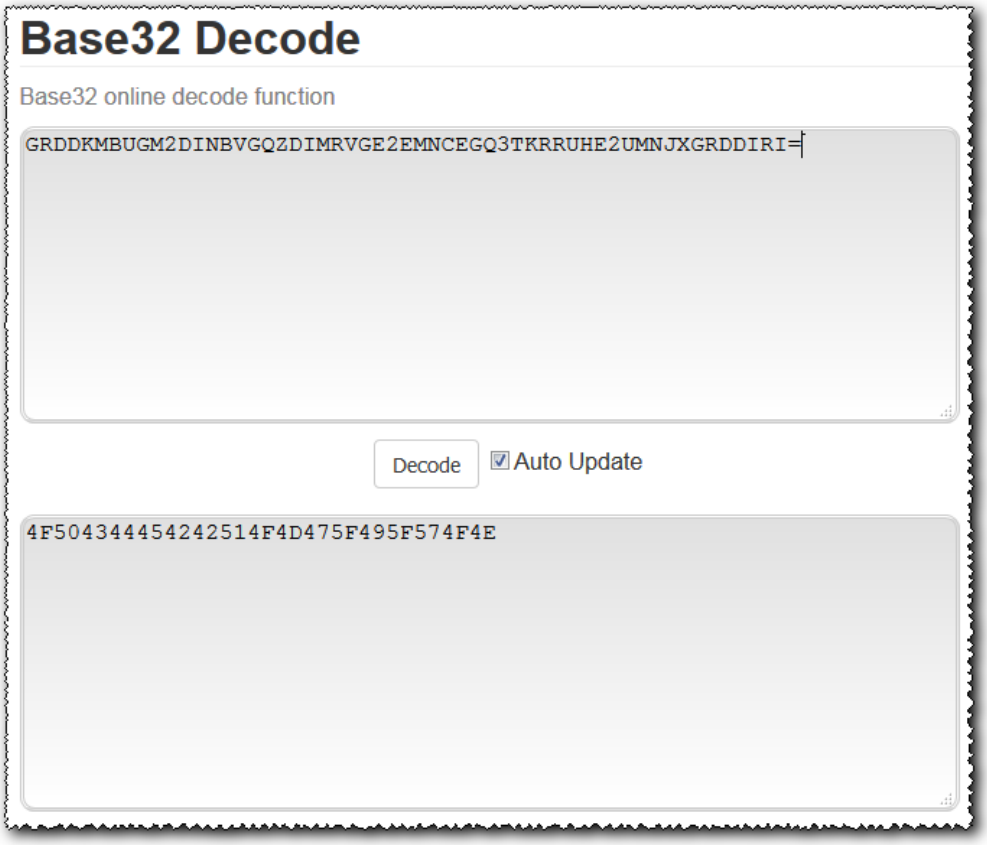


وبعد فحص الـ Smali للدالة تم اكتشاف النص التالي:



GRDDKMBUGM2DINBVGQZDIMRVGE2EMNCEGQ3TKRRUHE2UMNJXGRDDIRI=

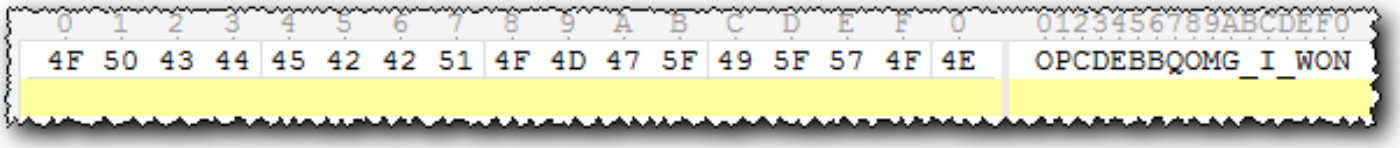
الملفت للنظر هو أن هذا النص مشابه للأساس Base64 ولكن لا توجد فيه أحرف صغيرة وهذا يدل على أنه من ترميز أو أساس أخر. ونجد أنه يحتوي على أحرف كبيرة فقط وأرقام ومجموعها ٢٦ حرف + ١٠ أرقام وبالتالي المجموع ٣٦ ونجد أن أقرب أساس له Base32 وتتم محاولة فك الترميز بواسطة الموقع:



بعد عملية فك الترميز Decoding يظهر الناتج التالي:

4F504344454242514F4D475F495F574F4E

وبعد عملية نسخ النتيجة في أداة 010 Editor تظهر النتيجة:



وهكذا تم اكتشاف الـ CTF وهو:

OPCDEBBQOMG_I_WON