# M9: Reverse Engineering

## secondo tassonomia mobile OWASP

Salvatore Quattrocchi
M01000080

Dicembre 2023

**Sommario**

In questo articolo tratteremo l'argomento del M9:Reverse Engineering secondo la tassonomia mobile OWASP. Parleremo in generale delle tecniche di Reverse Engineering Mobile, di come possiamo cercare di proteggere le nostre applicazioni da tale attacco e tratteremo degli esempi dimostrativi su come eseguire tale azione su delle Applicazioni Mobili.

# Indice

# 1 Introduzione

Prima di iniziare a parlare nello specifico iniziamo con una introduzione generale su cosa sia il Reverse Engineering.

## 1.1 Il Reverse Engineering

Il **Reverse Engineering** (**RE**) è l'atto di smantellare un oggetto per vedere come funziona. Viene utilizzato principalmente per analizzare e acquisire conoscenze sul modo in cui funziona qualcosa, ma viene spesso usato per il miglioramento o la duplicazione dell'oggetto stesso.

Nel 1990, l'IEEE, *Institute of Electrical and Electronics Engineers*, definì così il Reverse Engineering sui Software(SRE): "Il processo di analisi di un sistema per identificare i suoi componenti e le loro interazioni e creare una rappresentazione del sistema in un forma diversa da quello originale o ad un livello più elevato di astrazione".[1]

In altre parole, il Reverse Engineering è il processo di analisi di un prodotto finito, sia esso un software, una Web App[2] o una Mobile App[3], al fine di comprendere il suo funzionamento interno e la sua struttura.

### 1.1.1 Reverse Engineering Process Work

Il processo di Reverse Engineering è composto da diverse fasi, che possono variare a seconda del tipo di prodotto che viene analizzato. Tuttavia, ci sono tre passi principali che vengono seguiti, in generale, in tutti i processi di Reverse Engineering:

1. **Estrazione delle informazioni:** viene studiato il soggetto di interesse e vengono estratte informazioni sul suo design e sul suo funzionamento. Nel Software Reverse Engineering, ciò potrebbe richiedere la raccolta del codice sorgente e dei relativi documenti di progettazione per lo studio. Può anche comportare l'uso di strumenti, come un disassemblatore per suddividere il programma nelle sue parti costitutive.

2. **Modellazione:** Le informazioni raccolte vengono astratte in un modello concettuale, in cui ogni componente del modello spiega la propria funzione all'interno della struttura complessiva. Lo scopo di questo passaggio è quello di prendere informazioni specifiche dell'originale e astrattele in un modello generale che possa essere utilizzato per guidare la progettazione di nuovi oggetti o sistemi. Nell'ingegneria inversa del software, ciò potrebbe assumere la forma di un diagramma di flusso dei dati o di una struttura del software.

[1] Wikipedia, "Reverse Engineering",Estratto dal paragrafo "Software"[1]

[2] Abbrevizione di Web Application: indica un sofware accessibile/fruibile tramite Web. [2]

[3] Abbrevizione di Mobile Application, un software costruito per essere usato su Dispositivi Mobili. Estratto da MASTG pag. 28 [3]

3. **Revisione:** Ciò implica la revisione del modello e il suo test in vari scenari per garantire che sia un'astrazione realistica dell'oggetto o del sistema originale. Nell'ingegneria del software, ciò potrebbe assumere la forma di un test del software. Una volta testato, il modello può essere implementato per riprogettare l'oggetto originale.

### 1.1.2 Perché è importante il Reverse Engineering

Il Reverse Engineering è importante per diverse ragioni:

- **Analisi della sicurezza:** il Reverse Engineering può essere utilizzato per analizzare le vulnerabilità di sicurezza di un prodotto e sviluppare contromisure per proteggerlo da attacchi futuri.
- **Miglioramento del prodotto:** il Reverse Engineering può essere utilizzato per analizzare il funzionamento di un prodotto e identificare eventuali difetti o aree di miglioramento.
- **Protezione della proprietà intellettuale:** il Reverse Engineering può essere utilizzato per proteggere la proprietà intellettuale di un prodotto, identificando e prevenendo la violazione dei diritti d'autore.
- **Analisi dei malware:** il Reverse Engineering può essere utilizzato per analizzare il codice di un malware e identificare le sue funzionalità e le sue vulnerabilità.

Dopo aver fatto un'introduzione del nostro argomento, iniziamo a parlare più nello specifico del Reverse Engineering delle Mobile Application definito da OWASP.

## 1.2 Reverse Engineering mobile OWASP

OWASP tratta la sicurezza nelle applicazioni mobile come una delle sue principali preoccupazioni, questo ha portato alla redazione di due libri:

1. **Mobile Application Security Verification Standard (MASVS)**: una guida per la sicurezza delle applicazioni mobili che definisce un insieme di requisiti di sicurezza per le applicazioni mobili.[4]
2. **Mobile Application Security Testing Guide (MASTG)**: un manuale che tratta il Mobile App Security Testing e Reverse Engineering.[3]

In entrambi i libri, OWASP ha definito il Reverse Engineering come una delle principali minacce per la sicurezza delle applicazioni mobili, infatti la possiamo anche trovare nella sua *Top 10 Mobile Application Security Risks*[1].

[1] Nel 2016 viene classificato come M9, nell'attuale classifica provvisoria è stato unito insieme al precedente *M8:Code Tempering* in *M7:Insufficient Binary Protections*.[6][7]

Nel secondo libro che tratta proprio come testare la sicurezza delle applicazioni mobile, OWASP spiega che per come le Applicazioni Mobili vengono distribuite e isolate è progettato in modo più restrittivo rispetto alle Applicazioni Desktop, per questo non si possono fornire gli stessi sistemi di difesa.

Android permette ai Reverse Engineers di poterlo modificare a loro piacimento facilitando cosi il loro lavoro, cosa che non concede iOS[1] anche se questa loro "chiusura" indica anche un minor numero di opzioni difensive.

### 1.2.1 Tecniche di Reverse Engineering

Per poter effettuare un Reverse Engineering bisogna avere conoscenze riguardanti i Device su cui andranno le Applicazioni, i Sistemi Operativi (attualmente Android e iOS dominano il mercato), il linguaggio di programmazione specifico del Sistema Operativo e anche altri come Assembly[2] e Smali[3].

Infatti, ogni Sistema Operativo esce con un Software Development Kit (SDK) per lo sviluppo di Applicazioni specifico per se stesso, il quale permette lo sviluppo di App con uno specifico linguaggio di programmazione: per Android si usano Java e Kotlin[4] e per iOS si usano Swift[5] e Objective-C[6].

Quando si effettua un Reverse Engineering vi sono due tecniche di approccio: **White-Box** e **Black-Box**. Queste identificano se l'Engineer ha ha disposizione il codice (*White-Box*), quindi conoscendo tutti i processi dell'app, oppure si h solo l'applicazione e non si sa null'altro dell'app(*Black-Box*).

Esistono diverse tecniche di Reverse Engineering utilizzate per analizzare le applicazioni mobile. Esse variano in base alle esigenze del Reverse Enginee, ma le principali tecniche sono **Analisi Dinamica** e **Analisi Statica**

L'*Analisi Dinamica* consiste nell'esecuzione dell'applicazione in un ambiente controllato al fine di monitorare il suo comportamento e identificare anomalie che potrebbero portare ad eventiali punti deboli che potrebbero essere sfruttati da terze parti per copiare o duplicare l'applicazione o diffondere un malware tramite essa. Per effettuare queste analisi di possono usare programmi per il controllo della rete come Wireshark[7][21].

L'*Analisi Statica* consiste nell'analisi del codice sorgente dell'applicazione senza eseguirla. Questa tecnica viene eseguita decompilando l'applicazione mobile e analizzando il codice sorgente ottenuto alla ricerca di informazioni o altro. Questo processo viene fatto generalmente sul singolo file manualmente, ma può anche essere automatizzato. Infatti si possono utilizzare Software che scansiona-

[1] infatti riuscire a trovare un Emulatore iOS è quasi impossibile se non si ha un dispositivo Apple e quelli che si possono trovare senza sono a pagamento

[2] Linguaggio di programmazione a basso livello molto vicino al linguaggio macchina[8]

[3] Linguaggio di programmazione intermedio che permette di interpretare il Bytecode, con cui sono trascritte le applicazioni Android, in un formato più leggibile[9]

[4] per saperne di più su Kotlin `https://kotlinlang.org/`

[5] Per altre info su Swift vedi `https://developer.apple.com/swift/`

[6] per altre info su Objective-C vedi `https://en.wikipedia.org/wiki/Objective-C`

[7] Software OpenSource di packet sniffer, usato tipicamente per osservare tutto il traffico sulla rete

no il codice alla ricerca di parole chiave come "password", "username", "cipher" e che memorizzi questi elementi e li presenti alla fine.

Queste tipo di analisi procurano molto spesso farsi positivi per cui e sempre consigliato un controllo da parte dell'utente.

### 1.2.2 Basic Tampering Techniques

Le classiche tecniche di manomissione di un'app si classificano in due tipi: il **Binary Patching** e il **Code Injection**.

Il *Binary Patching* è il processo del cambiare l'app compilata, il modificare il suo bytecode o manomettere le sue risorse. Questo processo è conosciuto come *modding* nel mondo dei videogiochi. Il Patching può essere fatto in molti modi, incluso editare un file binario in un hex editor e decompilare, modificare e reassemblare l'applicazione.

Il *Code Injection* è una tecnica molto potente di manomissione che consente di inittare al suo inyerno dei pezzi di codice. Ciò può essere fatto per modificare il comportamento dell'applicazione o per accedere a informazioni riservate. Un software che permette questa azione è *Frida*[20] un software completamente gratuito multi piattaforma molto ricco e facile da installare. Infatti serve avere solo Python con versione maggiore della terza e usando il comando

```
pip install frida-tools
```

da PowerShell di Windows o sul Terminale di Linux/MacOS e il gioco è fatto. Frida è stato progettato con lo scopo di essere un "framework di strumentazione dinamica".

Queste sono le tecniche basi che vengono usate per la manomissione, le altre tecniche da usare cambiano da caso a caso e si acquisiscono tramite l'approfondita conoscenza e l'esperienza su campo.

## 1.3 Difesa dal Reverse Engineering

Nel caso del Reverse Engineering, l'uso della parola Difesa è un po improprio. Infatti non è possibile una difesa completa ma si può solo cercare di prevenire questo attacco.

La tecnica di prevenzione che viene sempre usata nelle Applicazioni Mobili è l'**Obfuscation**. Questa tecnica consiste nel modificare il codice sorgente per cercare di rendere il testo più difficile da comprendere e tal volta persino da decompilare. Obfuscation non può essere semplicemente spento , programmi possono essere resi completamente incromprensibili, se non totalmente, in parti.

Le tecniche usate per l'Obfuscation di un'applicazione comprendono: *Name Obfuscation*, *sostituzione delle istruzioni*, *inserimento di Dead Code*, *String Encryption*, *Packing* e *Control Flow Flattering*.

- **Name Obfuscation** consiste nella sostituzione dei nomi con altri in modo che effettuando una ricerca non vengano riconosciuti facilmente e quindi presi di mira.

- **Sostituzione delle Istruzioni** la sostituzione degli operatori binari standard con una più complessa rappresentazione che e che per ogni operazione ci siano più versioni sostituitive in modo da rendere semplice il riconoscimento ES: x= a+b diventerebbe x = - (-a) - (- b).
- **Inserimento di Dead Code** è una tecnica per aumentare la complessità del codice inserendo del codice che non crei effetti sul funzionamento dell'applicazione.
- **String Encryption** è una tecnica che di crttografia delle stringhe usate nell'applicazione e dei metodi che permettono la loro lettura corretta quando devono essere utilizzate.
- **Packing** è una tecnica di tipo dinamico che comprime il codice o lo codifica e che implementa un recupero dinamico durante l'esecuzione.
- **Control Flow Flattering**: è una tecnica che sostituisce il codice originale con una ancora più complessa rappresentazione. Una funzione può essere divisa nei suoi singoli componenti e con l'aggiunta di righe di codice superfluo viene inserito in un loop.

original

```
i = 1;
s = 0;


while (i <= 100) {



  s += i;
  i++;

}
```

control-flow flattening applied

```
int swVar = 1;
while (swVar != 0) {
  switch (swVar) {
    case 1: {
      i = 1;
      s = 0;
      swVar = 2;
      break;
    }
    case 2: {
      if (i <= 100)
        swVar = 3;
      else
        swVar = 0;
      break;
    }
    case 3: {
      s += i;
      i++;
      swVar = 2;
      break;
    }
  }
}
```

Come il control-Flow altera il codice

# 2 Reverse Engineering di InsecureShop

InsecureShop[18] è una applicazione Android scritta in Kotlin progettata per essere intenzionalmente vulnerabile. È un ottimo strumento per gli appassionati di sicurezza di Android poiché incorpora molte vulnerabilità. Per poter effettuare questo esempio avremo bisogno dei seguenti strumenti:

1. **InsecureShop**: il file APK[1] contenente l'applicazione mobile
2. **Android Studio**: un IDE per lo sviluppo di Applicazioni Mobile per Android[12] comprensivo di un emulatore e in suo pacchetto di Strumenti SDK.
3. **Platform Tools**: Pacchetto di Tools CLI[2] di Android; riguardo questo pacchetto, io ho riscontrato problemi poiché all'installazione il loro percorso non viene inserito nel PATH[3] di Windows, potete inserilo voi li o potete fare come me, scaricare[13] questo pacchetto, inserirlo nella directory dove andremo ad operare sulle Mobile App ed eseguirlo da una directory specifica aprendo il terminale lì[4].
4. **Visual Studio Code**[14] *VSCode*, un editor della Microsoft che in realta è molto di più.
5. **APKLab**:Estensione di VSCode scaricabile sia dal link citato qui[15] o direttamente da VSCode cercandola nell'apposito campo di ricerca nella sezione Estenzioni.Questa estenzione contiene in un unico pacchetto strumenti quali apktool[5] e jadx[6] per abilitare funzionalità tra cui decompressione, decompilazione, patching del codice e riconfezionamento delle app direttamente dall'editor.

[1] Formato delle applicazioni di Android

[2] Command Line Input: comandi da usare in un terminale

[3] PATH è una variabile di sistema utilizzata per individuare più facilmente gli eseguibili

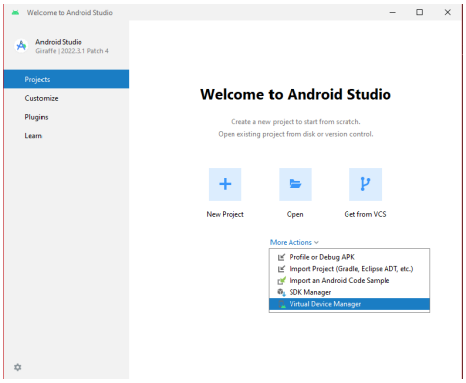
[4] Premendo Maiuscola nella tastiera ed il tasto destro del mouse sulla cartella, vi apparirà nel menù la scelta di aprire il prompt o PowerShell con il percorso della cartella.

[5] Usata per estrarre gli elementi all'interno di APK in un formato non leggibile, vedi[16]

[6] Dex to Java Decompiler: un tool che permette di convertire i file Android Dex in codice sorgente Java[17]

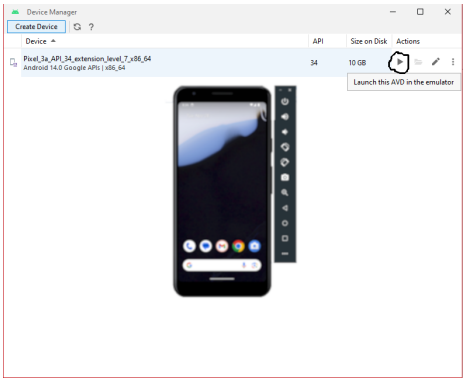
## 2.1 Avvio Emulatore e recupero prime informazioni App

Per prima cosa avviamo l'Emulatore di Android Studio cosi da poter installare l'applicazione e iniziare a reperire informazioni iniziali. Dalla Schemata di Welcome clicchiamo "*More Actions*" e da li "*Virtual Device Manager*"




Avvio Virtual Device Manager

All'apertura della nuova finestra, dovrebbe già essere presente un emulatore[1], premete il simbolo play per iniziarne l'avvio: nel nostro caso emula un Pixel 3A con sistema Android 14.




AVD Avviato

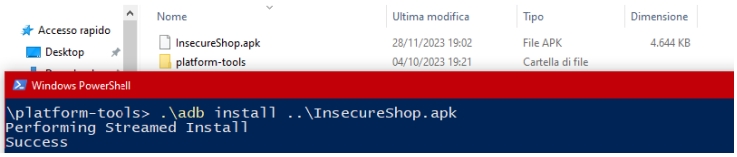
Adesso andiamo dove abbiamo salvato il nostro apk, preferibilmente una cartella dedicata, e utilizzando il tool adb.exe[2] presente in Platform Tools[13] di Android Studio installiamo la nostra App sul nostro emulatore scrivendo il seguente comando su PowerShell e premendo invio:

[1] Se non dovesse essere presente, premere il pulsante "*Create Device* per avviare la procedura guidata di creazione. Essa è molto intuitiva e facile sa eseguire"

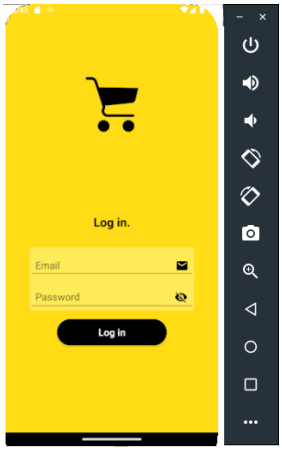
[2] Android Debug Bridge: strumento a riga di comando che ci permette di comunicare con nostro dispositivo Adroid

```
.\adb install ..\InsecureShop.apk
```

Questo comando può essere usato tale e quale se il file InsecureShop.apk si trova nella cartella contenente anche la cartella Platform Tools che contiene adb.exe.




Installazione tramite adb.exe

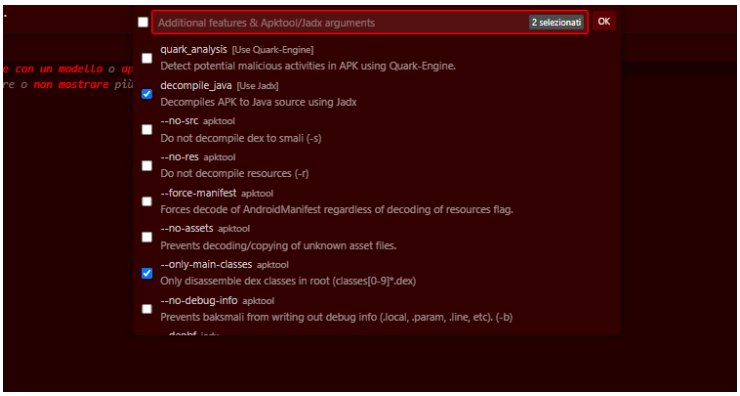



Applicazione InsecureShop all'avvio

All'avvio dell'applicazione osserviamo essere presente una classica schermata di login con due spazi per Email e Password e un tasto con la dicitura "*Log in*". Non essendoci altre informazioni ricavabili osservando, passiamo all'analisi del codice dell'applicazione.

## 2.2 Estrazione Codice Sorgente

Apriamo VSCode e eseguiamo la combinazione *Ctrl+Shift+P* per aprire il riquadro dei comandi e digitare *APKLab: Open an APK*. Dalla schermata apertasi andiamo a trovare la cartella dove è stato salvato InsecureShop.apk e lo selezioniamo e premiamo *OK*. Adesso APKLab aprirà una schermata su VSCode dove ci permetterà di selezionare delle opzioni aggiuntive: selezioniamo *"decompile_java [use Jadx]"*, cosi indichiamo a APKLab di usare Jadx per decompilare i file dell'app, e diamo l'Ok.




Menù delle opzioni di APKLab

Dipende da quanto codice deve analizzare e convertire potrebbe richiedere qualche tempo per finire. Come possiamo vedere dalla figura 7 Apklab utilizzerà due Tool:

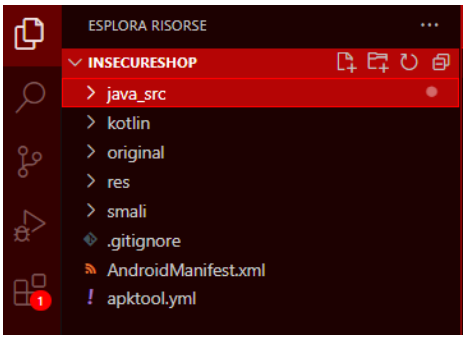
- Apktool: per decompilare l'applicazione e convertire i file in codice sorgente .dex
- Jadx: per decompilare i file .dex e convertirli in codice sorgente Java

Gli ultimi comandi, nella sezione *Initializing*, sono dei comandi relativi a Git[1] che non ha relazioni con il lavoro che dovremmo svolgere, possiamo anche eliminare la cartella .git generata.

```
------------------------------------------------------------------------------------------------------------------------
Decoding InsecureShop.apk into c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop
------------------------------------------------------------------------------------------------------------------------
java -jar C:\Users\salvo\.apklab\apktool_2.8.1.jar d c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop.apk -o
c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop --only-main-classes
I: Using Apktool 2.8.1 on InsecureShop.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\salvo\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
Decoding process was successful
------------------------------------------------------------------------------------------------------------------------
Decompiling InsecureShop.apk into c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop\java_src
------------------------------------------------------------------------------------------------------------------------
C:\Users\salvo\.apklab\jadx-1.4.7\bin\jadx.bat -r -q -ds c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop\java_src
c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop.apk
Decompiling process was successful
------------------------------------------------------------------------------------------------------------------------
Initializing c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop as Git repository
------------------------------------------------------------------------------------------------------------------------
cd /d "c:\Users\salvo\Documents\Università\IS\progetto file\InsecureShop\InsecureShop" && git init && git config core.safecrlf false && git add -A && git commit -q
"Initial APKLab project"
Initialized empty Git repository in C:/Users/salvo/Documents/Università/IS/progetto file/InsecureShop/InsecureShop/.git/
Initializing Git process was successful
```

Output APKLab

[1]Servizio per il salvataggio di progetti in repository online




Cartelle create da APKLab

## 2.3 Prima Analisi del Codice Estratto

Andiamo su VSCode e per prima cosa andiamo a leggerci il file *AndroidManifest.xml*, un file che fornisce importanti informazioni sulle caratteristiche dell'applicazione. Per iniziare la nostra Analisi proviamo ad usare le informazioni che abbiamo: quando abbiamo avviato l'app ci siamo trovati di fronte ad una schermata di Log in,vediamo se c'è qualche cosa di correlato.

Su *AndroidManifest.xml* troviamo una dicitura che cattura la nostra attenzione

```
<activity android:name="com.insecureshop.LoginActivity"/>
```

che ci indica che l'attività di Log in si trova in un file chiamato *LoginActivity.java*. Andiamo a cercare questo file, apriamo la cartella *java_src* e seguiamo il percorso *com.insecureshop.LoginActivity*

Controllando il codice troviamo una funzione *verifyUserNamePassword* usata per validare le credenziali d'accesso:

```
1  boolean auth = Util.INSTANCE.verifyUserNamePassword(username
       , password);
2  if (auth) {
3      Prefs prefs = Prefs.INSTANCE;
4      Context applicationContext = getApplicationContext();
5      Intrinsics.checkExpressionValueIsNotNull(
           applicationContext, "applicationContext");
6      prefs.getInstance(applicationContext).setUsername(
           username);
7
8      Prefs prefs2 = Prefs.INSTANCE;
9      Context applicationContext2 = getApplicationContext();
10     Intrinsics.checkExpressionValueIsNotNull(
           applicationContext2, "applicationContext");
11     prefs2.getInstance(applicationContext2).setPassword(
           password);
12
```

```
13      Util.saveProductList$default(Util.INSTANCE, this, null,
            2, null);
14
15      Intent intent = new Intent(this, ProductListActivity.
            class);
16      startActivity(intent);
17
18      return;
19  }
```

Questa funzione è chiamata da una Classe di nome Util importata all'inizio del file:

```
1  import com.insecureshop.util.Util;
```

Andiamo a darle uno sguardo. Troviamo subito il metodo *verifyUserName-Password*:

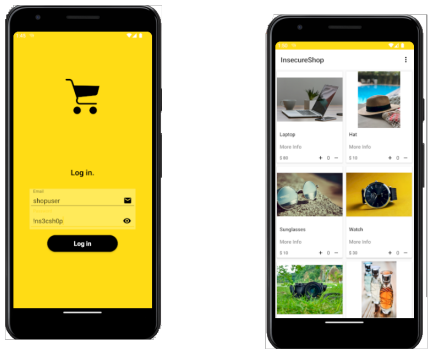
```
1       public final boolean verifyUserNamePassword(String
            username, String password) {
2           Intrinsics.checkParameterIsNotNull(username, "
                username");
3           Intrinsics.checkParameterIsNotNull(password, "
                password");
4
5           if (getUserCreds().containsKey(username)) {
6               String passwordValue = getUserCreds().get(
                    username);
7               return StringsKt.equals$default(passwordValue,
                    password, false, 2, null);
8           }
9           return false;
10      }
```

In questo vediamo che viene usato un altro metodo *getUserCreds* per controllare Username e Password:

```
1       private final HashMap<String, String> getUserCreds() {
2           HashMap userCreds = new HashMap();
3           userCreds.put("shopuser", "!ns3csh0p");
4           return userCreds;
5       }
```

L'Username e la Password sono codificate in chiaro in questa funzione, inseriamoli nell'applicazione. Siamo riusciti a entrare!!!

Questo è un chiaro caso di *Hardcoded Credentials*, di credenziali codificate direttamente nel codice. Questo tipologia di falla nella sicurezza affligge molti sistemi e continuerà a farlo perchè quando si pensa alla sicurezza nessuno pensa per prima cosa ad una password lasciata in chiaro, ma a cose ben più complicate finché non incappano in quel problema loro stessi.




Inserimento Credenziali trovate Nell'Applicazione

**Contromisure**

Per poter evitare falle del genere si consiglia l'utilizzo del metodo di Obfuscation String Encryption, in modo da non rendere facile l'individuazione di credenziali ma anche di altre stringhe all'interno del codice. Si potrebbe utilizzare una semplice funzione di crittografia come il Salt[1] che prevede il salvataggio di una stringa ricavata dall'uso del metodo Salt e del Salt stesso per poter effettuare il confronto, questo renderebbe utilizzabili attacchi di tipo BruteForce[2]. In alternativa, si potrebbero usare una **Password tipo OTP**, *One-Time Password*[3], che, per la sua stessa natura, dopo l'accesso perde la loro funzione, così anche se salvate in chiaro sarebbero quasi inutilizzabili.

## 2.4 Seconda Analisi del Codice

Ritoniamo al nostro *AndroidManifest.xml* e continuamo ad analizzare ciò che vi è scritto. Sotto la riga di codice riferita al Login troviamo una segmento di codice che tratta la "Vista Web":

```
<activity android:name="com.insecureshop.WebViewActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data
            android:host="com.insecureshop"
            android:scheme="insecureshop"/>
        </intent-filter>
</activity>
```

[1] vedi Wikipedia `https://it.wikipedia.org/wiki/Salt_(crittografia)`

[2] Attacchi di forza bruta in cui si tenta di trovare la password in modo forzato provando le varie combinazioni

[3] Password utilizzabile solo una volta

Ci segniamo il percorso all'interno della cartella java_src della classe e andiamo a dare uno sguardo. La classe è collocata nel seguente percorso

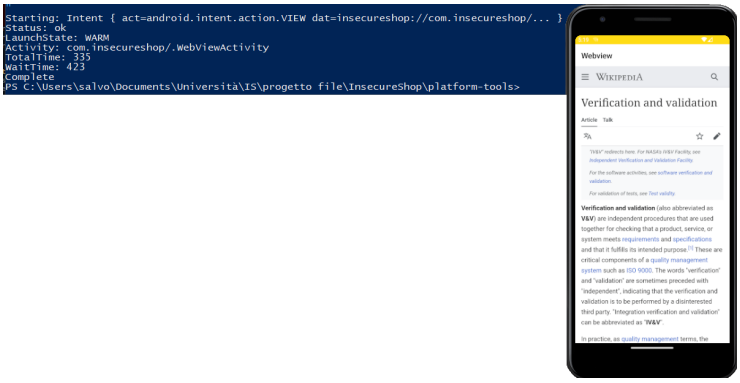
```
com.insecureshop.WebViewActivity.java
```

Analizzando il codice ci imbattiamo in una falla nella sicurezza all'interno del metodo onCreate; in esso si controlla se all'interno del *url* sono presenti le stinghe "*/web*" o */webview* ma non controlla nessun altra cosa. Questo permetterebbe a chiunque voglia può costruirsi un suo URI e passare un parametro arbitrario come url; vediamo come possiamo farlo utilizzando adb Activity Manager e un URI:

```
.\adb shell am start -W -a
   android.intent.action.VIEW -d
   "insecureshop://com.insecureshop/web?url=
        https://en.wikipedia.org/wiki/Verification_and_validation"

   # am: activity Manager
   # start:Avvia una attività specificata da un Intent
   # -W: aspetta la fine dell'avvio
   # -a: l'attività svolta dall'intent -d: intent URI
```

Quindi con questo comando avviamo una shell(terminale) nel nostro dispositivo android da cui avviamo un Activity Manager a cui diciamo di avviare uno specifico intent dato da intent URI che svolge una specifica attività, nel nostro caso "android.intent.action.VIEW" cioé di visualizzazzione.




Risultato del dell'invio di un URI intent alla nostra App

### Possibili Contromisure

Per evitare questo problema Google chiede a tutti di associare all'applicazione un sito web su cui pubblicare un file JSON Digital Asset Links che indica le

app associate al sito web e verificare gli intent dell'URL dell'app. Si possono associare più app ad un sito web e anche più siti ad una App.

Se quindi si attiva la funzione *android:autoVerify=true* e il dispositivo presenta un Android dal 6.0 in su, allora il sistema verificherà automaticamente gli host associati all'App.[10]

# 3 Reverse Engineering di UnCrackable1

UnCrackable1[19] è una Applicazione messa a disposizione dalla OWASP per permettere a persone che hanno appena cominciato con il Reverse Engineering a muovere i primi passi. Vengono usati anche nel loro libro MASTG[3]. Per poter efettuare questo esempio avremo bisogno:

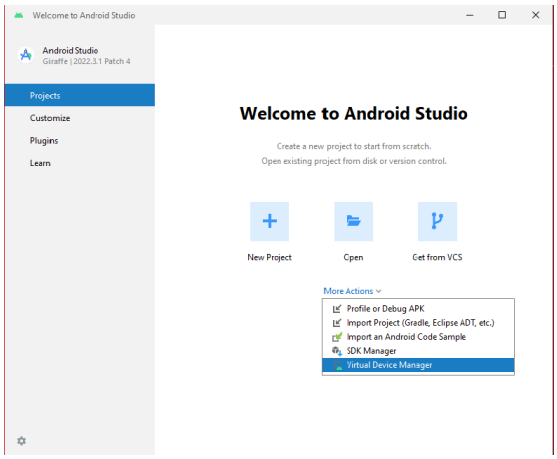
- **UnCrackable1.apk**: il file APK della OWASP contenente l'applicazione mobile;
- **Android Studio**[12]: un IDE per lo sviluppo di Applicazioni Mobile per Android comprensivo di un emulatore e in suo pacchetto di Strumenti SDK;
- **Platform Tools**[13]: Pacchetto di Tools CLI di Android; riguardo questo pacchetto, io ho riscontrato problemi poiché all'installazione il loro percorso non viene inserito nel PATH di Windows, potete inserilo voi li o potete fare come me, scaricare questo pacchetto, inserirlo nella directory dove andremo ad operare sulle Mobile App ed eseguirlo da una directory specifica, aprendo il terminale lì.
- **Visual Studio Code**[14]: *VS Code*, un editor della Microsoft che in realta è molto di più;
- **Apklab**[15]: Estensione di VSCode scaricabile sia dal link citato qui[15] o direttamente da VSCode cercandola nell'apposito campo di ricerca nella sezione Estenzioni.Questa estenzione contiene in un unico pacchetto strumenti quali apktool[1] e jadx[2] per abilitare funzionalità tra cui decompressione, decompilazione, patching del codice e riconfezionamento delle app direttamente dall'editor.

## 3.1 Avvio Emulatore e recupero prime informazioni App

Per prima cosa avviamo Android Studio e come abbiamo fatto prima, dalla schermata di Welcome clicchiamo "*More Actions*" e da li "*Virtual Device Manager*"

[1] Usata per estrarre gli elementi all'interno di APK in un formato non leggibile

[2] Dex to Java Decompiler: un tool che permette di convertire i file Android Dex in codice sorgente Java[17]



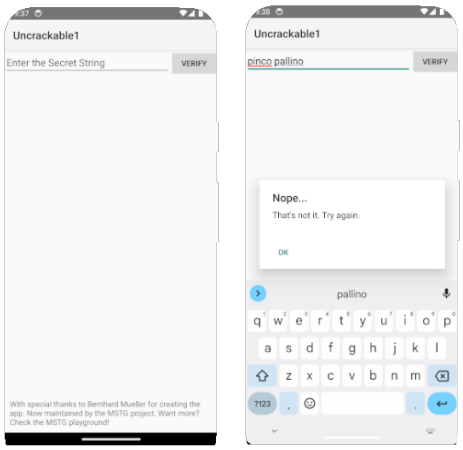

Avvio Virtual Device Manager

All'apertura della schermata,cliccheremo il pulsante play del nostro emulatore e aspettando il suo avvio, andemo nella cartella dove abbiamo salvato UnCrackable1.apk insieme al SDK Platform Tools.

Al suo avvio apriremo una PowerShell nella cartella di Platform e tramite adb.exe installeremo la nostra apk sul nostro dispositivo. Ricordo cancora che il comando da usare è

```
.\adb install ..\UnCrackable1.apk
```

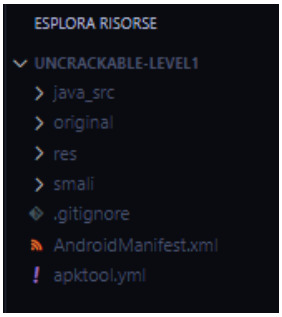
Finito il processo di installazione, andiamo ad aprire l'app. L'applicazione si presenta di grafica semplice: presenta solo una casella testuale e un tasto dalla dicitura *VERIFY*. Provando ad inserire qualche cosa e cliccando Verify appare una messaggio che ci avvisa dell'inserimento della frase sbagliata. Andiamo ad analizzare il codice.




Avvio e Analisi funzionamento App

## 3.2 Estrazione Codice Sorgente

Apriamo VS Code e eseguendo la combinazione *Ctrl+Shift+P*, apriamo il riquadro di comandi e digitiamo *Apklab: Open an APK*[1]. Dalla schermata apertasi, andiamo a trovare la nostro app da analizzare e diamo OK, ricordiamoci di spuntare nel menù delle opzioni aggiuntive di APKLab il comando *decompile_java [use Jadx]*. In questo momento APKLab sta utilizzando due tools, Apktool[16]




Directory Create da APKLab

e Jadx[17] per decomprimere e ricompilare i sorgenti della nostra app in file leggibili per noi.

## 3.3 Analisi del Codice Estratto

Andiamo su VS Code e andiamo a dare un'occhiata al file *AndroidManifest.xml*, un file che fornisce importati informazioni sulle caratteristiche dell'applicazione.

[1] Non è necessario scriverlo tutto, basta selezionarlo appena appare in lista

L'unico riferimento presente all'interno dell'AndroidManifest.xml è

```
<activity android:label="@string/app_name"
      android:name="sg.vantagepoint.uncrackable1.MainActivity">
```

essa ci indica la classe principale *MainActivity* che si trova al percorso *sg.vantagepoint.uncrackable1.MainActivity* apriamola. Ad una lettura della classe, individuiamo tre metodi:

1. *a*.
2. *onCreate*;
3. *verify*;

Il metodo *OnCreate* è il primo metodo ad essere invocato all'avvio dell'applicazione che, salta subito all'occhio, contiene un metodo di riconoscimento per i *Device Rooted*[1] o che possono essere eseguiti in modalità *Debuggable*[2]. Entrambi questi metodi ci mandano a controllare delle classi nel percorso *sg.vantagepoint.a*.

Questo metodo usa anche uno dei metodi presenti qui, *a*, per riferire il messaggio del rilevamento del *Root* o del *Debug* attivo e aziona l'arresto immediato dell'applicazione alla conferma della visione del messaggio. Il metodo *verify* sembra sia correlato alla frase misteriosa da trovare, ci torneremo di sicuro dopo.

Anche se non ne avremmo bisogno pensiamo a come disattivare i metodi di controllo in *onCreate*.

### 3.3.1 Disabilitare Sistema Anti-Rooted

Per prima cosa andiamo a controllare i metodi che permettono il riconoscimento dei sistemi con Root:

```
1   public static boolean a() {
2       for (String str : System.getenv("PATH").split(":"))
            {
3           if (new File(str, "su").exists()) {
4               return true;
5           }
6       }
7       return false;
8   }
9
10  public static boolean b() {
11      String str = Build.TAGS;
12      return str != null && str.contains("test-keys");
```

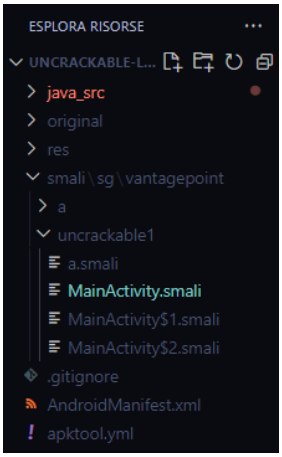
[1] il Rooting è un medoto che permette agli utenti di dispositivi android di ottenere controlli privilegiati sui vari sottosistemi Android, vedi `https://it.wikipedia.org/wiki/Rooting`

[2] Questo per evitare il debbugging che permette di vedere e modificare lo stato del software durante l'esecuzione

```
13      }
14
15      public static boolean c() {
16          for (String str : new String[]{"/system/app/
                Superuser.apk", "/system/xbin/daemonsu", "/system
                /etc/init.d/99SuperSUDaemon", "/system/bin/.ext/.
                su", "/system/etc/.has_su_daemon", "/system/etc/.
                installed_su_daemon", "/dev/com.koushikdutta.
                superuser.daemon/"}) {
17              if (new File(str).exists()) {
18                  return true;
19              }
20          }
21          return false;
22      }
```

Questi metodi controllano la presenza di determinate stringhe all'interno del OS Android per controllarne l'avvenuta modifica.

L'unico metodo per risolvere la situazione che sembra possibile effettuare è la modifica dei valori riportati dai vari metodi in modo che ritornino *false* e, quindi, il recompilamento dell'applicazione. Per fare ciò andiamo dal menù laterale sinistro ad aprire il corrispondente della classe *MainActivity.java* all'interno delle classi generate in *smali*. Guardando i metodi scritti in smali, si decide

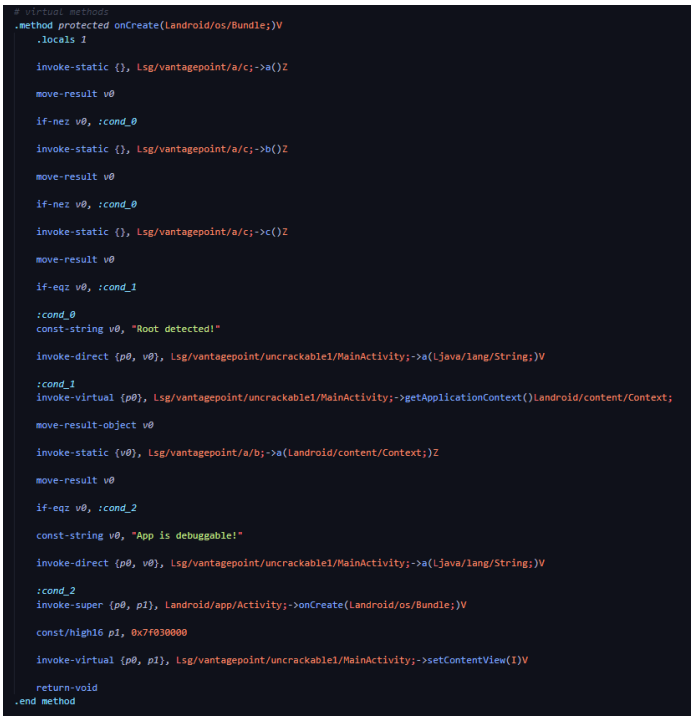



MainActivity all'interno della Directory Smali

per comodità di modificare non modificare tutti i metodi ma solo il metodo *onCreate* cosi da non dover modificare troppo il codice.

Prima di parlare delle modifiche da apportare diamo alcune semplici informazioni per poter interpretare questo codice smali: Il seguente codice

```
invoke-static {}, Lsg/vantagepoint/a/c;->a()Z
move-result v0
```



```
# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 1

    invoke-static {}, Lsg/vantagepoint/a/c;->a()Z

    move-result v0

    if-nez v0, :cond_0

    invoke-static {}, Lsg/vantagepoint/a/c;->b()Z

    move-result v0

    if-nez v0, :cond_0

    invoke-static {}, Lsg/vantagepoint/a/c;->c()Z

    move-result v0

    if-eqz v0, :cond_1

    :cond_0
    const-string v0, "Root detected!"

    invoke-direct {p0, v0}, Lsg/vantagepoint/uncrackable1/MainActivity;->a(Ljava/lang/String;)V

    :cond_1
    invoke-virtual {p0}, Lsg/vantagepoint/uncrackable1/MainActivity;->getApplicationContext()Landroid/content/Context;

    move-result-object v0

    invoke-static {v0}, Lsg/vantagepoint/a/b;->a(Landroid/content/Context;)Z

    move-result v0

    if-eqz v0, :cond_2

    const-string v0, "App is debuggable!"

    invoke-direct {p0, v0}, Lsg/vantagepoint/uncrackable1/MainActivity;->a(Ljava/lang/String;)V

    :cond_2
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    const/high16 p1, 0x7f030000

    invoke-virtual {p0, p1}, Lsg/vantagepoint/uncrackable1/MainActivity;->setContentView(I)V

    return-void
.end method
```

Metodo onCreate nella Directory Smali

viene usato per invocare un metodo (*invoke-static*), che non ha bisogno di parametri ($\{\}$)[1] facente parte della classe *sq/vantagepoint/a/c Lsg/vantagepoint/a/c*. Il nome del metodo invocato è a ($\rightarrow a()$) che ritorna un booleano ($Z$). Con *move-result* si passa il valore ottenuto con il metodo alla variabile, in questo caso v0 che sarà 0 se viene ritornato false mentre 1 se è true.

I confronti fra due variabili vengono effettuati con la seguente codifica:

```
if-eq vA, vB, label   #valuta se vA e vB sono uguali.
if-ne vA, vB, label   #valuta se vA e vB non sono uguali.
if-eqz vA, label   #Valuta se vA è uguale a zero (FALSE).
if-nez vA, label   #Valuta se vA non è uguale a zero (TRUE).
```

Il termine label qui usato fa riferimento ad una porzione del codice che inizia proprio col termite indicato in label, il quale è il codice da eseguire se la condizione è verificata.

Per la dichiarazione di variabili, esse vengono indicate con la lettera v seguita da un numero e si dichiarano precedendole con il termine *const*

```
const v0, 10 #Assegnamo il valore 10 a v0
const-string v0, "test" #equivale a String v0 = "test"
```

[1] quando si passa un parametro si scrive {p0}

Ad const di aggiungono elementi per specificare il tipo di da dichiarare, il registro e altro ancora.

Dopo aver dato questa introduzione al codice smali andiamo a modificare il testo. Come gia accennato prima io non andrei a modificare i metodi a, b, c, ma andrei a modificare l'azione effettuata da onCreate alla ricezione della conferma del Root o del Debug disponibile.

Quindi andremo a modificare i valori riportati dalle espressioni *if-nez v0 :cond_0* con *if-nez v0 :cond_1*. Qui trascrivo la spiegazione del codice:

```
if-nez v0 :cond_0
# Se v0 diverso da 0 esegui :cond_0

...

:cond_0
const-string v0, "Root detected!"
# String v0="Root detected!"

invoke-direct {p0, v0},
    Lsg/vantagepoint/uncrackable1/MainActivity;
                              ->a(Ljava/lang/String;)V
# richiamo il metodo a(),
# percorso sg/vantagepoint/uncrackable1/MainActivity,
# passandogli il parametro, non ritorna nulla (Void)
# Ricordo che il metodo a() in MainActivity è
#                                 in metodo che chiude
# l'app dopo che rileva delle anomalie

:cond_1
invoke-virtual {p0},
    Lsg/vantagepoint/uncrackable1/MainActivity;
        ->getApplicationContext()Landroid/content/Context;
# Questo metodo va a richiamare una parametro
#                       che viene dato fornito da Android

move-result-object v0
# passa il risultato del metodo precedente a v0
```

Quindi diremo al sistema che se è Rooted, va a ricavare la variabile Context, cosi da evitare il richiamo al metodo a(), ma ancora non è finita. Dobbiamo aggiungero una riga in modo da evitare la lettura della seguente parte di codice

```
const-string v0, "App is debuggable!"

invoke-direct {p0, v0},
```

```
    Lsg/vantagepoint/uncrackable1/MainActivity;
                                ->a(Ljava/lang/String;)V
```

Questo pezzo viene eseguito dopo *if-eqz v0, :cond_2* che si trova immediatamente dopo a *:cond_1* visto prima.

Noi dobbiamo fare in modo che qualunque valore abbia v0 in questo *if-eqz* il nostro codice vada a leggere *cond_2* dove si trova il metodo di avvio dell'app. quindi aggiungimo sotto a *if-eqz v0, :cond_2* che anche se il valore non è zero si deve andare a leggere *cond_2*, per cui:

```
if-eqz v0, :cond_2
if-nez v0, :cond_2

...

# ecco a voi anche il codice richiamato
:cond_2
invoke-super {p0, p1}, Landroid/app/Activity;
                        ->onCreate(Landroid/os/Bundle;)V
# invoca la procedura di avvio

const/high16 p1, 0x7f030000
# inizializza un paramentro,
# questo viene usato per i float in genere

invoke-virtual {p0, p1},
    Lsg/vantagepoint/uncrackable1/MainActivity;
                                ->setContentView(I)V
# Questo ci carica gli elementi visivi

return-void
# riga finale di chiusura del metodo onCreate
# che ritorna Void cioé nulla
```

Quindi per ricapitolare

- cambiamo *if-nez v0 :cond_0*⇒ *if-nez v0 :cond_1*
- aggiungiamo*if-nez v0, :cond_2* sotto a *if-eqz v0, :cond_2*

Adesso dovremmo dare due cose: ricompilare UnCrackable1.apk con nostro codice modificato e avviarlo su un dispositivo Rooted per verificare che le correzioni funzionano. Ma visto che abbiamo gia tutto pronto continuiamo con la nostra analisi per trovare la frase segreta.

### 3.3.2 Ricerca Frase Segreta

Ritorniamo a esaminare il file MainActivity.java e controlliamo il metodo *verify()*:

```
    public void verify(View view) {
        String str;
        String obj = ((EditText) findViewById(R.id.edit_text
            )).getText().toString();
        AlertDialog create = new AlertDialog.Builder(this).
            create();
        if (a.a(obj)) {
            create.setTitle("Success!");
            str = "This is the correct secret.";
        } else {
            create.setTitle("Nope...");
            str = "That's not it. Try again.";
        }
        create.setMessage(str);
        create.setButton(-3, "OK", new DialogInterface.
            OnClickListener() {
            @Override
            public void onClick(DialogInterface
                dialogInterface, int i) {
                dialogInterface.dismiss();
            }
        });
        create.show();
    }
```

Come possiamo vedere, esso crea una Stringa di nome *obj* e le passa il testo scritto nella casella di testo dell'app. *obj* viene passato come parametro al metodo *a()* che si trova in *sg/vantagepoint/uncrackable1/a*, quindi nella stessa cartella di MainActivity.

Dato che questo metodo a() ritorna un boolean, sembra effettuare una qualche sorta di validazione, sembra che controllare questo metodo sia la strada giusta.

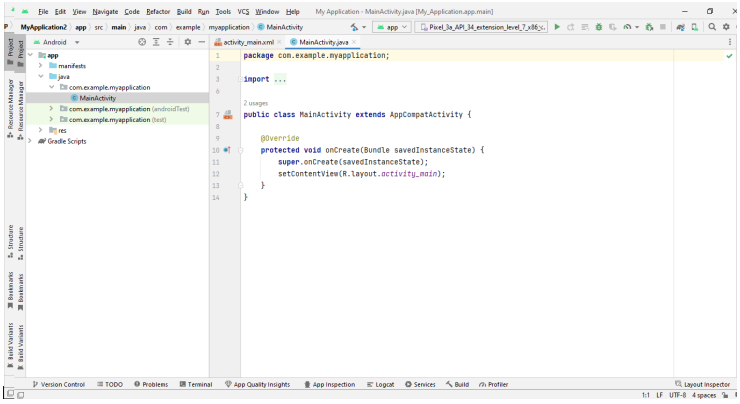
```
public class a {
    public static boolean a(String str) {
        byte[] bArr;
        byte[] bArr2 = new byte[0];
        try {
            bArr = sg.vantagepoint.a.a.a(b("8
                d127684cbc37c17616d806cf50473cc"), Base64.
                decode("5
                UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=",
                 0));
        } catch (Exception e) {
            Log.d("CodeCheck", "AES error:" + e.getMessage()
                );
            bArr = bArr2;
        }
        return str.equals(new String(bArr));
```

```
12      }
13
14      public static byte[] b(String str) {
15          int length = str.length();
16          byte[] bArr = new byte[length / 2];
17          for (int i = 0; i < length; i += 2) {
18              bArr[i / 2] = (byte) ((Character.digit(str.
                    charAt(i), 16) << 4) + Character.digit(str.
                    charAt(i + 1), 16));
19          }
20          return bArr;
21      }
22  }
```

Il metodo a() prende come input la frase che noi scriviamo nell'app e la confrota con la variabile bArr.
Se sono uguali il metodo ritorna TRUE, se non lo sono ritorna FALSE, per cui, in qualunque caso, la variabile *bArr* contiene la frase in chiaro che stiamo cercando. Ora come facciamo a ottenerla?

Tutto ciò che serve alla generazione della Frase Segreta si trova qui, quindi se prendiamo tutto l'occorrente, possiamo dunque crearci una app che semplicemente calcoli la frase.

Andiamo su *Android Studio* e cliccliamo *New Project*. Nella schermata che appare selezioniamo rimaniamo sulla selezione *Phone* nello specchietto a sinistra e a destra selezioniamo *Empty View Activity*. Premiano *Next*, in questa schermata cambiamo Language in *Java*, cliccliamo *Finish* e attendiamo la fine della creazione di questo progetto. Potrebbe volerci un pò per scaricare tutte le dipendenze, soprattutto se la vostra connessione non va molto veloce.




Nuovo Progetto con Android Studio

Appena finito il caricamento, iniziamo aggiungendo il codice dove è presente bArr nel metodo *onCreate* della nostra nuova app, eliminado il riferimeto al

percorso *sg.vantagepoint.a.a* del rigo 6 e sostituendo al posto del *return* (rigo 11) con

```
Log.d("Solved", new String(bArr));
```

per stampare su *Logcat* la frase segreta. Ricorda di aggiungere il pacchetto del Log inserendo

```
import android.util.Log;
```

all'inizio del file.

Al di fuori del metodo onCreate incolliamo i due metodi di supporto per il metodo appena inserito

```
1     public static byte[] b(String str) {
2         int length = str.length();
3         byte[] bArr = new byte[(length / 2)];
4         for (int i = 0; i < length; i += 2) {
5             bArr[i / 2] = (byte) ((Character.digit(str.
                  charAt(i), 16) << 4) + Character.digit(str.
                  charAt(i + 1), 16));
6         }
7         return bArr;
8     }
9
10    public static byte[] a(byte[] bArr, byte[] bArr2) {
11        SecretKeySpec secretKeySpec = new SecretKeySpec(bArr
              , "AES/ECB/PKCS7Padding");
12        Cipher instance = Cipher.getInstance("AES");
13        instance.init(2, secretKeySpec);
14        return instance.doFinal(bArr2);
15    }
```

Se guardate il codice ci saranno dei metodi colorati in rosso, questo perché dobbiamo aggiungere gli import per Base64, Cipher e SecretKeySpec e indicare la gestione delle Eccezioni correlati a questi pacchetti. Per cui aggiungiamo

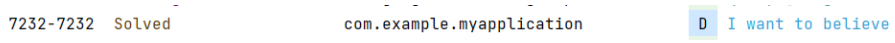
```
1  import android.util.Base64;
2  import javax.crypto.Cipher;
3  import javax.crypto.spec.SecretKeySpec;
4
5  import java.security.InvalidKeyException;
6  import java.security.NoSuchAlgorithmException;
7
8  import javax.crypto.BadPaddingException;
9  import javax.crypto.IllegalBlockSizeException;
10 import javax.crypto.NoSuchPaddingException;
```

e modifichiamo il metodo cosi:

```
1 public static byte[] a(byte[] bArr, byte[] bArr2) throws
      InvalidKeyException, NoSuchPaddingException,
      NoSuchAlgorithmException, IllegalBlockSizeException,
      BadPaddingException {
2     SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "
          AES/ECB/PKCS7Padding");
3     Cipher instance = Cipher.getInstance("AES");
4     instance.init(2, secretKeySpec);
5     return instance.doFinal(bArr2);
6 }
```

Adesso non ci resta che eseguire l'applicazione cliccando il simbolo play verde in alto o premendo la combinazione "*Shift+F10*[1]. Aspettiamo la compilazione dell'app e l'installazione dell'app sull'emulatore, apriamo Logcat e nel campo di ricerca cerchiamo "*Solved*".

Ecco trovata la Frase Segreta: *I want to believe*.




Risultato Estratto da Logcat

Adesso per finire non ci resta che provare la frase ma per fare ciò dobbiamo avere a disposizione in nostro apk modificato per eliminate Anti-Root e un dispositivo dove verificare se la nostra modific funzioni.

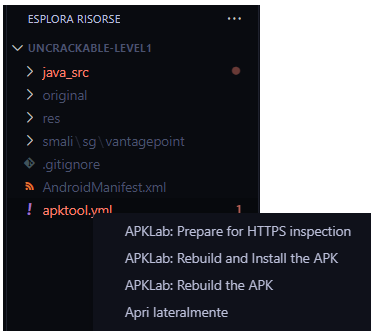
### 3.3.3 Creazione Device Rooted on Android Studio e Build Apk modificato

Per ricompilarlo basta cliccare col tasto destro del mouse sul file *apktool.yml* presente su "*Esplora Risorse*" nella parte sinistra di VS Code e selezionare *APKLab: Rebuild the APK*, alla comparsa del menu a tendina deselezioniamo "*–use-aapt2*" poiché la nostra app non usa questo pacchetto per la generazione del suo codice binario ma *aapt*.

Uno dei motivi per cui ho deciso di utilizzare APKLab è perchè racchiude in se molti tools utili e uno di questi viene usato proprio in questo momento: **APKSigner**.

*Apksigner* ci permette di inserire una firma alla nostra applicazione senza la quale i dispositivi android non vorranno eseguirla. Questa firma non è una forma di blocco dell'esecuzione dell'app, ma più una forma di protezione per l'inserimento dell'app da parte di terzi all'interno dello Store. Essa fa riferimento ad un Certificato dell'azienda creatrice o del creatore dell'apk. Quindi, identifica il proprietario dell'Applicazione Mobile e permette il ridiuto dell'applicazione nel caso in cui, al momento del caricamento di una nuova versione, essa non corrisponda con quella memorizzata. Il sistema usato per queste firme è quello della crittografia Asincrona.

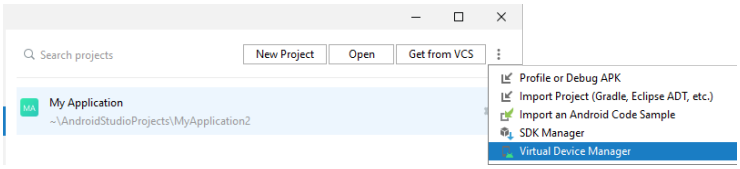
[1] Shift è il tasto Maiuscola della tastiera




APKLab: Rebuild The APK

Nel nostro caso non ci vogliamo spacciare per nessuno, ma vogliamo solo permettere all'applicazione di essere eseguita senza problemi. L'Apk generato verrà salvato all'interno della cartella dist del progetto dell'apk decompilato, che si trova nella stessa cartella del unCrackable1.apk iniziale.

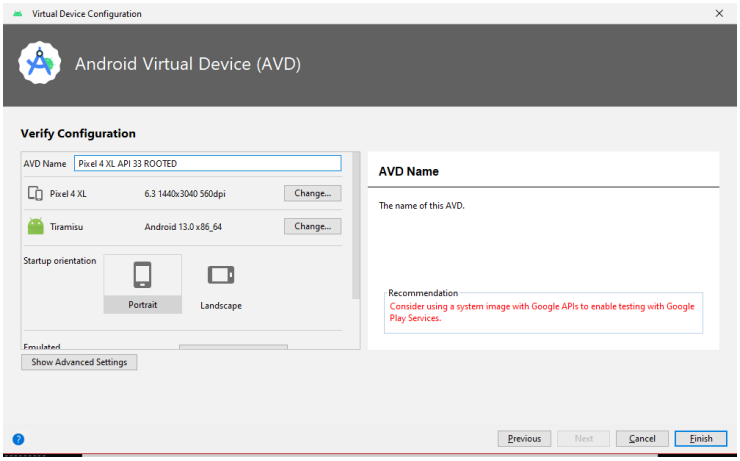
Adesso che abbiamo l'applicazione modificata da noi in formato apk, torniamo su Android Studio e, visto che adesso abbiamo un progetto in lista, per andare su *Visual Device Manager*, dobbiamo premere i tre puntini in alto a destra e selezionare Visual Device Manager.




Accede a Visual Device Manager dalla lista Progetti

All'apertura della seconda pagina contenete le Device, cliccate in alto a sinistra "*Create Device*" e attendiamo l'apertura della procedura guidata per la creazione della nuova Device. Lasciamo la scelta su Phone in "*Category*", dalla lista accanto scegliamo un modello di Smartphone, io ne consiglio uno con una schermo largo cosi da poter vedere meglio. Nel mio caso ho scelto un *Pixel 4 XL* da 6.3".

Clicchiamo Next e adesso ci tocca scegliere l'OS Android da usare. Per il nostro bisogno passiamo alla schermata con scritto *x86 Images* e scegliamone uno dalla lista. Io ho scelto il sistema Tiramisù con API 33 e Android 13, cliccando Next se non avete mai scaricato quella Image, iniziera il suo download. Adesso vi apparirà una schermata di riepilogo in cui io ne ho approfittato per inserire la parola "ROOTED" in AVD Name cosi da poterlo distinguere dagli altri se dovessi avere più Device uguali. Clicchiamo Finish.




Riepilogo creazione Device con Root

Avviamo il nuovo emulatore e per prima cosa installiamo la nostra App iniziale come test per il controllo del Root.

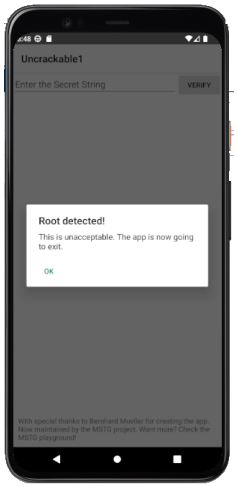
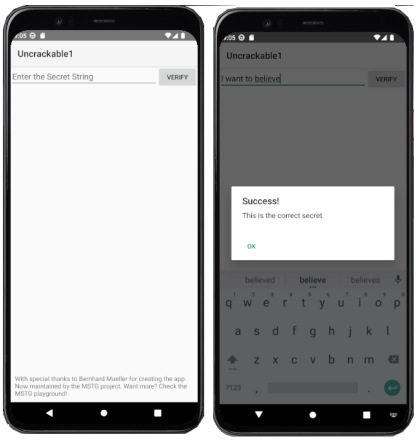



Figura 21: Blocco applicazione per Device Rooted

Come si vede nella Figure21 l'applicazione riconosce il dispositivo come Rooted. Disinstalliamo questa app e tramite il comando

```
.\adb install ..\UnCrackable-Level1\dist\UnCrackable-Level1.apk
```

installiamo la nostra applicazione modificata e essa si avvia senza problemi, adesso inseriamo la Frase Segreta trovata in precedenza ( I want to believe). Siamo riusciti nel nostro intento.




Apertura applicazione modificata senza avviso e inserimento frase

### Contromisure

In questo apk è stato applicato solo il minimo necessario di tecniche di Obfuscation e protezione. L'unica contromisura per la prevensione sarebbe l'aumento di sicurezza per proteggere il metodo di recupero della frase e di utilizzare più tecniche di Obfuscation come lo String Encryption, Dead Code Injection, la sostituzione delle istruzioni e aumentare il livello del Name Obfuscation usato.

# 4 Conclusioni

Scrivere questa relazione non è stato facile. Non è stato facile trovare Applicazioni Mobili sullo Store su cui preparare questo testo perché tutte superavano la mia preparazione. Pur avendo appreso il funzionamento generico e l'uso dei strumenti usati in questa relazione,il Reverse Engineering di tali applicazioni richiederebbe conoscenze specifiche che io non possiedo.

Per questo mi sono affidato ad applicazioni create per questo scopo. Mi è stato impossibile poter fornire un esempio di Reverse Engineering Mobile su sistemi iOS perché non ho potuto trovare gratuitamente un Emulatore iOS adatto e le alternative trovate richiedevano il possesso di un dispositivo fisico della Apple.

Comunque abbiamo parlato delle basi per effettuare un Reverse Engineering sulle applicazioni, di come è possibile effettuarlo, visto degli esempi, di come può essere utile per la sicurezza e per l'analisi dei malware e delle tecniche di prevenzione.

Essere capace di effettuare un Reverse Engineering su un'applicazione e capire, per modificare o altro, il suo funzionamento interno è un grade "superpotere" da avere, ma che richiede grande impegno e dedizione poiché, come affermato anche da OWASP, su questo argomento si possono riempire librerie intere e io so di aver visto solo la punta dell'Iceberg.

Alla fine posso dire, nel mio piccolo, di essere d'accordo con OWASP:
*il Reverse Engineering è un'arte.*

# Riferimenti bibliografici


[1] Wikipedia. *Reverse Engineering* `https://en.wikipedia.org/wiki/Reverse_engineering`

[2] Wikipedia. *Applicazione Web* `https://it.wikipedia.org/wiki/Applicazione_web`

[3] OWASP, Ottobre 2023. *Mobile Application Security Testing Guide (MASTG) v1.7.0.* Disponibile su `https://mas.owasp.org/MASTG/`

[4] OWASP. *Mobile Application Security Verification Standard (MASVS)* Disponibile su `https://mas.owasp.org/MASVS/`

[5] TechTarget. *Reverse Engineering* `https://www.techtarget.com/searchsoftwarequality/definition/reverse-engineering`

[6] OWASP. 2016. *Mobile Top 10.* Estratto da `https://owasp.org/www-project-mobile-top-10/2016-risks/`

[7] OWASP. *Mobile Top 10.* Estratto da `https://owasp.org/www-project-mobile-top-10/`

[8] Wikipedia. *Assembly Language* https://en.wikipedia.org/wiki/Assembly_language

[9] Just Mobile Security, Luglio 2023. *Android Static Analysis Fundamentals: Smali Code Introduction and Modifications.* LinkedIn https://www.linkedin.com/pulse/android-static-analysis-fundamentals-smali-code-introduction

[10] Android developers. *Verifica i Link per app Android* https://developer.android.com/training/app-links/verify-android-applinks?hl=it#web-assoc

[11] Istituto Italiano Edizioni Atlas. *Firma dell'app e pubblicazione nello Store* https://www.edatlas.it/scarica/Informatica/prosia_5_2019/Capitolo5Mobile/MaterialiOnline/FirmaPubblicazioneApp.pdf

## Applications & Tools

[12] Android Studio IDE. https://developer.android.com/studio

[13] SDK Platform Tools. https://developer.android.com/tools/releases/platform-tools

[14] Visual Studio Code, Windows https://code.visualstudio.com/

[15] Surendrajat. *APKLab*, VSCode. https://marketplace.visualstudio.com/items?itemName=Surendrajat.apklab

[16] Apktool, Android. https://github.com/iBotPeaches/Apktool

[17] Jadx, Android https://github.com/skylot/jadx

[18] InsecureShop.apk , Android. https://github.com/hax0rgb/InsecureShop

[19] MAS Crackmes. https://mas.owasp.org/crackmes/

[20] Frida. Dynamic Instrumentation Toolkit https://frida.re/

[21] WireShark https://www.wireshark.org/