



## DIPARTIMENTO DI INFORMATICA

*Corso di Laurea in Informatica*

---

Tesi di Laurea

### ANALISI DELLE MINACCIE ALLA SICUREZZA DI UN DISPOSITIVO ANDROID

Relatore:

Prof. Paolo Buono

Laureando:

Daniele Alessandro  
Matricola n.

708814

*Anno Accademico 2022/2023*

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Obiettivo del progetto . . . . .	3
1.2	Struttura del documento . . . . .	4
1.3	Android . . . . .	4
1.3.1	Architettura . . . . .	5
1.3.2	Componenti principali . . . . .	7
1.4	Sicurezza in Android . . . . .	9
<b>2</b>	<b>Stato dell'arte</b>	<b>15</b>
2.1	OWASP Mobile Application Security . . . . .	15
2.2	Strumenti per l'analisi automatica delle vulnerabilità . . . . .	21
2.3	Strumenti per la visualizzazione . . . . .	25
<b>3</b>	<b>Metodologie e strumenti</b>	<b>27</b>
3.1	Macchina virtuale . . . . .	28
3.2	Genymotion . . . . .	29
3.3	Architettura client-server . . . . .	29
3.4	Android Debug Bridge (ADB) . . . . .	31
3.5	Mobile Security Framework . . . . .	31
3.6	Python . . . . .	34
<b>4</b>	<b>Implementazione</b>	<b>38</b>
4.1	Setup dell'ambiente di lavoro . . . . .	38
4.1.1	Server . . . . .	38
4.1.2	Ambiente di sviluppo . . . . .	40

4.2	Architettura del progetto . . . . .	45
4.3	Ricerca del dispositivo collegato . . . . .	46
4.4	Estrazione degli APK . . . . .	48
4.5	Scansione degli APK . . . . .	51
4.6	Elaborazione dei risultati della scansione . . . . .	54
4.7	Visualizzazione dei risultati . . . . .	56
4.8	Risultati . . . . .	57
4.9	Conclusioni . . . . .	68
4.10	Sviluppi futuri . . . . .	70
	<b>Bibliografia</b>	<b>71</b>

# **Elenco delle tabelle**

# Elenco delle figure

1	Numero di utenti globali che utilizzano dispositivi mobili per l'accesso a internet	1
2	Percentuale di utilizzo dei dispositivi desktop e mobili per l'accesso a internet	2
3	Quota di mercato dei vari sistemi operativi mobili nel mondo . . . . .	5
4	Architettura di Android . . . . .	13
5	Output comando <code>ls -l</code> eseguito in un emulatore Android all'interno della directory <code>/data/data</code> . . . . .	14
6	Output comando <code>ps</code> eseguito in un emulatore Android . . . . .	14
7	Elenco OWASP Mobile Top 10 . . . . .	19
8	Pagina principale MobSF . . . . .	32
9	Fase di analisi MobSF . . . . .	33
10	Prima parte di un report MobSF . . . . .	34
11	Parte della sotto-sezione Code Analysis di un report MobSF . . . . .	35
12	Funzionamento REST API . . . . .	36
13	Installer Python 3.12.2 per Windows 64-bit . . . . .	41
14	Pagina principale di Genymotion con pulsante creazione configurazione . . .	43
15	Selezione configurazione dispositivo . . . . .	44
16	Esempio report JSON per il file cm.aptoide.pt.apk . . . . .	55
17	Attivazione modalità sviluppatore . . . . .	58
18	Abilitazione Debug USB . . . . .	59
19	File eseguibile dell'applicazione . . . . .	60
20	Ricerca del dispositivo . . . . .	60
21	Autorizzazione per eseguire il Debug USB . . . . .	61
22	Dispositivo individuato . . . . .	61

23	Analisi del pacchetto “com.openai.chatgpt.apk” . . . . .	62
24	Punteggio di sicurezza globale . . . . .	62
25	Numero di vulnerabilità individuate ordinate per quantità . . . . .	63
26	Numero di vulnerabilità individuate ordinate per criticità . . . . .	63
27	Lista vulnerabilità individuate . . . . .	64
28	Reindirizzamento al OWASP MASTG per la vulnerabilità “Android Logging”	65
29	Dettagli delle applicazione analizzate . . . . .	66
30	Esempio prime pagine di un report PDF per l’app “Prime Video” . . . . .	67

# Capitolo 1

## Introduzione

L'evoluzione dei dispositivi mobili ha trasformato radicalmente il modo in cui si interagisce con il mondo digitale. In una società sempre più interconnessa, questi dispositivi non sono più semplici strumenti di comunicazione, ma sono diventati veri e propri centri di archiviazione ed elaborazione dati.

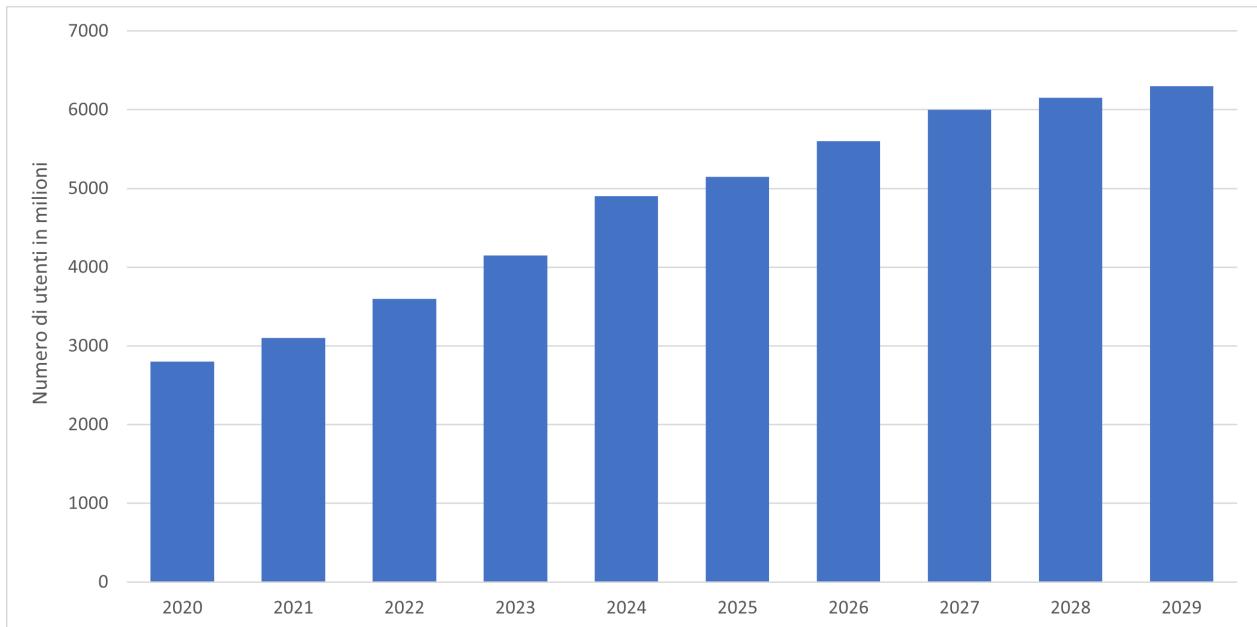


Figura 1: Numero di utenti globali che utilizzano dispositivi mobili per l'accesso a internet

Il numero di utenti di dispositivi mobili è cresciuto in modo significativo, e si prevede continuerà a crescere. Secondo un report di Statista [1], è previsto un notevole incremento del numero di utenti di smartphone nel periodo compreso tra il 2024 e il 2029, con un aumento del 30,6% (1,5 miliardi) di nuovi utenti. Questa crescita porterà il numero totale di utilizzatori a raggiungere la cifra stimata di 6,4 miliardi.

Negli ultimi anni, i dispositivi mobili hanno rapidamente superato l'uso dei computer tradizionali. Secondo uno studio condotto dal Pew Research Center sugli americani, l'81% possiede uno smartphone e il 74% un computer desktop/laptop [2]. Questa tendenza non è destinata a rallentare, con i dispositivi mobili che continuano a trasformare sempre più il modo in cui si interagisce con la tecnologia.

I dispositivi mobili offrono una vasta gamma di servizi e applicazioni, che spaziano dai social network, ai servizi bancari, alla biglietteria e allo shopping, tutti ora facilmente accessibili tramite questi dispositivi.

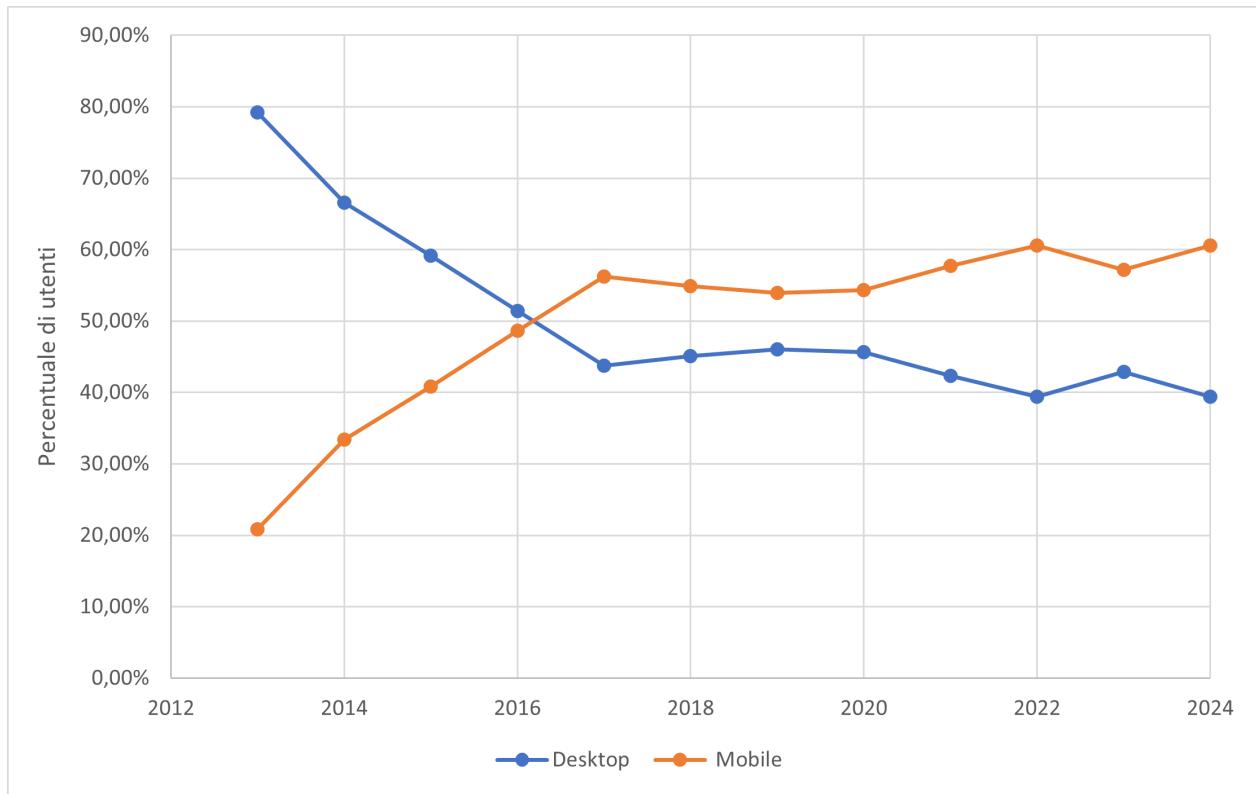


Figura 2: Percentuale di utilizzo dei dispositivi desktop e mobili per l'accesso a internet

Questo andamento trova un’ulteriore conferma nello studio condotto da MobiCloud[3], i cui risultati sono riassunti nel grafico 2.

I dati confermano un notevole incremento nell’utilizzo di Internet su dispositivi mobili rispetto a quelli desktop nel corso degli ultimi 10 anni. Già alla fine del 2016, l’utilizzo di Internet sui dispositivi mobili ha per la prima volta superato quello sui dispositivi desktop. Attualmente, quelli mobili rappresentano il 60,59% del traffico Internet, a differenza di quelli desktop che si fermano al 39,41%.

Con il crescente utilizzo dei dispositivi mobili, cresce anche l’importanza della protezione dei dati usati dalle applicazioni mobili. Poiché ora i dati più sensibili risiedono su tali dispositivi, i malintenzionati stanno gradualmente spostando la loro attenzione su questo ambiente. Di conseguenza, le applicazioni vulnerabili possono diventare dei facili bersagli per gli aggressori [4]. Inoltre, la crescente adozione delle politiche “bring-your-own-device” (BYOD) da parte delle aziende rende ancora più difficile la distinzione tra uso personale e uso aziendale di un dispositivo, aumentando il rischio di esposizione di dati aziendali. In questo contesto, comprendere le minacce, sviluppare sistemi sicuri, ed adottare misure preventive diventa essenziale per proteggere i dati e la privacy degli utenti mobili.

## 1.1 Obiettivo del progetto

L’obiettivo della tesi è semplificare l’analisi della sicurezza delle applicazioni Android, rendendo questo processo accessibile anche a coloro che non possiedono competenze in ambito informatico. Il progetto mira a sviluppare un sistema in grado di fornire un quadro complessivo dello stato di sicurezza del dispositivo, partendo dalle applicazioni installate su di esso. Attraverso un’interfaccia user-friendly e intuitiva, il progetto si propone di tradurre in modo chiaro e semplice le informazioni generate dagli strumenti di analisi statica, consentendo una migliore comprensione dei rischi e delle vulnerabilità delle app installate sui dispositivi degli utenti.

## 1.2 Struttura del documento

Il documento è strutturato come segue:

- Nel capitolo 1 si introduce l'argomento della sicurezza informatica per i dispositivi mobili, fornendo una prospettiva sulle problematiche e sulle motivazioni che hanno portato alla scelta di questo argomento di ricerca. In questo capitolo viene descritto l'ecosistema Android, con le sue componenti e la sua architettura. Segue una panoramica generale riguardante le minacce che si possono incontrare in ambito informatico in generale, e più nello specifico nei dispositivi mobili.
- Nel capitolo 2 si descrive lo stato dell'arte nell'ambito dell'analisi della sicurezza delle applicazioni Android. In questo capitolo viene descritto il progetto OWASP, il quale offre strumenti e tecniche focalizzate al miglioramento della sicurezza. Successivamente, si descrivono gli strumenti disponibili per l'analisi automatica di applicazioni Android, confrontando le diverse soluzioni al fine di identificare il più adatto per il processo di ricerca in questione. Infine, nel paragrafo dedicato agli strumenti per la visualizzazione, si esplorano soluzioni che consentono di semplificare la rappresentazione delle informazioni ottenute come risultato dell'analisi della sicurezza.
- Nel capitolo 3 si esaminano le metodologie e gli strumenti analizzati nel capitolo precedente, con un'attenzione particolare all'integrazione di Python e ADB per ottimizzare e semplificare il processo di analisi.
- Nel capitolo 4, ci si concentra sulle fasi di progettazione e sviluppo del progetto. Si esplorano le scelte effettuate durante il processo di selezione degli strumenti da utilizzare, evidenziandone vantaggi e svantaggi. Si discutono i dettagli del setup dell'ambiente di lavoro, si descrive l'architettura del progetto e si delineano le fasi che attraverserà l'applicazione: dall'estrazione delle applicazioni installate sul dispositivo, all'analisi di tali file, all'elaborazione e alla visualizzazione dei risultati. Infine, si mostrano i risultati di un caso reale utilizzando un dispositivo di un utente.

## 1.3 Android

Tra i sistemi operativi per dispositivi mobili più utilizzati al mondo, Android occupa una posizione di rilievo, come mostrato nella figura 3.

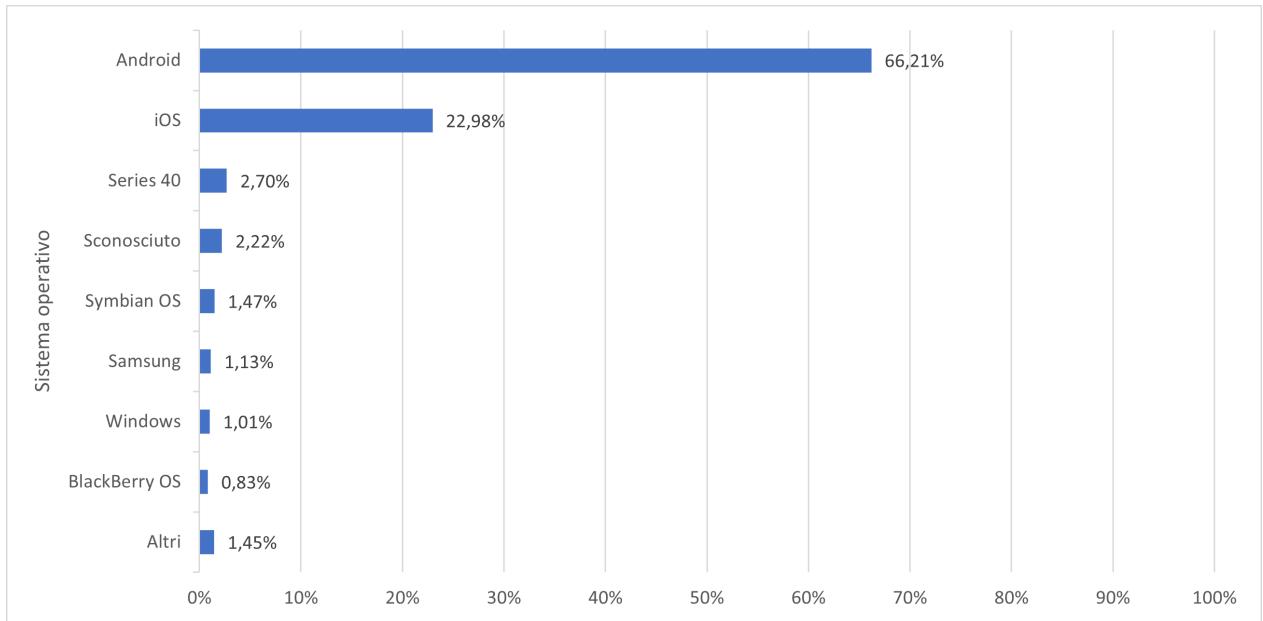


Figura 3: Quota di mercato dei vari sistemi operativi mobili nel mondo

Lanciato per la prima volta da Google nel settembre 2008, Android ha conquistato una vasta utenza, contando attualmente 3,9 miliardi di utenti, ovvero il 48,22% della popolazione mondiale totale [5]. Questo successo è attribuibile a diversi fattori, tra cui un prezzo competitivo, un'elevata personalizzazione, l'alta compatibilità con i vari dispositivi hardware. Un altro aspetto che ha conquistato la preferenza degli utenti è l'ecosistema di Android, che ospita milioni di applicazioni sul Play Store e offre l'integrazione con i servizi Google (Gmail, Google Drive e altri). Inoltre Android nel tempo ha ampliato la sua portata, andando oltre gli smartphone e i tablet ed estendendosi ad una vasta gamma di dispositivi, tra cui smartwatch, televisori, automobili ed altro.

### 1.3.1 Architettura

Come detto precedentemente, Android è ampiamente utilizzato su una vasta gamma di dispositivi. Ciò è possibile grazie alla sua architettura, progettata per rendere il sistema operativo indipendente dall'hardware e capace di adattarsi alle diverse piattaforme.

Come si evince dalla figura 4, il sistema operativo è diviso in strati, dove ognuno di essi utilizza i servizi esposti dallo strato sottostante (di più basso livello) per offrire servizi a

quello sovrastante (di più alto livello). Gli strati che compongono l'architettura Android sono (partendo dal basso verso l'alto):

- Kernel Linux: Si occupa di gestire le risorse hardware del dispositivo, come la memoria, la CPU o le periferiche.
- Hardware Abstraction Layer (HAL): Questo strato si interpone tra il sistema operativo e l'hardware, fornendo l'implementazione dei driver necessari per interagire con l'hardware. Questo strato è di interesse dei produttori, i quali sviluppano e ottimizzano i driver per il proprio dispositivo, garantendo così al sistema operativo un'indipendenza dalle singole componenti hardware.
- Native C/C++ Libraries: Questo strato prevede l'implementazione di librerie scritte nei linguaggi di programmazione C o C++, ovvero in codice nativo, eseguito direttamente, senza dover passare prima dal runtime (ART o Dalvik). Una libreria è un insieme di funzioni o strutture dati definite e predisposte per essere collegate ad un programma software attraverso un opportuno collegamento. Alcune componenti della piattaforma sono scritte in codice nativo, pertanto hanno necessità di tali librerie.
- Android Runtime: definisce l'implementazione dell'esecuzione delle applicazioni.

Prima di descrivere le componenti di questo strato, è importante notare che le applicazioni nell'ecosistema Android di solito sono scritte in Java, ma subiscono varie fasi di compilazione prima di poter essere eseguite direttamente sui dispositivi. L'SDK di Android converte il codice sorgente Java e altre risorse in un pacchetto compresso noto come file .APK [6]. Questo pacchetto Android include il file **AndroidManifest.xml**, che contiene le autorizzazioni Android e altri dati importanti, insieme a un file **classes.dex** che contiene il bytecode Dalvik, oltre ad altre risorse in formato XML o grafico [6].

Questo strato è composto da due diversi componenti: la Virtual Machine e le Core Libraries.

- La Virtual Machine è un ambiente di esecuzione delle applicazioni, dove si effettua la traduzione del bytecode (un linguaggio intermedio, ottenuto dalla compilazione dei file java) in codice dex (un altro tipo di linguaggio intermedio, contenente istruzioni native). I file .dex vengono quindi eseguiti in una macchina virtuale Java personalizzata: Dalvik o ART. La VM (Virtual Machine) Dalvik veniva utilizzata principalmente in passato, prima dell'introduzione di Android 5.0. Era basata su

tecnologia Just-in-time (JIT), il che significa che il codice sorgente Java veniva compilato in bytecode solo parzialmente durante lo sviluppo. Successivamente, durante l'esecuzione dell'applicazione, un interprete software (la Dalvik VM stessa) si occupava di tradurre il codice in bytecode in codice dex in tempo reale. Con l'introduzione di Android 5.0, ART è diventato l'ambiente di esecuzione predefinito. Questo si basa sulla tecnologia di compilazione ahead-of-time (AOT), in cui l'intero codice sorgente delle applicazioni viene compilato durante la fase di installazione dell'applicazione.

- Le core libraries sono un insieme di librerie sviluppate principalmente in Java, ma con alcune dipendenze dal codice nativo. Esse consentono agli sviluppatori di accedere alle funzionalità del linguaggio Java. Inoltre grazie all'utilizzo della Java Native Interface (JNI), il codice nativo può essere integrato all'interno delle librerie Java, facilitando l'interazione bidirezionale tra le applicazioni e il codice nativo.
- Java API Framework: rappresenta un insieme di funzionalità che vengono esposte agli sviluppatori attraverso un set di API scritte in linguaggio Java. Queste API rappresentano gli elementi costitutivi necessari per creare applicazioni Android, semplificando il riutilizzo di componenti e servizi di sistema. Tra i componenti più importanti presenti in questo strato troviamo: l'Activity Manager, che gestisce il ciclo di vita delle Activity dell'applicazione; il Resource Manager, che gestisce le risorse come layout e stringhe; il Location Manager, che fornisce servizi di localizzazione; il Notification Manager, gestisce le notifiche e gli avvisi.
- System Apps: descrive le applicazioni incluse nel sistema operativo (Calendario, Fotocamera, Orologio, Email, ecc.).

### 1.3.2 Componenti principali

Le applicazioni Android sono composte da diversi componenti di alto livello. I componenti principali sono:

- Activity
- Intent
- Broadcast receiver

- Servizi

## Activity

In Android una Activity è una classe Java che permette di rappresentare il concetto di schermata di un'applicazione. Per questo motivo esse contengono tutti gli elementi che costituiscono un'interfaccia utente: fragments, views e layout. Quindi creare un'applicazione significa implementare una o più Activity e definire le eventuali comunicazioni tra esse.

Le Activity hanno un loro ciclo di vita, passando attraverso vari stati gestiti dal sistema operativo. Esse possono trovarsi nei seguenti stati: attive, in pausa, interrotte e inattive. Nell'implementazione di una Activity, è possibile definire specifici metodi che vengono invocati automaticamente dal sistema operativo quando si verifica una transizione tra questi stati.

## Intent

Si è precedentemente detto che creare un'applicazione significa implementare una o più Activity e definire le eventuali comunicazioni tra esse. Il meccanismo degli Intent in Android permettono di fare proprio questo, ovvero permettono la comunicazione e lo scambio di messaggi tra componenti di un'applicazione.

I tre casi d'uso principali degli Intent sono l'avvio di un'activity, l'avvio di un servizio e l'invio di un messaggio broadcast.

Esistono due tipi di Intent: esplicativi e impliciti. Gli Intent esplicativi riguardano un contesto locale dell'applicazione, specificando il nome completo del componente (interno all'applicazione). Gli Intent impliciti, invece, vengono inviati al sistema operativo e specificano un'azione sotto forma di URI. Il sistema operativo verifica quali app possono gestire quell'azione e, se più di un'app è idonea, richiede all'utente di selezionare quale utilizzare.

## Broadcast receiver

I Broadcast Receiver sono componenti che permettono alle applicazioni di ricevere notifiche da altre app e dal sistema. Questi sono strettamente connessi agli Intent e vengono attivati per eseguire azioni specifiche in risposta a Intent specifici, rimanendo inattivi fino a tale attivazione.

## Servizi

I servizi sono componenti che permettono di eseguire operazioni in background, senza interagire direttamente con l'utente.

I servizi sono lanciati usando gli Intent, che rappresentano la descrizione astratta dell'operazione da eseguire.

## 1.4 Sicurezza in Android

L'architettura di Android è stata progettata con un'attenzione particolare alla sicurezza, ereditando il modello di sicurezza dal kernel Linux e incorporando diversi meccanismi per proteggere sia le applicazioni che i dati degli utenti. Questo approccio si riflette nell'utilizzo dell'Android Security Model.

A livello più basso, ogni applicazione viene eseguita come un processo separato, con il proprio insieme di strutture di dati. Questo approccio, noto come sandboxing, assicura che ogni applicazione operi in un ambiente isolato, impedendo ad altre applicazioni o processi di interferire con la sua esecuzione. A livello più alto invece, si utilizza un sistema di permessi per consentire alle applicazioni di interagire con altre applicazioni, componenti o risorse critiche.

### Sandboxing a livello di kernel

Quando una nuova applicazione viene installata su un dispositivo Android, il sistema le assegna due numeri interi, ovvero un UserID (UID) univoco e un GroupID (GID). Gli UID sono numerati in base al tipo di privilegi che ogni applicazione possiede:

- Le applicazioni e i daemon aventi UID 0 sono estremamente rare e generalmente non dovrebbero essere consentite per ragioni di sicurezza. L'UID 0 concede a un'applicazione privilegi elevati, consentendo di eseguire qualsiasi tipo di operazione.
- Le applicazioni e i daemon di sistema partono da 1000, che corrisponde all'utente di sistema, il quale dispone di privilegi speciali (ma pur sempre limitati).
- Gli UID assegnati automaticamente alle applicazioni utente partono da 10000.

Ogni applicazione installata ha un insieme di strutture dati e file che sono associati al suo UID e GID, ed i permessi di accesso a queste strutture e file sono consentiti solo all'applicazione avente stesso ID o all'utente con i massimi privilegi di accesso (root). Di conseguenza le altre applicazioni non posso accederci perché non hanno i privilegi necessari. Inoltre quando si avviano le applicazioni, queste vengono eseguite in processi separati identificati dai rispettivi UID, contenuti quindi nelle rispettive sandbox. Questo permette alle applicazioni di utilizzare il codice nativo (e le librerie native) senza preoccuparsi delle implicazioni per la sicurezza: se ne occupa Android.

Da notare che Android è progettato per offrire un'esperienza utente singola, a differenza di Linux, che è un sistema operativo multi-utente. Android, quindi, sfrutta il modello di sicurezza multi-utente e lo applica alle applicazioni. In altre parole, anziché gestire contemporaneamente gli utenti, Android gestisce simultaneamente le applicazioni tramite la gestione dei permessi basata su Linux.

Ogni applicazione installata su Android ha una propria cartella all'interno della directory */data/data*.

La figura 5 mostra il comando:

```
$ ls -l
```

Il comando **ls -l** è utilizzato nei sistemi Unix per elencare i file e le directory presenti in una determinata cartella, in questo caso si vuole mostrare il contenuto di */data/data*. L'opzione **-l** specifica di visualizzare i risultati in un formato dettagliato mostrando informazioni aggiuntive quali permessi, proprietario, gruppo, dimensione e altri dettagli.

La gestione degli UID delle applicazioni avviene attraverso i file di sistema **packages.xml** e **packages.list**, situati nella directory */data/system*. Questi file vengono utilizzati da Android per memorizzare informazioni sui pacchetti delle applicazioni installate sul dispositivo. A differenza del file **packages.xml**, che contiene informazioni dettagliate sui pacchetti delle applicazioni, **packages.list** è più una lista semplificata utilizzata internamente dal sistema operativo per scopi legati alla gestione delle applicazioni, inclusi processi come il monitoraggio delle installazioni e delle disinstallazioni.

Il comando **ps** fornisce un elenco dei processi in esecuzione e le relative informazioni di stato. Come si può vedere in nella figura 6, ogni processo (applicazione) appartiene all'UID corrispondente. Ad esempio in questo caso il processo **com.google.android.gms.persistent** è

di proprietà di `a_93` (UID 93), che è stato assegnato all'applicazione durante il processo di installazione.

## Permessi e Android Manifest

A livello di applicazione, Android implementa un sistema di gestione dei permessi che garantisce il controllo dell'accesso a determinate funzionalità o risorse del dispositivo.

I permessi Android si trovano in genere nel file `AndroidManifest.xml`; sono utilizzati dagli sviluppatori per dichiarare i diritti di accesso di cui ha bisogno la loro applicazione per essere eseguita [7]. All'interno di questo file, i permessi sono definiti utilizzando il tag `<uses-permission>`, che specifica il tipo di accesso richiesto dall'applicazione. Android fornisce anche una serie di permessi predefiniti per funzionalità di base, ma le applicazioni possono definirne altri in base alle esigenze specifiche.

Un concetto centrale di questo sistema dei permessi è che nessuna applicazione ha automaticamente l'autorizzazione per eseguire operazioni che potrebbero influenzare altre applicazioni, il sistema operativo o l'utente stesso. Questo include l'accesso ai dati privati dell'utente, la lettura o la scrittura dei file di altre applicazioni, l'accesso alla rete e altre operazioni ritenute rischiose.

I permessi sono suddivisi in diversi livelli di protezione, che influenzano se e come l'utente deve concedere l'autorizzazione all'applicazione. Solo i permessi considerati “pericolosi” richiedono il consenso esplicito dell'utente. Il modo in cui Android richiede l'autorizzazione dipende dalla versione del sistema operativo e dalla versione dell'applicazione in esecuzione sul dispositivo dell'utente.

I livelli di protezione sono i seguenti:

**Normal** Riguardano aree in cui l'applicazione accede a dati o risorse al di fuori della sua sandbox, ma dove il rischio è minimo o perfino nullo. Durante l'installazione, il sistema concede automaticamente l'autorizzazione per questi permessi e non richiede l'approvazione dell'utente. Gli utenti non possono revocare tali autorizzazioni.

**Dangerous** Riguardano aree in cui un'applicazione richiede l'accesso a dati o risorse che coinvolgono informazioni sensibili dell'utente o il funzionamento di altre applicazioni. Per esempio, il permesso di leggere i contatti salvati in rubrica. Per ottenere questo tipo di

permessi, l'utente deve concedere esplicitamente l'autorizzazione all'applicazione. Fino a quando l'utente non approva l'autorizzazione, l'applicazione non può fornire funzionalità che dipendono da tale permesso.

**Signature** Vengono concessi al momento dell'installazione, ma solo se l'applicazione che richiede il permesso è firmata con lo stesso certificato dell'applicazione che definisce il permesso. Questo livello di protezione garantisce che solo le applicazioni affidabili, firmate con lo stesso certificato, possano accedere a determinate funzionalità o risorse.

**Signature or system** Sono soddisfatti solo dalle applicazioni che si trovano nel sistema o che sono firmate con lo stesso certificato di quelle presenti nel sistema.

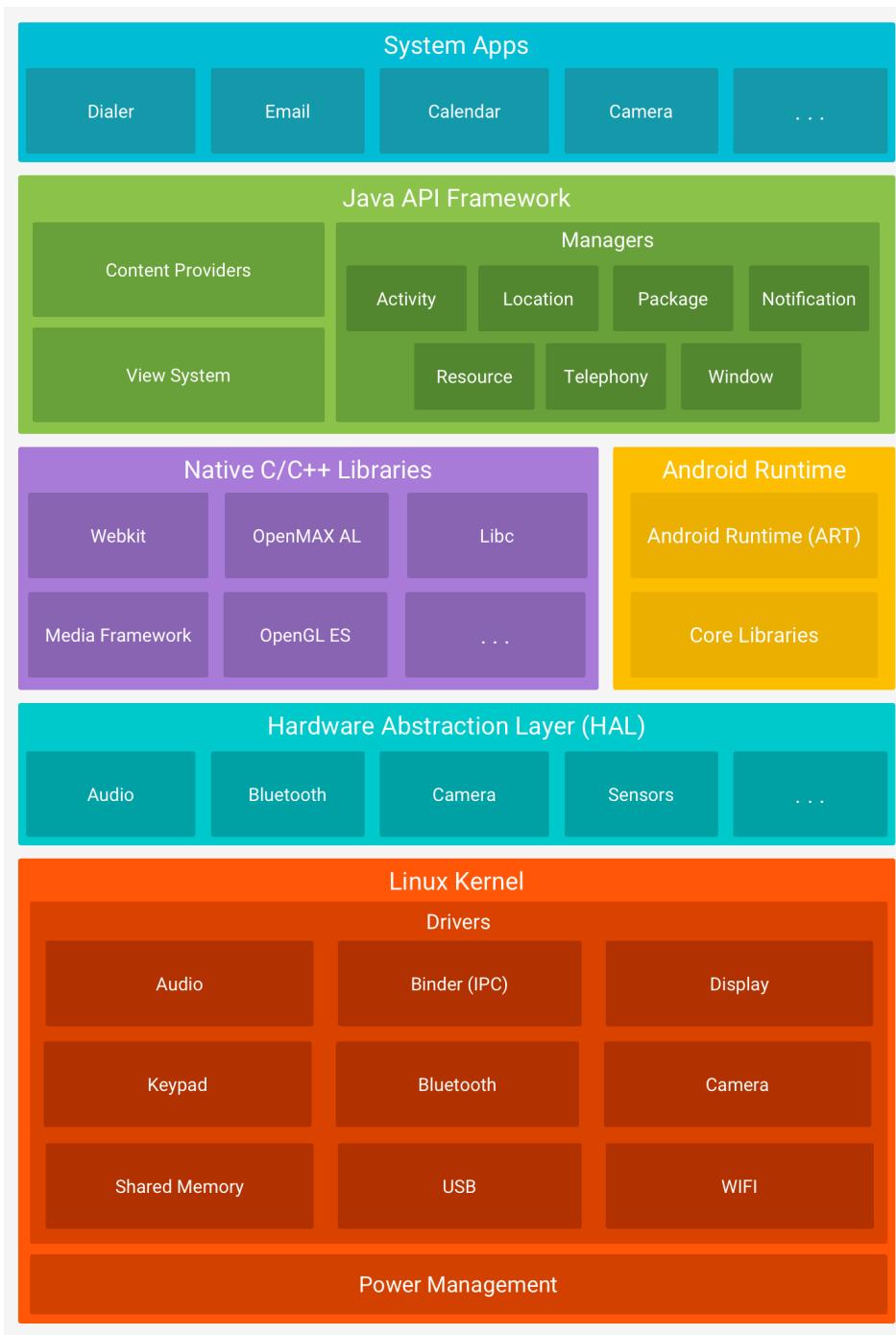


Figura 4: Architettura di Android

```
vbox86p:/ # cd /data/data
vbox86p:/data/data # ls -l
total 540
drwxr--r-- 4 system      system        4096 2024-02-09 12:07 android
drwxr--r-- 7 u0_a83      u0_a83        4096 2024-03-05 10:18 android.ext.services
drwxr--r-- 4 u0_a31      u0_a31        4096 2024-02-09 12:07 android.ext.shared
drwxr--r-- 8 u0_a87      u0_a87        4096 2024-02-09 12:17 cm.aptoides.pt
drwxr--r-- 4 u0_a46      u0_a46        4096 2024-02-09 12:07 com.amaze.filemanager
drwxr--r-- 4 u0_a85      u0_a85        4096 2024-02-09 12:07 com.android.adservices.api
drwxr--r-- 4 u0_a19      u0_a19        4096 2024-02-09 12:07 com.android.backupconfirm
drwxr--r-- 4 u0_a22      u0_a22        4096 2024-02-09 12:07 com.android.bips
drwxr--r-- 4 bluetooth    bluetooth     4096 2024-02-09 12:07 com.android.bluetooth
drwxr--r-- 4 u0_a38      u0_a38        4096 2024-02-09 12:07 com.android.bluetoothmidiservice
drwxr--r-- 4 u0_a45      u0_a45        4096 2024-02-09 12:07 com.android.bookmarkprovider
drwxr--r-- 5 u0_a64      u0_a64        4096 2024-02-09 12:07 com.android.calendar
drwxr--r-- 4 u0_a12      u0_a12        4096 2024-02-09 12:07 com.android.callogbackup
drwxr--r-- 5 u0_a59      u0_a59        4096 2024-02-09 12:07 com.android.camera2
drwxr--r-- 4 u0_a40      u0_a40        4096 2024-02-09 12:07 com.android.cameraextensions
drwxr--r-- 4 u0_a44      u0_a44        4096 2024-02-09 12:07 com.android.captiveportallogin
drwxr--r-- 4 u0_a65      u0_a65        4096 2024-02-09 12:07 com.android.carrierconfig
drwxr--r-- 4 u0_a42      u0_a42        4096 2024-02-09 12:07 com.android.carrierdefaultapp
drwxr--r-- 4 u0_a27      u0_a27        4096 2024-02-09 12:07 com.android.cellbroadcastreceiver
drwxr--r-- 4 u0_a84      u0_a84        4096 2024-02-09 12:07 com.android.cellbroadcastreceiver.module
drwxr--r-- 4 network_stack network_stack 4096 2024-02-09 12:07 com.android.cellbroadcastservice
drwxr--r-- 4 u0_a33      u0_a33        4096 2024-02-09 12:07 com.android.certinstaller
drwxr--r-- 4 u0_a32      u0_a32        4096 2024-02-09 12:07 com.android.companiondevicemanager
drwxr--r-- 4 u0_a77      u0_a77        4096 2024-02-09 12:07 com.android.connectivity.resources
drwxr--r-- 5 u0_a50      u0_a50        4096 2024-02-09 12:07 com.android.contacts
drwxr--r-- 4 u0_a29      u0_a29        4096 2024-02-09 12:07 com.android.cts.ctsshim
drwxr--r-- 4 u0_a11      u0_a11        4096 2024-02-09 12:07 com.android.cts.priv.ctsshim
drwxr--r-- 6 u0_a62      u0_a62        4096 2024-02-09 12:07 com.android.deskclock
drwxr--r-- 4 system      system        4096 2024-02-09 12:07 com.android.development
drwxr--r-- 7 u0_a51      u0_a51        4096 2024-02-09 12:07 com.android.dialer
drwxr--r-- 6 u0_a20      u0_a20        4096 2024-02-09 12:07 com.android.documentsui
drwxr--r-- 4 u0_a34      u0_a34        4096 2024-02-09 12:07 com.android.dreams.basic
drwxr--r-- 4 u0_a56      u0_a56        4096 2024-02-09 12:07 com.android.dreams.photatable
```

Figura 5: Output comando `ls -l` eseguito in un emulatore Android all'interno della directory `/data/data`

u0_a82	1442	365	13597384	105824	0	0 S com.android.providers.media.module
system	1462	365	13535212	86948	0	0 S com.genymotion.settings
system	1478	365	13535076	85584	0	0 S com.genymotion.systempatcher
system	1496	365	13531488	83872	0	0 S com.genymotion.tasklocker
system	1512	365	13535288	88520	0	0 S com.genymotion.genyd
u0_a93	1542	365	13907860	251640	0	0 S com.google.android.gms.persistent

Figura 6: Output comando `ps` eseguito in un emulatore Android

# Capitolo 2

## Stato dell'arte

### 2.1 OWASP Mobile Application Security

In un'era in cui i dispositivi mobili si sono trasformati da semplici strumenti di comunicazione a veri e propri centri di archiviazione e gestione dati, la sicurezza delle applicazioni è diventata un tema cruciale, evidenziando la necessità di proteggere queste informazioni sensibili. Un aspetto da considerare è la continua evoluzione delle minacce. La natura dinamica delle applicazioni, con aggiornamenti frequenti e nuove funzionalità, richiede un costante adattamento delle strategie di sicurezza. La rapidità con cui nuove vulnerabilità vengono scoperte e sfruttate sottolinea l'importanza di un approccio proattivo alla sicurezza delle applicazioni. Diversamente dai tradizionali paradigmi di sicurezza, la protezione delle informazioni sensibili nelle applicazioni mobili non può essere affrontata con approcci standard, come la scansione antivirus basata sulla firma, ma richiede una visione più contestuale. In questo scenario complesso, organizzazioni come OWASP giocano un ruolo chiave nel fornire linee guida, best practice e strumenti per affrontare i problemi relativi alla sicurezza delle applicazioni.

OWASP, acronimo di Open Web Application Security Project, è un progetto open-source dedicato a migliorare la sicurezza del software. Fondato nel 2001, OWASP riunisce esperti, sviluppatori e professionisti della sicurezza informatica con l'obiettivo di identificare e mitigare le vulnerabilità del software. Il progetto fornisce risorse, strumenti e linee guida per aiutare le organizzazioni a sviluppare, acquisire e manutenere applicazioni software sicure. Nel contesto specifico delle applicazioni mobili, l'OWASP Mobile Application Security (MAS) è un

progetto che fornisce uno standard di sicurezza (OWASP MASVS) e un manuale completo per i test di sicurezza (OWASP MASTG).

Tra i progetti più importanti sviluppati dall'OWASP MAS ci sono:

- Mobile Application Security Verification Standard (MASVS)
- Mobile Application Security Testing Guide (MASTG)
- OWASP Mobile Top 10

### **Mobile Application Security Verification Standard**

È lo standard di settore per la sicurezza delle app mobili. Il MASVS definisce due livelli di verifiche della sicurezza, noti come MASVS-L1 (Standard security) e MASVS-L2 (Defense in depth). Vi è inoltre un terzo livello, denominato MASVS-R, creato specificamente per il reverse engineering e che generalmente viene utilizzato parallelamente ad uno dei due livelli precedenti. [8]

- MASVS-L1 (Standard Security): rappresenta il livello base di verifica della sicurezza e include controlli essenziali che ogni applicazione dovrebbe implementare per proteggere i dati e prevenire attacchi comuni. Esso si concentra su misure di sicurezza di base come l'autenticazione, la gestione delle sessioni, la crittografia dei dati sensibili e la protezione contro le vulnerabilità note.
- MASVS-L2 (Defense in depth): rappresenta un livello avanzato di verifica della sicurezza. Include controlli aggiuntivi e più rigorosi rispetto al MASVS-L1. Questo livello richiede una protezione più completa e personalizzata contro minacce più sofisticate. Ad esempio, può richiedere la verifica dell'integrità del codice, la protezione degli algoritmi crittografici, la gestione delle autorizzazioni, la protezione delle risorse di sistema e altre misure di sicurezza avanzate.
- Il livello di resilienza (MASVS-R), si concentra sulla protezione dell'applicazione contro tentativi di analisi e de-compilazione del codice sorgente. Il reverse engineering può essere utilizzato da malintenzionati per identificare vulnerabilità o per creare versioni modificate dell'applicazione con scopi malevoli. Questo livello suggerisce misure come l'offuscamento del codice, l'implementazione di controlli anti-debugging e l'utilizzo di strumenti di protezione del codice per rendere più difficile l'analisi e la manipolazione dell'applicazione.

L'obiettivo di questo standard è definire una serie di requisiti di sicurezza condivisi per valutare la sicurezza delle applicazioni mobili. In questo modo, si assiste gli sviluppatori nella creazione di applicazioni più sicure e si aiuta i revisori di sicurezza nel valutare la sicurezza delle applicazioni in modo più efficiente.

I requisiti standard definiti dal MASVS coprono le principali aree di sicurezza per le applicazioni mobili, e sono:

- V01: Architecture, Design, and Threat Modeling (Architettura, Progettazione e Modello delle Minacce): Questo requisito richiede che l'applicazione mobile abbia un'architettura sicura e un processo di progettazione che includa una valutazione delle minacce potenziali per identificare e mitigare i rischi di sicurezza.
- V02: Data Storage and Privacy (Archiviazione dei Dati e Privacy): Questo requisito si concentra sulla protezione dei dati sensibili immagazzinati nell'applicazione mobile, richiedendo l'utilizzo di tecniche di crittografia adeguata a proteggere i dati in riposo.
- V03: Cryptography (Crittografia): Questo requisito copre l'uso corretto della crittografia nell'applicazione mobile, compresi l'uso di algoritmi di crittografia robusti e l'implementazione sicura di funzionalità di crittografia.
- V04: Authentication and Session Management (Autenticazione e Gestione delle Sessioni): Questo requisito riguarda l'implementazione di meccanismi di autenticazione sicura per verificare l'identità degli utenti e la gestione delle sessioni per garantire che le sessioni siano correttamente gestite e protette.
- V05: Network Communication (Comunicazione di Rete): Questo requisito richiede che le comunicazioni tra l'applicazione mobile e i server siano crittografate utilizzando protocolli sicuri come HTTPS per proteggere i dati trasmessi dall'intercettazione malevola.
- V06: Platform Interaction (Interazione con la Piattaforma): Questo requisito si riferisce all'interazione dell'applicazione mobile con la piattaforma sottostante, inclusi i controlli di sicurezza necessari per proteggere l'applicazione da minacce come il jailbreak/rooting del dispositivo.
- V07: Code Quality and Build Settings (Qualità del Codice e Impostazioni di Compilazione): Questo requisito riguarda la scrittura di codice sicuro e la configurazione

corretta delle impostazioni di compilazione per evitare vulnerabilità come le falle di sicurezza nel codice sorgente o le configurazioni errate.

- V08: Reverse Engineering and Tampering (Reverse Engineering e Manomissione): Questo requisito richiede misure per proteggere l'applicazione mobile dall'analisi del codice sorgente o dall'alterazione malevola, ad esempio attraverso l'implementazione di controlli di sicurezza e la rilevazione delle minacce.

La prima categoria si concentra principalmente sulla fase di progettazione, comprendente i processi che si svolgono durante questo passaggio e che sono difficili da verificare tramite strumenti software automatizzati. Per le altre categorie, invece, è possibile identificare diversi strumenti che, attraverso analisi statica, dinamica o una combinazione di entrambe, sono in grado di individuare e evidenziare le vulnerabilità di un'applicazione specifica.

Gli strumenti disponibili verranno esaminati successivamente, ma prima di descriverli è utile distinguere tra i due tipi di analisi. L'analisi statica consente di controllare l'isolamento delle informazioni (taint analysis), verificare l'assenza di interferenze tra dati riservati e pubblici (information flow analysis) e analizzare le sequenze di operazioni eseguite (control flow analysis). Tuttavia, queste tecniche presentano limitazioni teoriche, come la complessità, che ne limitano l'applicabilità.

L'analisi dinamica può superare alcune di queste limitazioni ed è utilizzabile nella maggior parte dei contesti. Tuttavia, le tecniche basate sull'esecuzione del software non riescono a coprire in modo esaustivo tutti i possibili comportamenti, fornendo quindi solo garanzie parziali. [9].

## **Mobile Application Security Testing Guide**

Il Mobile Application Security Testing Guide (MASTG) è un manuale completo che fornisce una guida dettagliata sui processi tecnici necessari per verificare i controlli indicati nel MAVS. Questo strumento si focalizza su un approccio completo per condurre i test, includendo metodologie sia black box che white box. Per ogni aspetto critico segnalato all'interno del documento, viene quindi definito un insieme di test, sia di natura statica che dinamica, da effettuare per certificarne la presenza all'interno dell'applicazione analizzata.



Figura 7: Elenco OWASP Mobile Top 10

## OWASP Mobile Top 10

È un elenco delle principali vulnerabilità che possono compromettere la sicurezza delle applicazioni mobili. Questa lista viene periodicamente aggiornata da OWASP in seguito ad analisi e studi sulle nuove minacce, fornendo per ognuna di esse una descrizione, un esempio di attacco, le possibili conseguenze e una serie di contromisure per mitigare il rischio.

Facendo riferimento alla OWASP Mobile Top 10 più recente, aggiornata al 2024, ecco una descrizione di ciascuna vulnerabilità:

- Improper Platform Usage [M1]:

Si manifesta quando un'applicazione non utilizza correttamente le funzionalità di sicurezza fornite dalla piattaforma mobile. Affinché questa vulnerabilità possa essere sfruttata, l'applicazione deve esporre un servizio web o una chiamata API che implementano in modo inappropriate i meccanismi di crittografia, i controlli di accesso o le autorizzazioni. Di conseguenza, attraverso l'interfaccia mobile, un potenziale attaccante può fornire input malevoli o sequenze inaspettate di eventi all'endpoint vulnerabile.

- Insecure Data Storage [M2]:

Si manifesta quando un'applicazione memorizza dati sensibili in modo non sicuro sul dispositivo mobile. Questa vulnerabilità viene sfruttata quando l'applicazione archivia i dati in chiaro o utilizzando algoritmi di crittografia deboli. Di conseguenza, gli aggressori possono accedere a queste informazioni tramite l'accesso al filesystem.

- Insecure Communication [M3]:

Si manifesta quando un'applicazione mobile non implementa correttamente metodi di sicurezza durante la comunicazione con i server backend o altri servizi esterni. Questa vulnerabilità viene sfruttata quando l'applicazione utilizza protocolli non sicuri o trasmette dati senza crittografia, consentendo agli aggressori di intercettare e manipolare il traffico di rete. L'assenza di crittografia durante la trasmissione rende vulnerabile il flusso di informazioni tra il dispositivo mobile e i server. Questo può permettere agli attaccanti di ottenere accesso non autorizzato ai dati sensibili degli utenti.

- Insecure Authentication [M4]:

Si manifesta quando un'applicazione mobile implementa in modo non sicuro il processo di autenticazione degli utenti. Questa vulnerabilità viene sfruttata quando l'applicazione utilizza metodi di autenticazione obsoleti. Questo include l'utilizzo di password deboli, la mancanza di protezione contro attacchi bruteforce o la presenza di account predefiniti non sicuri. Questi problemi possono consentire agli aggressori di ottenere accesso non autorizzato all'applicazione o ai dati associati.

- Insufficient Cryptography [M5]:

Si manifesta quando un'applicazione mobile utilizza un algoritmo di crittografia non sicuro. Questo consente ad un attaccante di ottenere dati sensibili nella loro forma originale, non crittografata.

- Insecure Authorization [M6]:

Si manifesta quando un'applicazione mobile presenta metodi di autorizzazione difettosi o mancanti, consentendo a un utente non autorizzato di eseguire azioni privilegiate o accedere a risorse riservate.

- Client Code Quality [M7]:

Si manifesta quando un'applicazione presenta problemi di sicurezza derivanti dalla scarsa qualità del codice sorgente. Gli attacchi tipici per questa vulnerabilità sfruttano perdite di memoria, buffer overflow e altri problemi che si traducono in una cattiva pratica di programmazione.

- Code Tampering [M8]:

Si manifesta quando il codice dell'applicazione mobile viene compromesso o alterato da un aggressore. Un attaccante può modificare il codice, cambiando dinamicamente il contenuto della memoria, sostituire le API utilizzate o modificare i dati e le risorse dell'applicazione. Questo fornisce all'aggressore un metodo diretto per alterare il comportamento dell'applicazione a proprio vantaggio.

- Reverse Engineering [M9]:

Si manifesta quando un attaccante esegue il reverse engineering dell'applicazione mobile, ottenendo il codice sorgente e altre informazioni sensibili. Affinché questa vulnerabilità possa essere sfruttata, è necessario che l'applicazione sia scritta in linguaggi o framework che consentono un'introspezione dinamica durante il runtime, come Java, .NET, Objective C o Swift.

- Extraneous Functionality [M10]:

Si manifesta quando un'applicazione include funzionalità non necessarie o non utilizzate nella sua implementazione e lasciate abilitate nella versione pubblica. Gli aggressori possono esaminare i file di log, i file di configurazione e il binario dell'app per scoprire switch nascosti o codice di test non disabilitato.

## 2.2 Strumenti per l'analisi automatica delle vulnerabilità

Android, presente non solo in smartphone ma anche in tablet e sistemi di intrattenimento domestico, ha conquistato una predominanza significativa nel panorama tecnologico. Secondo i dati forniti da StatsCounter nel novembre 2019, Android detiene una quota di mercato del 75,85%, superando di gran lunga il 22,87% di iOS [10]. Questo considerevole utilizzo ha aumentato la sua esposizione a potenziali minacce, con l'aumento di malware appositamente progettato per questo sistema operativo. Un rapporto di Fortinet del 2011 evidenzia l'esistenza di circa 2000 campioni di malware per Android appartenenti a 80 famiglie diverse [11]. Oltre alle minacce di malware, si sono osservate crescenti preoccupazioni legate alla privacy, poiché molte applicazioni tracciano gli utenti per scopi pubblicitari, ma spesso sono a rischio di abusi o intrusioni della privacy attraverso spyware. Le vulnerabilità riscontrate nelle applicazioni tradizionali si sono trasferite nel panorama Android, richiedendo approcci

specifici. Con questo scopo sono nati strumenti open-source dedicati all’analisi della sicurezza delle applicazioni mobili. Questi strumenti mirano a svolgere funzioni cruciali, tra cui l’identificazione delle potenziali vulnerabilità all’interno delle applicazioni, il reverse engineering degli APK (Android Application Packages) e la generazione di rapporti con informazioni utili sull’analisi.

Tra gli strumenti di analisi statica e dinamica di applicazioni Android trovati in letteratura abbiamo:

- Mobile Security Framework (MobSF)
- Mobile Application Reverse engineering and Analysis Framework (MARA Framework)
- ImmuniWeb
- AndroBugs

## **Mobile Security Framework**

Rappresenta uno strumento all-in-one per l’assessment di vulnerabilità delle applicazioni mobili, supportando i principali sistemi operativi come Android ed iOS.

È un progetto open source realizzato utilizzando Python, implementando un’interfaccia grafica accessibile sia tramite web browser che tramite terminale. Questo strumento è utilizzato per l’analisi statica e dinamica di sicurezza delle applicazioni mobili e viene raccomandato da organizzazioni come l’OWASP. Una volta caricato l’APK, MobSF esegue un’analisi dettagliata identificando le vulnerabilità presenti nell’applicazione. Il report risultante fornisce i dettagli sulle minacce individuate, ordinandole in base alla loro criticità. Inoltre, a fine analisi MobSF genera un indice numerico che varia da 0 a 100 e rappresenta il livello di sicurezza generale dell’applicazione.

Infine, un’altra caratteristica di MobSF è la possibilità di interfacciarsi con il tool tramite REST API, consentendo l’integrazione nella propria pipeline di CI/CD. Grazie a queste API è possibile automatizzare il processo di analisi ed elaborazione dei risultati, ottenendo ad esempio il report in formato JSON.

Uno svantaggio di questo strumento è la sua velocità di analisi. Questa risulta essere piuttosto lenta, soprattutto con applicazioni grandi e complesse. Questo è dovuto in parte all’assenza di

strumenti avanzanti per il reverse engineering, che risulta essere la fase più lunga del processo di analisi [12].

## MARA Framework

È un framework open source progettato per testare la sicurezza delle applicazioni secondo gli standard OWASP, utilizzando una serie di strumenti integrati. MARA unisce due delle funzioni più richieste nei tools di penetration testing, ossia la reverse engineering e l'analisi delle vulnerabilità. L'interfaccia utente è pensata per l'uso tramite terminale, facilitando l'iterazione per sviluppatori ed analisti. Tra le sue molteplici funzionalità, MARA permette la ricerca di vulnerabilità attraverso l'uso di strumenti come AndroBugs ed offre diversi metodi per la decompilazione degli APK utilizzando strumenti quali apktool ed enjarify. Una volta completate le operazioni di analisi, MARA restituirà delle cartelle contenenti tutti i risultati ottenuti, quali ad esempio il codice sorgente deoffuscato ed i report delle analisi di sicurezza. Uno svantaggio di questo strumento è l'assenza di API, non permettono una maggiore automatizzazione dell'analisi (ad esempio nel gestire e scansionare più applicativi insieme [13]).

## ImmuniWeb

ImmuniWeb è uno strumento di analisi automatica per applicazioni mobili, accessibile tramite un'interfaccia web utilizzabile tramite browser. Dopo la registrazione sul sito web, è possibile caricare l'APK da analizzare. L'analisi avviene automaticamente e ImmuniWeb restituisce un report delle vulnerabilità rilevate, classificate in base a quattro livelli di pericolosità: high risk, medium risk, low risk e warning. Inoltre, queste ultime sono categorizzate secondo le 10 categorie della top ten OWASP. Oltre alle vulnerabilità, lo strumento mostra tutti i permessi richiesti dall'applicazione e li cataloga in tre categorie: dangerous, normal ed unknown. ImmuniWeb non supporta l'analisi dinamica delle applicazioni e a causa della sua natura proprietaria, è necessario il pagamento per utilizzare le API messe a disposizione dagli sviluppatori [14].

## AndroBugs

È un framework open source progettato con lo scopo di analizzare le vulnerabilità di Android effettuando un'analisi statica del codice. Sviluppato in Python 3. Durante l'analisi, AndroBugs calcola l'hash dell'applicazione, estraе la sua firma e decompila l'APK per analizzare il

codice Java alla ricerca di vulnerabilità e linee guida non rispettate. Al termine dell'analisi, il tool genera un report contenente le vulnerabilità rilevate, assegnando una gravità a ciascuna di esse. I risultati sono salvati in un file testuale, o su un DBMS non relazionale. AndroBugs manca di una GUI, consentendo l'iterazione esclusivamente tramite terminale. Inoltre non mette a disposizione degli sviluppatori delle API [15].

## Confronto riassuntivo

Di seguito viene riassunto quali sono i vantaggi e svantaggi per ciascuno di questi strumenti:

	MobSF	MARA	ImmuniWeb	AndroBugs
Open-source	SI	SI	NO	SI
Supporto multi-piattaforma (Windows, Linux e macOS)	SI	SI	SI	Solo Windows e Linux
Supporta sia l'analisi statica che dinamica	SI	SI	NO	NO
Supporta sia Android che iOS	SI	NO	SI	NO
Raccomandato da OWASP	SI	NO	NO	NO
Supporto per l'integrazione tramite REST API	SI	NO	A pagamento	NO
Assegna un punteggio di sicurezza per ogni minaccia individuata	SI	NO	NO	NO
Generazione report analisi di sicurezza	PDF, JSON	File testuale	PDF	File testuale o DBMS
Fornisce strumenti avanzati di reverse engineering	NO	SI	NO	NO
Richiede registrazione per l'utilizzo	NO	NO	SI	NO
Interfaccia utente	Terminale, web	Terminale	Web	Terminale
Velocità di analisi	Lenta	Alta	Intermedia	Alta

È importante notare che non tutti questi strumenti forniscono metodi di visualizzazione che possono assistere gli utenti nell'esplorazione dei risultati da essi prodotti. E, se lo fanno, nessuno dei tool attualmente disponibili consente una visualizzazione completa dell'analisi, integrando più applicazioni insieme.

## 2.3 Strumenti per la visualizzazione

Esistono numerosi strumenti e librerie progettati per l'implementazione di metodi di visualizzazione che coprono una vasta gamma di esigenze e ambiti applicativi.

Nel contesto di questo progetto, ci si è concentrati sull'utilizzo di Python come linguaggio di programmazione. Python è noto per la sua versatilità e per la vasta comunità di sviluppatori che supporta attivamente il linguaggio. Inoltre, Python offre un'ampia gamma di librerie specializzate, tra cui librerie grafiche per la creazione di interfacce utente (UI).

Di seguito, vengono descritte le librerie più note:

### Tkinter

Tkinter è una libreria per la creazione di interfacce utente grafiche che offre una vasta gamma di widget, tra cui finestre, buttoni, caselle di testo, etichette ed altre componenti, che possono essere integrate per creare interfacce interattive. Ideale per applicazioni di piccole e medie dimensioni che necessitano una GUI di base grazie alla sua semplicità e facilità d'uso. Le funzionalità grafiche avanzate sono quindi limitate rispetto ad altre librerie. Essendo parte integrante di Python, non richiede installazioni aggiuntive.

### PyQt

PyQt è una libreria Python per il framework Qt, un toolkit di sviluppo GUI multi-piattaforma che offre la possibilità di creare interfacce utente sofisticate e professionali. Utilizzata principalmente in ambito professionale, è supportata da un'ottima documentazione e da una grande comunità di sviluppatori. Con PyQt è possibile creare una vasta gamma di elementi, come finestre, dialoghi, menu e barre degli strumenti, e supporta la creazione di applicazioni cross-platform per Windows, macOS e Linux.

### Kivy

Kivy è una libreria open-source multi-piattaforma per lo sviluppo di applicazioni mobili e desktop. Supporta il multitouch ed offre una sintassi dichiarativa per una rapida prototipazione. Kivy è particolarmente utile per applicazioni che richiedono una grafica avanzata e interazioni touch.

## wxPython

wxPython è una libreria per il framework wxWidgets che fornisce un set completo di strumenti per la creazione di interfacce utente native per diverse piattaforme. In questa libreria, la curva di apprendimento è leggermente più ripida rispetto alle alternative descritte precedentemente. Inoltre wxPython supporta il binding di librerie C++ per l'integrazione di funzionalità esistenti.

## Confronto riassuntivo

Di seguito viene riassunto quali sono i vantaggi e svantaggi per ciascuno di questi strumenti:

	PyQt	Tkinter	Kivy	wxPython
Permette la creazione di interfacce utente	Sofisticate	Base	Sofisticate	Sofisticate
Prestazioni adatte per applicazioni	Piccole, medie, grandi	Piccole, medie	Mobile	Piccole, medie
Funzionalità grafiche avanzate	SI	NO	SI	SI
Aspetto visivo allineato con lo stile moderno dei SO	SI	NO	SI	SI
Ottimizzata per aggiornamenti rapidi dell'interfaccia utente	SI	NO	SI	NO
Curva di apprendimento	Ripida	Graduale	Ripida	Graduale
Supporta il multitouch	NO	NO	SI	NO
Integrazione diretta con Python	NO	SI	NO	NO

# Capitolo 3

## Metodologie e strumenti

In questo capitolo, ci si concentra sull'illustrare le metodologie, gli strumenti e le tecniche utilizzati nel progetto per sviluppare un sistema che, a partire dalle applicazioni installate su un dispositivo, fornisca un quadro complessivo dello stato di sicurezza di quest'ultimo.

Tra gli strumenti di analisi statica esaminati nel capitolo 2, si è optato per MobSF. Immuniweb è stato escluso a causa della sua natura proprietaria e delle relative restrizioni, come la necessità di registrazione per condurre l'analisi. La scelta finale si è dunque limitata tra MobSF e il framework MARA, considerando che AndroBugs è incluso in quest'ultimo. MobSF ha prevalso su MARA poiché offre risultati più gestibili. Si è dimostrato [16] che MobSF presenta i report in modo più chiaro e strutturato rispetto a MARA, il quale tende a disperdere le informazioni in modo caotico. MobSF fornisce inoltre una categorizzazione più dettagliata delle vulnerabilità, con una presentazione logica e ordinata dei risultati. Le differenze nella categorizzazione delle vulnerabilità hanno influenzato anche i falsi positivi, con MARA che ne presenta di più rispetto a MobSF e con alcune discrepanze nella classificazione delle vulnerabilità. MobSF risulta quindi essere un'opzione più efficace e user-friendly per l'analisi degli APK. Inoltre, MARA non offre funzionalità come la generazione di report in formato JSON e l'accesso tramite API, rendendo più complesso automatizzare il processo di analisi rispetto a MobSF. Queste due funzionalità aggiuntive consentono un'automazione più completa del processo di analisi, permettendo di estrarre in modo programmatico le informazioni più rilevanti e renderle disponibili in modo chiaro e semplificato per l'utente.

## 3.1 Macchina virtuale

Una macchina virtuale è un software progettato per replicare il comportamento di una macchina fisica all'interno di un sistema operativo ospite. Questo processo, chiamato virtualizzazione, consente di eseguire più sistemi operativi su un singolo hardware fisico e fornisce un ambiente isolato per ciascun sistema operativo.

Le macchine virtuali possono essere categorizzate in base a diversi criteri che ne definiscono le caratteristiche. I tre criteri principali sono:

- Metodo di comunicazione con l'hardware sottostante: Questo definisce come avviene la comunicazione tra le macchine virtuali e le macchine reali sottostanti. Ciò può avvenire tramite l'uso di un hypervisor (virtualizzazione completa), o attraverso l'interazione diretta con il sistema operativo ospite (virtualizzazione basata su software).
- Complessità della virtualizzazione: Questo aspetto si riferisce a quanto della configurazione e delle caratteristiche dell'hardware reale vengono replicate all'interno della macchina virtuale. Le tipologie possono variare da una virtualizzazione completa, dove viene emulata l'intera configurazione hardware, a una virtualizzazione parziale, dove solo alcune componenti vengono emulate.
- Grado di invasività: Indica quanto il sistema operativo reale sia influenzato dall'esecuzione di una macchina virtuale. Alcune macchine virtuali possono operare in modo completamente isolato, senza influenzare il sistema operativo ospite, mentre altre potrebbero richiedere una maggiore interazione con il sistema operativo o l'hardware su cui è installato il software di virtualizzazione.

In base alle categorie sopra elencate, è possibile distinguere le macchine virtuali di processo e le macchine virtuali di sistema.

### Macchine virtuali di sistema

Questa tipologia di macchina virtuale si basa sull'emulazione completa di un sistema operativo e di un hardware. Il suo obiettivo principale è eseguire un sistema operativo completo, chiamato sistema guest, permettendo l'esecuzione di programmi applicativi su di esso. Questo tipo di macchine virtuali richiedono un hypervisor, che alloca le risorse a ciascuna macchina

virtuale. In questo progetto si è utilizzata questo tipo di macchina per emulare il sistema operativo Android, utilizzando lo strumento Genymotion.

### **Macchine virtuali di processo**

Sono macchine virtuali che vengono eseguite come una normale applicazione all'interno del sistema operativo host. Forniscono un ambiente di esecuzione indipendente dalle specifiche dell'hardware e del sistema operativo, offrendo un elevato livello di astrazione e consentendo all'applicazione di operare su diverse piattaforme.

Queste macchine virtuali operano attraverso un interprete e spesso utilizzano la compilazione Just-in-Time per ottenere prestazioni comparabili a quelle dei programmi in linguaggi nativi. L'esecuzione del codice è gestita dall'istanza della macchina virtuale, che si occupa dell'interpretazione del codice e dell'interazione con il sistema operativo sottostante.

Un esempio di utilizzo di macchine virtuali di processo è il sistema operativo Android, con le macchine virtuali: Dalvik Virtual Machine e ART. Queste macchine virtuali sono temporanee e vengono create quando un utente avvia un'applicazione specifica, per poi essere distrutte una volta terminata quell'applicazione.

## **3.2 Genymotion**

Genymotion è un emulatore Android progettato per assistere gli sviluppatori e gli analisti della sicurezza nel testare le applicazioni in un ambiente virtuale sicuro e isolato. Il funzionamento di Genymotion si basa sulla creazione di una macchina virtuale tramite VirtualBox, che fornisce un emulatore Android completo di supporto per sensori hardware come GPS, accelerometro e batteria. Questo strumento offre la possibilità di simulare una vasta gamma di configurazioni hardware e versioni di Android, consentendo l'utilizzo di una varietà di dispositivi virtuali con specifiche differenti.

## **3.3 Architettura client-server**

L'architettura client-server è un modello di comunicazione che organizza e gestisce le risorse in un sistema informatico, suddividendo i ruoli e le responsabilità tra due componenti principali: il client e il server.

Il client è il componente che chiede risorse e dati al server. È rappresentato dall’interfaccia grafica che permette agli utenti di richiedere e interagire con i servizi forniti dal server. In pratica, il client è l’applicazione che consente all’utente di ottenere informazioni o eseguire operazioni attraverso la rete.

Il server invece fornisce i servizi, le risorse e i dati al client. Ha il compito di accettare le richieste provenienti dai client, elaborarle e restituire l’output delle rispettive richieste. Solitamente, il server è una macchina dedicata per tale scopo. Rispetto ai client, i server sono infatti configurati per essere più potenti e affidabili, in modo da poter gestire efficacemente le richieste di più client contemporaneamente.

Il modello client-server si basa sull’architettura di rete, dove il client e il server comunicano tramite protocolli standard come HTTP (Hypertext Transfer Protocol), TCP/IP (Transmission Control Protocol/Internet Protocol), o altri protocolli specifici per l’applicazione. Il client invia una richiesta al server tramite la connessione di rete stabilita attraverso Internet; successivamente, il server elabora tale richiesta e invia i risultati al client, garantendo così lo scambio di informazioni tra le due entità.

Questo modello offre diversi vantaggi. Tra i principali, vi è lo sviluppo modulare, che consente agli sviluppatori di lavorare separatamente sia sul client che sul server, utilizzando tecnologie specifiche per ciascuna componente. Ciò favorisce una maggiore specializzazione e ottimizzazione delle competenze, migliorando l’efficienza dello sviluppo. Inoltre, poiché il client e il server sono componenti separate, è possibile aggiornarli indipendentemente l’uno dall’altro. Ad esempio, se viene introdotto un nuovo miglioramento nell’interfaccia utente, questo può essere distribuito senza la necessità di modificare il codice o la configurazione del server. Questa flessibilità facilita la fase di manutenzione e riduce il rischio di interruzioni del servizio.

In questo progetto il client è rappresentato dall’interfaccia che permette agli utenti finali di richiedere e interagire con i servizi di analisi delle applicazioni Android offerti dal server, per avviare e monitorare il processo di analisi. Il server è costituito dall’infrastruttura informatica che ospita l’analizzatore di APK. Esso fornisce i servizi necessari per eseguire l’analisi delle applicazioni, inclusa l’archiviazione dei dati, il calcolo e l’elaborazione delle informazioni

## 3.4 Android Debug Bridge (ADB)

Android Debug Bridge (ADB) è uno strumento di sviluppo fornito da Google che facilita la comunicazione con un dispositivo Android.

Una volta stabilita la connessione, ADB offre una vasta gamma di funzionalità accessibili tramite la shell Unix del dispositivo, permettendo di interagire e comunicare con il sistema operativo.

Tramite l'Android Debug Bridge, è possibile eseguire una serie di operazioni utili per l'analisi delle applicazioni Android. Ad esempio, è possibile ispezionare le cartelle e i file presenti nella memoria interna del dispositivo, visualizzare il contenuto dei singoli file, estrarre i pacchetti per un'analisi successiva, o anche installare, disinstallare e gestire le applicazioni.

## 3.5 Mobile Security Framework

Nel secondo capitolo, ci si è concentrati sulla descrizione di Mobile Security Framework (MobSF), delineandone le caratteristiche e le proprietà generali. Di seguito invece, si procede con un'analisi più dettagliata di questo strumento, esaminando le funzionalità di analisi e le API offerte.

### Funzionalità dell'analisi

Nella figura 8 è possibile osservare la pagina principale di MobSF.

Dopo aver ricevuto l'applicazione mobile da analizzare, viene avviato il processo di caricamento del file. Una volta completato, MobSF avvia la fase di analisi dell'APK. Durante questa fase, vengono calcolati gli hash del file e vengono estratte le informazioni rilevanti dell'applicazione, compresa l'estrazione delle risorse, del file manifest e dei file di configurazione. Inoltre, vengono elaborati i dettagli sulla composizione dell'applicazione, come i permessi necessari, i servizi e i provider.

Successivamente, MobSF procede con la decompilazione del file APK, estraendo le relative classi. A questo segue un'analisi del codice Java per individuare vulnerabilità note e verificare il rispetto delle linee guida OWASP.

Infine genera un report contenente queste informazioni.

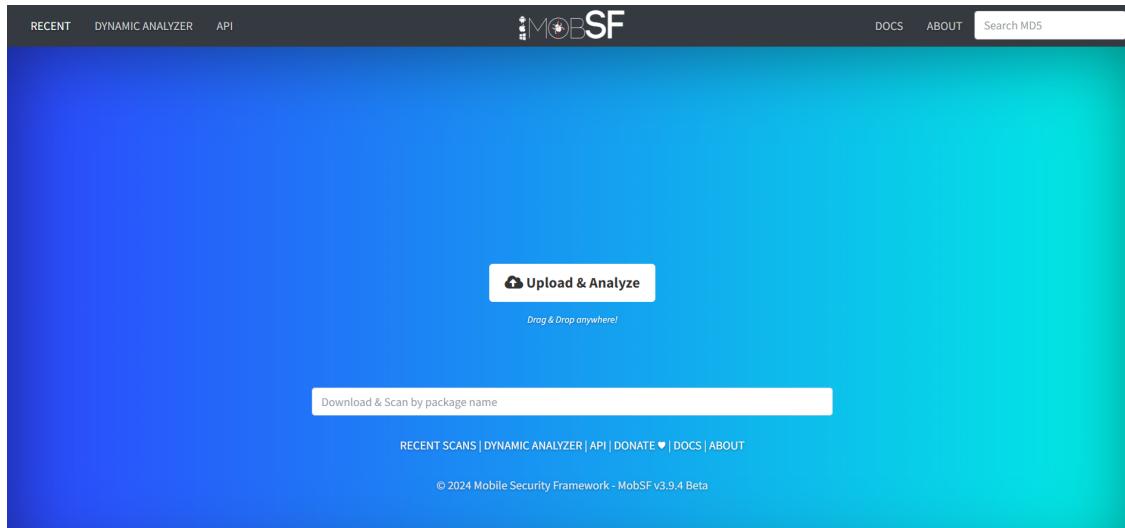


Figura 8: Pagina principale MobSF

Come evidenziato nella figura 10, il report generato da MobSF è suddiviso in diverse sezioni.

La prima sezione fornisce informazioni generali sull'applicazione, come il nome del file e le dimensioni. Altre sezioni riguardano le componenti dell'applicazione e le sue proprietà, come il certificato di firma digitale associato, i permessi richiesti, le informazioni sull'utilizzo delle API Android ed altro ancora.

Particolarmente rilevante è la sezione “Security Analysis”, in cui vengono elencate le vulnerabilità di sicurezza individuate durante l'analisi statica. Questa sezione è fondamentale per comprendere le possibili minacce presenti nell'applicazione.

All'interno della sezione “Security Analysis”, troviamo la sotto-sezione “Code Analysis”, che riguarda il codice sorgente. Per ogni rilevazione, MobSF descrive la vulnerabilità individuata, fornendo un collegamento al Mobile Application Security Testing Guide (MASTG) specifico per quella determinata problematica. Inoltre, viene assegnata una valutazione di gravità in base all'impatto potenziale che tale problematica può avere sull'utente finale.

## REST API offerte

Le REST API, acronimo di “Representational State Transfer Application Programming Interfaces”, sono un tipo di architettura software per i sistemi distribuiti.

Le REST API permettono a due sistemi software di comunicare tra loro attraverso il web in

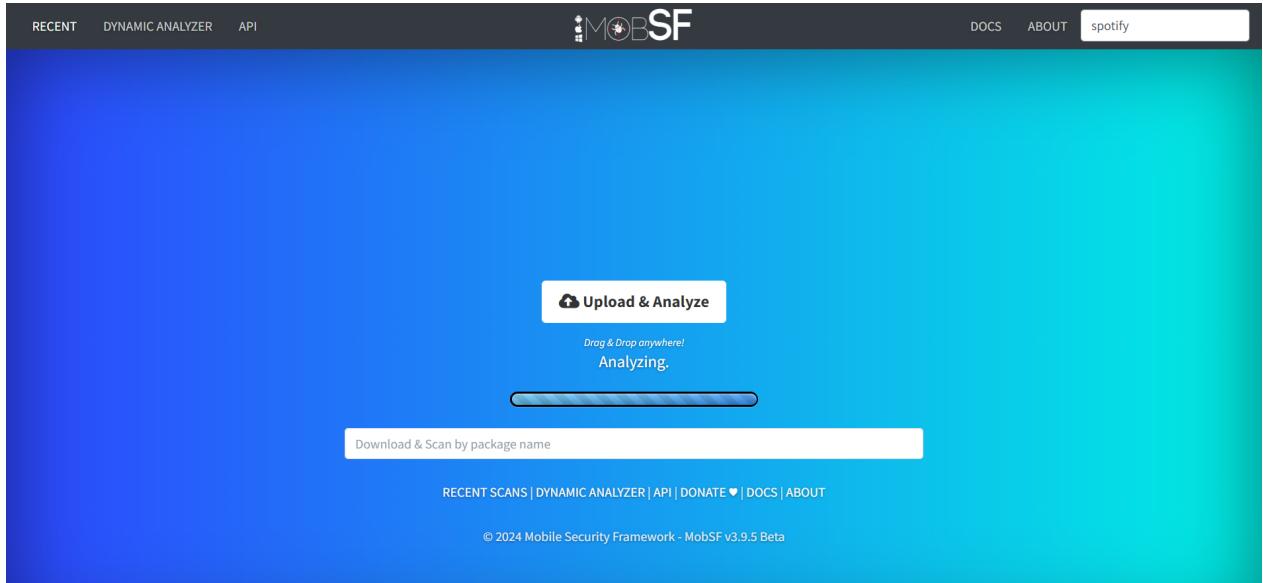


Figura 9: Fase di analisi MobSF

modo standardizzato. Consentono l’invio di richieste HTTP (come GET e POST) al server per accedere e manipolare risorse, dati o funzionalità, gestite da tale server. Le risposte del server sono restituite in un formato standard (come JSON), e forniscono le informazioni richieste dall’applicazione client. Le API si basano sui principi di REST, che è uno stile di progettazione per reti ipertestuali.

Le caratteristiche principali sono:

- Statelessness (Stato Nullo): Ogni richiesta del client al server deve contenere tutte le informazioni necessarie per comprendere e soddisfare la richiesta.
- Uniform Interface (Interfaccia Uniforme): Le API devono avere un’interfaccia uniforme per semplificare l’interazione tra client e server. Questo comprende l’uso di URI (Uniform Resource Identifiers) per identificare risorse, metodi HTTP standard (GET, POST, PUT, DELETE) per indicare l’azione richiesta, e la rappresentazione dei dati in JSON per comunicare le informazioni tra client e server.
- Cacheability (Possibilità di Cache): Le risposte del server possono essere contrassegnate come cacheable (possibili da memorizzare nella cache) o non-cacheable (non adatte per la memorizzazione nella cache), consentendo una gestione della cache per migliorare le prestazioni delle applicazioni.

Figura 10: Prima parte di un report MobSF

- Client-Server Separation (Separazione Client-Server): Client e server devono essere separati tra di loro, consentendo loro di evolversi indipendentemente e migliorando così la scalabilità e la portabilità del sistema.

## 3.6 Python

Python è un linguaggio di programmazione utilizzato in vari ambiti applicativi, tra cui la data science, lo sviluppo web, le applicazioni desktop e altro ancora.

Creato da Guido van Rossum nel 1980, la sua prima versione, Python 0.9.0, è stata rilasciata nel febbraio 1991. Il nome “Python” è un omaggio alla serie televisiva comica britannica “Monty Python’s Flying Circus”, di cui van Rossum era un grande fan.

È un linguaggio pseudo compilato. Ovvero invece di essere tradotto direttamente in linguaggio macchina, il codice sorgente è soggetto ad una fase preliminare in cui viene prima convertito in byte code. Questo byte code, un linguaggio intermedio tra il linguaggio macchina e quello di programmazione, viene riutilizzato dopo la prima esecuzione del programma. Questo approccio elimina la necessità di reinterpretare il sorgente ad ogni esecuzione, migliorando l’efficienza.

NO	ISSUE	SEVERITY	STANDARDS	FILES	OPTIONS
1	This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.	secure	<b>OWASP MASVS:</b> MSTG-NETWORK-4	<a href="#">Show Files</a>	<a href="#">🔗</a>
2	MD5 is a weak hash known to have hash collisions.	warning	<b>CWE:</b> CWE-327: Use of a Broken or Risky Cryptographic Algorithm <b>OWASP Top 10:</b> M5: Insufficient Cryptography <b>OWASP MASVS:</b> MSTG-CRYPTO-4	<a href="#">Show Files</a>	<a href="#">🔗</a>
3	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	<b>CWE:</b> CWE-312: Cleartext Storage of Sensitive Information <b>OWASP Top 10:</b> M9: Reverse Engineering <b>OWASP MASVS:</b> MSTG-STORAGE-14	<a href="#">Show Files</a>	<a href="#">🔗</a>
4	IP Address disclosure	warning	<b>CWE:</b> CWE-200: Information Exposure <b>OWASP MASVS:</b> MSTG-CODE-2	<a href="#">Show Files</a>	<a href="#">🔗</a>

Figura 11: Parte della sotto-sezione Code Analysis di un report MobSF

Un'altra caratteristica di Python è la sua ampia libreria, che contiene moduli e funzioni progettate per una vasta gamma di operazioni. Questi includono l'elaborazione dei dati, l'interazione con i database, la creazione di interfacce grafiche e altro ancora. In questo modo si riduce la necessità di scrivere codice da zero, rendendo questo linguaggio uno dei più ricchi e comodi da usare.

## PyQt

Per quanto riguarda lo sviluppo di interfacce grafiche, PyQt è una libreria progettata per la creazione di interfacce utente sofisticate. Questa libreria permette di utilizzare una vasta gamma di elementi, tra cui finestre, dialoghi, menu e barre degli strumenti. Supporta la creazione di applicazioni cross-platform per Windows, macOS e Linux. Un altro vantaggio è la sua documentazione molto ampia e dettagliata.

Un aspetto rilevante di PyQt sono i layout manager. Questi manager si occupano di gestire automaticamente il posizionamento e le dimensioni dei widget in base alle dimensioni della finestra e alle preferenze dell'utente. In questo modo si automatizza e semplifica la disposizione dei widget all'interno delle finestre dell'applicazione.

Inoltre, questa libreria adotta il meccanismo di segnali e slot per gestire gli eventi e le inte-

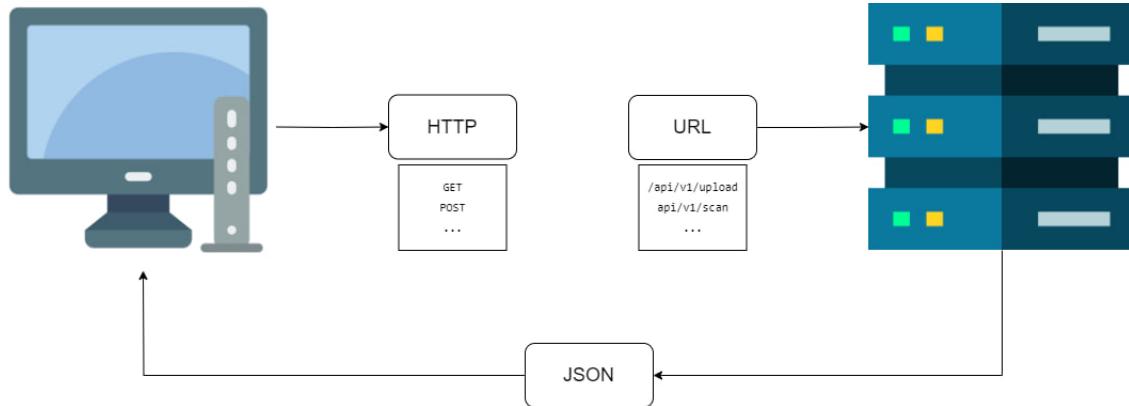


Figura 12: Funzionamento REST API

razioni dell’utente. I widget possono emettere segnali in risposta a determinate azioni (ad esempio, quando viene premuto un pulsante), i quali sono collegati a degli slot, ossia funzioni Python che verranno eseguite in risposta al segnale emesso. Questo modello di programmazione event-driven permette di gestire le interazioni dell’utente nelle applicazioni dotate di una GUI.

PyQt offre anche un supporto per la gestione dei thread, consentendo lo sviluppo di applicazioni multithread. Questo è particolarmente utile per eseguire operazioni lunghe o pesanti, come ad esempio il caricamento di dati da una rete, senza bloccare l’interfaccia utente principale dell’applicazione.

## PyInstaller

PyInstaller è una libreria progettata per semplificare il processo di distribuzione delle applicazioni Python, convertendo gli script Python in eseguibili autonomi per diverse piattaforme, tra cui Windows, macOS e Linux. Questo rende l’installazione e l’esecuzione dell’applicazione più semplice per gli utenti finali.

Una delle sue caratteristiche più importanti è la gestione automatica delle dipendenze dell’applicazione, che include sia i moduli Python esterni sia le librerie di sistema necessarie per il funzionamento dell’applicazione stessa.

Questa libreria offre diverse opzioni di personalizzazione per il processo di confezionamento dell’applicazione. È possibile definire l’aspetto dell’eseguibile finale, inclusi dettagli come

icone personalizzate e informazioni sull'applicazione. Inoltre, consente anche di includere file supplementari, come ad esempio le configurazioni.

Un altro vantaggio di PyInstaller è rappresentato dalla sua vasta comunità di sviluppatori e dalla disponibilità di una documentazione dettagliata.

# Capitolo 4

## Implementazione

Questo capitolo illustra le fasi di sviluppo del progetto, il cui obiettivo è fornire un quadro complessivo dello stato di sicurezza del dispositivo mobile attraverso l'analisi delle applicazioni installate su di esso. Utilizzando i metodi e gli strumenti descritti nei capitoli precedenti, il progetto mira a semplificare il processo di valutazione della sicurezza, facilitando l'individuazione di eventuali vulnerabilità e rischi associati alle applicazioni installate.

### 4.1 Setup dell'ambiente di lavoro

#### 4.1.1 Server

Il server ospita gli strumenti necessari per eseguire l'analisi statica delle applicazioni Android, fungendo da piattaforma centralizzata in cui gli utenti possono inviare le richieste di analisi e ricevere i risultati. Questo approccio elimina la necessità per l'utente di installare e configurare manualmente lo strumento di analisi di sicurezza sul proprio computer.

**Piattaforma utilizzata** Il server è stato configurato su Parrot Security OS versione 5.3, una distribuzione Linux basata su Debian nota per le sue caratteristiche di penetration testing, privacy e sicurezza, sviluppata in Italia da Lorenzo Palinuro Faletra. Questa distribuzione utilizza il kernel Linux 6.1 e il desktop environment MATE. Il processo di installazione dei vari strumenti descritto successivamente è pertanto applicabile a questo sistema

operativo specifico, ma le istruzioni sono generalmente valide anche per altre distribuzioni Debian-based, come ad esempio Ubuntu, POP OS e Debian stesso.

## Installazione dei requisiti

**Docker** È una piattaforma software che semplifica la distribuzione e la gestione di applicazioni tramite container. I container Docker sono pacchetti che racchiudono tutti gli elementi necessari per l'esecuzione di un'applicazione, inclusi codice, librerie e dipendenze. Questo consente di eseguire l'applicazione su diverse infrastrutture.

Per installare MobSF, è necessario Docker. È disponibile un'immagine contenitore preconfigurata che racchiude tutte le risorse necessarie per avviare e utilizzare lo strumento di analisi statica di sicurezza. L'installazione può essere effettuata lanciando il comando:

```
$ sudo apt install docker.io
```

**MobSF** È uno strumento utilizzato per condurre l'analisi statica di sicurezza sugli APK. Per installarlo, è possibile utilizzare Docker, lanciando il seguente comando:

```
$ docker pull opensecurity/mobile-security-framework-mobsf:latest
```

Il comando *pull* di Docker consente di scaricare un'immagine da un registro Docker. Un registro Docker è un servizio di archiviazione centralizzato che contiene le immagini Docker, ossia dei pacchetti che contengono tutto il necessario per poter eseguire un'applicazione su diverse piattaforme. I registri Docker fungono da repository per queste immagini, consentendo agli sviluppatori di distribuire, condividere e recuperare le immagini delle loro applicazioni. In questo caso, il registro utilizzato è Docker Hub, fornito da Docker, Inc. Nel comando, si specifica l'immagine che si desidera scaricare (*opensecurity/mobile-security-framework-mobsf*), e si specifica il tag *latest*, che indica l'ultima versione disponibile di quell'immagine.

Una volta scaricata l'immagine, è possibile avviare MobSF con il seguente comando:

```
$ docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

Di seguito le specifiche di tale comando:

- **docker run:** Questo comando viene utilizzato per avviare un contenitore Docker.

- `-it`: Questi due flag combinati (`-i` e `-t`) sono usati per assegnare un terminale interattivo al contenitore, consentendo allo sviluppatore di interagire direttamente con MobSF tramite riga di comando.
- `-rm`: Questo flag indica a Docker di rimuovere il contenitore una volta che l'esecuzione di MobSF è terminata. Ciò è utile per liberare risorse dopo l'utilizzo del contenitore.
- `-p 8000:8000`: Questa opzione mappa la porta 8000 del contenitore (dove MobSF è in esecuzione) alla porta 8000 del sistema host. In questo modo, è possibile accedere all'interfaccia di MobSF utilizzando l'indirizzo alla porta 8000.
- `-opensecurity/mobile-security-framework-mobsf:latest`: Questo è il nome dell'immagine Docker che si desidera avviare.

Una volta completato il processo di avvio, MobSF sarà pronto per ricevere input dall'utente. È possibile interagire con MobSF all'indirizzo sulla porta 8000 tramite l'interfaccia web oppure tramite REST API. Questo permette di avviare l'analisi statica di sicurezza sull'APK fornito in input.

### 4.1.2 Ambiente di sviluppo

L'ambiente di sviluppo ospita gli strumenti necessari per la realizzazione e l'implementazione del progetto. Qui vengono installati e configurati i vari strumenti di sviluppo descritti nei capitoli precedenti, tra cui il linguaggio di programmazione Python, le librerie fondamentali e l'emulatore Genymotion utilizzato per il testing.

**Piattaforma utilizzata** L'ambiente di sviluppo è stato configurato su un sistema operativo Windows 10 versione 22H2, noto per la sua ampia adozione sia tra gli utenti non esperti che tra gli sviluppatori ed esperti del settore [17]. La sua diffusione fornisce un ecosistema stabile e ben supportato, garantendo compatibilità con una vasta gamma di strumenti e risorse utili.

#### Installazione dei requisiti

**Python** È il linguaggio di programmazione scelto per sviluppare il progetto. Avendo scelto come piattaforma Windows, è possibile installarlo scaricando direttamente l'installer presente sul sito ufficiale. Nello specifico, si andrà a scegliere l'ultima versione stabile di Python, che attualmente è la 3.12.2.

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore	SBOM
Gzipped source tarball	Source release		4e64a004f8ad9af1a75607cf0d5a8c8	25.9 MB	SIG	.sigstore	SPDX
XZ compressed source tarball	Source release		e7c178b97bf8f7cccd677b94d614f7b3c	19.6 MB	SIG	.sigstore	SPDX
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	f88981146d943b5517140fa96e96f153	43.5 MB	SIG	.sigstore	
Windows installer (64-bit)	Windows	Recommended	44abfae489d87cc005d50a9267b5d58d	25.4 MB	SIG	.sigstore	
Windows installer (ARM64)	Windows	Experimental	f769b05cd9d336d2d6e3f6399cb573be	24.7 MB	SIG	.sigstore	
Windows embeddable package (64-bit)	Windows		ded837d78a1efa7ea47b31c14c756faa	10.6 MB	SIG	.sigstore	
Windows embeddable package (32-bit)	Windows		787d286b66a3594e697134ca3b97d7fe	9.4 MB	SIG	.sigstore	
Windows embeddable package (ARM64)	Windows		1ffc0d4ea3f02a1b4dc2a6e74f75226d	9.8 MB	SIG	.sigstore	
Windows installer (32-bit)	Windows		bc4d721cf44a52fa9e19c1209d45e8c3	24.1 MB	SIG	.sigstore	

Figura 13: Installer Python 3.12.2 per Windows 64-bit

**Requests-toolbelt** È una libreria esterna di Python progettata per lavorare con richieste HTTP, in particolare sulla gestione dei dati multipart/form-data. Questo tipo di codifica è comunemente usata per inviare file tramite richieste HTTP POST. Nel contesto di questo progetto, la libreria viene utilizzata nel processo di upload dell'APK da analizzare, il quale viene codificato e inviato al server MobSF tramite le REST API. Per installarlo è necessario eseguire il seguente comando:

```
$ pip install requests-toolbelt
```

**PyQt** Necessario per sviluppare il front-end del progetto. È possibile installarlo usando il pip manager, ovvero il gestore dei pacchetti di Python che semplifica l'installazione di librerie esterne:

```
$ pip install PyQt5
```

Per quanto riguarda l'integrazione dei grafici nel progetto, sarà necessario installare il componente aggiuntivo di PyQt dedicato ai grafici. Questo componente consente di presentare in modo più chiaro e riassuntivo lo stato di sicurezza del dispositivo agli utenti. Per installarlo è possibile eseguire il seguente comando:

```
$ pip install pyqtchart
```

**adbutils** Adbutils è una libreria open-source sviluppata da OpenATX che offre un’interfaccia per comunicare con dispositivi Android tramite ADB (Android Debug Bridge) usando Python.

Le sue funzionalità includono l’estrazione dei file APK delle applicazioni installate, l’invio di eventi di input come toccare lo schermo, l’installazione e la disinstallazione di applicazioni, insieme a tutte le altre operazioni che è possibile utilizzare con ADB.

Questa libreria semplifica l’interazione con i dispositivi Android, facilitando l’automazione di varie operazioni. Per installarla è possibile usare il pip manager di Python, lanciando il seguente comando:

```
$ pip install adbutils
```

**PyInstaller** Necessario per convertire il codice Python in un’eseguibile autonomo per la distribuzione. Può essere installato utilizzando il gestore dei pacchetti di Python:

```
$ pip install pyinstaller
```

Una volta installata, questa libreria viene utilizzata per convertire gli script del progetto in un’unico eseguibile, facilitando l’installazione e l’esecuzione per gli utenti, soprattutto quelli meno esperti. Per fare ciò si utilizza il seguente comando:

```
$ pyinstaller --onefile --noconsole main.py
```

- `--onefile`: Questo parametro indica a PyInstaller di generare un singolo file eseguibile anziché una serie di file.
- `--noconsole`: Questa opzione avvia l’applicazione senza aprire una finestra di console. È stata inclusa poiché il progetto è basato su un’interfaccia grafica (GUI) e non richiede interazione con la console.
- `main.py`: Questo è il nome del file sorgente Python principale. PyInstaller utilizzerà questo file come punto di ingresso per creare l’eseguibile.

**Genymotion** È un emulatore in grado di virtualizzare dispositivi Android. Tramite Genymotion è possibile emulare un dispositivo fra tutti quelli presenti nel mercato Android con la versione dell’OS di Google a scelta. Per questo progetto si utilizzerà la versione desktop

di Genymotion che ha una licenza free per uso personale. Avendo scelto come piattaforma Windows, è possibile installarlo scaricando direttamente l'installer presente sul sito ufficiale. Nello specifico, si andrà a scegliere l'ultima versione disponibile, che attualmente è la 3.6.

Di seguito vengono illustrati i passi che hanno portato alla creazione dell'ambiente virtuale Android.

All'avvio, l'applicazione mostrerà la finestra di scelta dei dispositivi installati. Per creare una nuova configurazione virtuale, basterà selezionare l'icona con il simbolo “+” situata nell'angolo in alto a sinistra.



Figura 14: Pagina principale di Genymotion con pulsante creazione configurazione

Genymotion mostra una schermata iniziale che elenca una serie di configurazioni predefinite basate sulle specifiche hardware di alcuni modelli di dispositivi commerciali. Questo software però, offre anche la possibilità di creare un nuovo dispositivo personalizzato, con specifiche hardware definite dall'utente. Per questo progetto si utilizzerà la configurazione associata al “Google Nexus 4”.

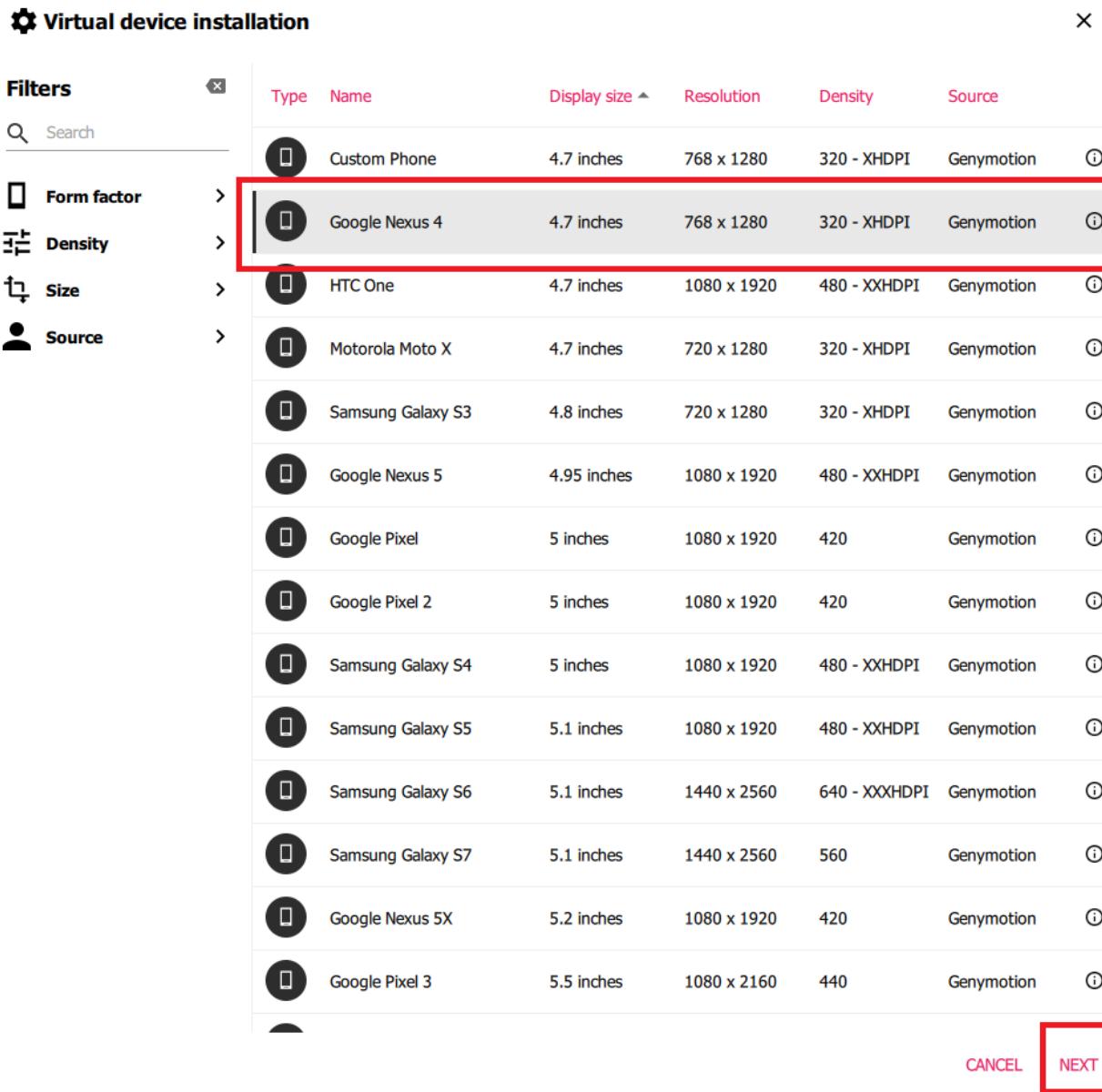


Figura 15: Selezione configurazione dispositivo

Successivamente, il processo consente di personalizzare il nome della configurazione e di selezionare la versione del sistema operativo da installare nell'ambiente virtuale. Per questo progetto, si mantiene il nome predefinito e si seleziona la versione 13.0 di Android.

Successivamente, si ha la possibilità di personalizzare alcuni parametri che definiscono il

dispositivo. Per questo progetto, non è necessario apportare modifiche ai valori predefiniti, pertanto si procede lasciando tutti i valori di default.

Nell'ultima schermata, si configura il valore “Bridge” all'interno della proprietà “Network mode” per consentire l'accesso al dispositivo da altre macchine nella rete. Una volta completata la configurazione, si procede con l'installazione dell'ambiente virtuale.

Una volta terminata l'installazione, sarà possibile avviare la macchina virtuale cliccando sul pulsante di avvio corrispondente alla macchina.

## 4.2 Architettura del progetto

```
/  
|   main.py  
|   utils.py  
|   errors.py  
|   backend/  
|       device_discovery.py  
|       apk_extractor.py  
|       apk_analyzer.py  
|       analysis_result.py  
|       mobsf_api_handler.py  
|   frontend/  
|      mainwindow.py  
|       device_discovery_widget.py  
|       device_info_widget.py  
|       analysis_progress_widget.py  
|       package_analysis_details_widget.py  
|       results_dashboard_widget.py  
|       custom_elements/  
|           stacked_bar_chart.py  
|           vulnerabilities_analysis_table_list.py
```

La struttura del progetto è organizzata in modo gerarchico, con una cartella principale che contiene le sottocartelle backend e frontend, le quali separano le responsabilità tra le diverse parti dell'applicazione.

Il progetto è stato sviluppato seguendo un approccio modulare, utilizzando file separati per le diverse funzionalità presenti. La suddivisione dei widget dell'interfaccia utente in file separati, contribuisce infatti a semplificare la gestione dell'interfaccia stessa. Inoltre la cartella

`custom_elements` contiene elementi personalizzati per la GUI, così da adattare meglio gli elementi e renderli più chiari all'utente.

I due file principali dell'applicazione sono ‘main.py’ e ‘mainwindow.py’, i quali collaborano per gestire l’interfaccia utente e le diverse operazioni di analisi degli APK.

‘main.py’ è il punto di ingresso dell'applicazione. Questo coordina le varie funzionalità dell'applicazione, quali la scoperta del dispositivo, l'estrazione e l'analisi degli APK.

Una funzionalità importante di questo file è la gestione dei segnali. Sfruttando il meccanismo di segnali e slot di PyQt, ‘main.py’ coordina la comunicazione tra le diverse parti dell'applicazione. Ad esempio, si collega ai segnali emessi dalle classi backend, come `DeviceDiscovery`, `ApkExtractor` e `ApkAnalyzer`, e li associa ai relativi slot. Questa implementazione permette al frontend di ricevere aggiornamenti in tempo reale sul progresso dell'analisi e dell'estrazione degli APK.

‘mainwindow.py’ invece rappresenta la finestra principale dell'applicazione. Questa classe gestisce la visualizzazione dei widget e la gestione degli eventi, compresa la logica per avviare e interrompere l'analisi. Inoltre, gestisce l'aggiornamento dinamico dell'interfaccia utente, fornendo dei feedback in tempo reale sull'avanzamento delle operazioni.

### 4.3 Ricerca del dispositivo collegato

Una volta completata l'installazione degli strumenti necessari, il primo passo consiste nell'individuare il dispositivo su cui si andrà ad estrarre e poi analizzare le varie applicazioni installate.

Per questo compito è stata sviluppata la classe `DeviceDiscovery`. La sua funzione principale è quella di individuare il dispositivo disponibile, utilizzando l'interfaccia esposta da `adbutils` che si appoggia sui servizi di ADB.

Quando il thread di esecuzione della classe `DeviceDiscovery` viene avviato, inizia la ricerca di un dispositivo. Inizialmente, cerca di stabilire una connessione con l'emulatore, specificando un limite di tempo massimo entro il quale attendere una risposta durante il tentativo di connessione. Se il tentativo di connessione con l'emulatore non ha successo entro il tempo specificato, viene effettuato un tentativo di connessione con un dispositivo fisico, se presente.

Il risultato di queste operazioni viene memorizzato nell'attributo dell'istanza di classe chia-

mato *device*, che rappresenta il dispositivo individuato e servirà in seguito per restituire le informazioni sul dispositivo (come il brand, il modello ed altro). Il successo o il fallimento dell'operazione di connessione invece viene notificato al thread principale utilizzando i segnali di PyQt, con un valore booleano che indica se la connessione è riuscita o meno.

```
class DeviceDiscovery(QThread):
    signal = pyqtSignal(bool)

    def run(self):
        self.device = None
        try:
            adb.connect("192.168.89.101:5555", timeout = 3.0)
            self.device = adb.device()
            if self.device != None:
                self.signal.emit(True)
                return
        except AdbTimeout:
            pass

        try:
            self.device = adb.device()
            if self.device != None:
                self.signal.emit(True)
                return
        except Exception:
            self.signal.emit(False)
```

Al termine della ricerca, se il dispositivo è stato individuato con successo, verranno mostrate all'utente le informazioni principali sul dispositivo, quali il brand, il modello, e altro ancora.

A questo punto, l'utente potrà avviare l'intero processo di analisi premendo sul relativo pulsante di avvio dell'analisi.

## 4.4 Estrazione degli APK

Il processo di analisi inizia con l'estrazione degli APK dal dispositivo connesso. Per questa fase è stata sviluppata la classe `ApkExtractor`, che si compone di 3 attività principali:

1. Ottenere l'elenco dei pacchetti delle applicazioni installate
2. Ottenere il percorso del file APK dell'applicazione specificata
3. Estrarre il file APK dell'applicazione dal dispositivo

### Ottenere l'elenco dei pacchetti delle applicazioni installate

Questa operazione è gestita dal metodo `get_installed_packages_list()`:

```
def get_installed_packages_list(self):  
    if not self.device:  
        return None  
  
    packages_list = self.device.shell('pm list packages -3 | tr -d "\r"  
    | sed "s/package://g"')  
  
    if not packages_list:  
        return None  
  
    return packages_list.strip().splitlines()
```

Utilizzando la shell di adb tramite adbutils, viene eseguito il comando `pm list packages -3`:

- pm: È l'abbreviazione di “Package Manager”, il gestore dei pacchetti di Android. Questo permette di eseguire operazioni sulle applicazioni installate sul dispositivo, come l'installazione, la disinstallazione e la gestione dei pacchetti.
- list packages: Questa parte del comando indica che si vuole ottenere l'elenco dei pacchetti delle applicazioni installate sul dispositivo.
- -3: Questa opzione specifica che si vogliono elencare solo i pacchetti delle applicazioni di terze parti.

- Infine, l'output viene elaborato per rimuovere i caratteri di ritorno a capo e il prefisso “package:”.

Di seguito un esempio di output di questo metodo:

```
$ com.insecureshop  
$ cm.aptoide.pt
```

## Ottenere il percorso del file APK dell'applicazione specificata

Questo compito è gestito dal metodo `get_installed_package_path(package)`:

```
def get_installed_package_path( self , package ) :
    if not self . device :
        return None

    package_file_path = self . device . shell( f 'pm path { package } | grep "base.apk" | tr -d "\r" | sed "s/package://g"' )
    if package_file_path :
        return package_file_path . strip () . split (":") [-1] . strip ()

    return None
```

Anche in questo caso viene utilizzata la shell, eseguendo il comando `pm path package`:

- pm: Come già accennato, è l'abbreviazione di “Package Manager”, il gestore dei pacchetti di Android.
- path: Questa parte del comando specifica che si vuole ottenere il percorso del file associato al pacchetto di un'applicazione installata.
- Anche qui, l'output viene elaborato per rimuovere i caratteri di ritorno a capo e il prefisso “package:”.

Di seguito un esempio di output di questo metodo:

```
$ /data/app/~~M66N256QFJy3StxVDoPhaQ==/com.insecureshop-4
RXuNpT5yFZAkvn5BgbLlQ==/base.apk
$ /data/app/~~_eDRwAxJDPmSqdSPzu-JHQ==/cm.aptoide.pt-d0cSSayelDBsi
-9X-N3ZkA==/base.apk
```

## Estrarre il file APK dell'applicazione dal dispositivo

Questa fase è gestita dal metodo `pull_package(package)`:

```
def pull_package(self, package):
    if not self.device:
        return None

    if self.stopped:
        return

    package = package.strip()

    if package:
        package_file_path = self.get_installed_package_path(package)
        if package_file_path:
            self.device.sync.pull(package_file_path, utils.
                get_package_output_path(f'{package}.apk'))
```

Questo metodo è responsabile dell'estrazione effettiva del file APK.

Utilizzando `adb.sync.pull` di `adbutils`, il file APK viene trasferito in modo sincrono dal dispositivo Android al computer host. In questo processo quindi, la funzionalità offerta da `adbutils` rimarrà in attesa fino a quando il trasferimento non sarà completato con successo o terminato con un errore.

Durante tutto il processo, la classe `ApkExtractor` comunicherà con il thread principale, notificando qual è il pacchetto attualmente in fase di estrazione e l'avanzamento del processo di estrazione degli APK. Questa comunicazione avviene attraverso l'utilizzo dei segnali di `PyQt`, inviando un valore intero rappresentante la percentuale di completamento dell'estrazione.

## 4.5 Scansione degli APK

Dopo aver estratto gli APK, trasferendoli dal dispositivo Android all'host e posizionandoli nella cartella specificata nel file `utils.py`, viene avviata la fase di scansione di tali APK. Per questa fase è stata sviluppata la classe `ApkAnalyzer`, che funge da intermediario tra

il sistema e MobSF, gestendo il processo attraverso la classe `MobSFAPIMHandler`. La classe `ApkAnalyzer` si suddivide in due attività principali:

1. Upload
2. Scansione

## Upload

L'upload del file APK prima della scansione è essenziale per consentire a MobSF l'accesso al codice sorgente, in modo tale da identificare le potenziali vulnerabilità.

Questa fase è rappresentata dal metodo `upload_file()`, il quale richiama il metodo `upload` della classe `MobSFAPIMHandler` precedentemente citata:

```
class ApkAnalyzer(QThread):  
    # ...  
    def upload_file(self, file_path):  
        upload = self.mobsf_instance.upload(file_path)  
        return upload  
    # ...  
  
class MobSFAPIMHandler:  
    # ...  
    def upload(self, file):  
        multipart_data = MultipartEncoder(fields={"file": (file  
            , open(file, "rb"), "application/octet-stream"))}  
        headers = {  
            "Content-Type": multipart_data.content_type,  
            "Authorization": self.__apikey,  
        }  
  
        r = requests.post(f"{self.__server}/api/v1/upload",  
            data=multipart_data, headers=headers)  
        return r.json()  
    # ...
```

Nell'implementazione, si fa uso delle REST API messe a disposizione da MobSF, inviando una richiesta POST al server dichiarando gli header necessari per la richiesta e i dati multipart per includere il file come parte di tale richiesta HTTP. Infine, il metodo interpreta la risposta dal server, fornita in formato JSON, e la restituisce.

## Scansione

Se l'upload è andato a buon fine, si procede con la scansione vera e propria dell'APK.

Questa fase è gestita dal metodo `scan_file()` della classe `ApkAnalyzer`:

```
class ApkAnalyzer(QThread):
    # ...
    def scan_file(self, file_path, upload_hash):
        scan = self.mobsf_instance.scan(file_path, upload_hash)
        return scan
    # ...

class MobSFAPIHandler:
    # ...
    def scan(self, filename, scanhash):
        post_dict = {
            "scan_type": "apk",
            "file_name": filename,
            "hash": scanhash,
            "re_scan": False,
        }

        headers = {"Authorization": self.__apikey}

        r = requests.post(f"{self.__server}/api/v1/scan", data=
            post_dict, headers=headers)
        return r.json()
    # ...
```

Qui si fa uso dell'hash del file, ottenuto dall'output della richiesta di upload. Questo hash funge da riferimento univoco per il file sul server. Anche in questo caso, si fa uso delle REST

API messe a disposizione da MobSF, inviando una richiesta POST all'endpoint `/api/v1/scan` del server, utilizzando gli header di autorizzazione con la chiave API. Una volta ricevuta la richiesta, il server MobSF la elabora e avvia la scansione dell'APK.

Così come per la fase precedente, durante tutto il processo la classe `ApkAnalyzer` comunicherà con il thread principale, notificando qual è il pacchetto attualmente in fase di scansione e l'avanzamento del processo di analisi degli APK. Questa comunicazione avviene attraverso l'utilizzo dei segnali di PyQt, inviando un valore intero rappresentante la percentuale di completamento.

## 4.6 Elaborazione dei risultati della scansione

Una volta completata la scansione di un singolo APK, vengono estratti e memorizzati i dati relativi a quella determinata scansione.

La classe `AnalysisResult` è stata progettata appositamente per questo scopo, offrendo diverse funzionalità per l'estrazione di informazioni rilevanti, calcolare punteggi e restituire i risultati.

Una volta terminata la scansione, è infatti possibile richiedere il report JSON utilizzando le REST API e accedendo all'endpoint `api/v1/report_json`, come fatto in precedenza con altre richieste fatte al server MobSF. Questo file JSON contiene una vasta gamma di informazioni riguardanti l'APK scansionato, come il nome dell'applicazione, le sue componenti, le vulnerabilità rilevate, i permessi e altro ancora. Questa risorsa è molto utile, poiché permette di accedere a tali dati in modo programmatico per analizzarli, gestirli o visualizzarli.

In questo progetto, ci si concentra sull'estrazione dei dati relativi alle vulnerabilità individuate. Per ciascuna vulnerabilità, vengono memorizzati diversi metadati: la sua inclusione o meno nella “OWASP Mobile Top 10”, il relativo livello di conformità con l'OWASP Mobile Application Security Verification Standard (MASVS), l'URL correlato che reindirizza direttamente al repository OWASP per ulteriori informazioni e infine il livello di gravità della vulnerabilità. Tali metadati vengono estratti utilizzando la funzione `extract_metadata_result()`, che analizza il report JSON per recuperare queste informazioni specifiche:

```

1  {
2      "version": "v3.9.4 Beta",
3      "title": "Static Analysis",
4      "file name": "cm.aptoide.pt.apk",
5      "app name": "Aptoide",
6      "app type": "apk",
7      "size": "18.78MB",
8      "md5": "af0400e229e9df6915d7b467ee267d82",
9      "sha1": "08ac76b5eb360073eaa4e86b7493b408a231a765",
10     "sha256": "d7fd87c404fb518932bbd7834ad19c1a3da9c7ce7b8abc00fa840c0ac28a6f01",
11     "package name": "cm.aptoide.pt",
12     "main activity": "cm.aptoide.pt.view.MainActivity",
13     "exported activities": "[ 'com.facebook.CustomTabActivity' , 'cm.aptoide.pt.DeepLinkIntentReceiver' ]",
14     "Browsable activities": [
15         "activities": [
16             "receivers": [
17                 "providers": [
18                     "services": [
19                         "libraries": [],
20                         "target sdk": "25",
21                         "max sdk": "",
22                         "min sdk": "16",
23                         "version name": "9.20.6.1",
24                         "version code": "12010",
25                         "icon path": "af0400e229e9df6915d7b467ee267d82-icon.png",
26                         "permissions": [
27                             "malware permissions": [
28                                 "certificate analysis": [
29                                     "manifest analysis": [
30                                         "network security": [
31                                             "binary analysis": [],
32                                             "file analysis": [
33                                                 "android api": [
34                                                     "code analysis": [
35                                                         "findings": [
36                                                             "android logging": [
37                                                                 "files": [
38                         "metadata": [
39                             "cvss": 7.5,
40                             "cwe": "CWE-532: Insertion of Sensitive Information into Log File",
41                             "owasp-mobile": "",
42                             "masvs": "MSTG-STORAGE-3",
43                             "ref": "https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#logs",
44                             "description": "The App logs information. Sensitive information should never be logged.",
45                             "severity": "info"
46                         }
47                     ],
48                     "android sql raw query": [
49                         "android hardcoded": [
50                             "android ssl pinning": [
51                                 "android read write external": []
52

```

Figura 16: Esempio report JSON per il file cm.aptoide.pt.apk

```

def extract_metadata_result(self , json_report):
    metadata_dict = {}
    code_analysis = json_report.get( 'code_analysis' , {} )
    findings = code_analysis.get( 'findings' , {} )

    for key , value in findings.items():
        if 'metadata' in value:
            metadata = value[ 'metadata' ]
            metadata_dict[key] = {
                'owasp-mobile': metadata.get( 'owasp-mobile' ),
                'masvs': metadata.get( 'masvs' ),
                'ref': metadata.get( 'ref' ),

```

```
        'severity': metadata.get('severity')  
    }  
  
    return metadata_dict
```

Oltre a ciò, la classe `AnalysisResult` tiene traccia anche del punteggio di sicurezza attribuito a ciascun APK. Questo punteggio rappresenta una valutazione della sicurezza dell'applicazione mobile, basata su una serie di criteri come la presenza di vulnerabilità note, l'utilizzo di librerie sicure, l'appropriatezza delle autorizzazioni richieste e altri fattori. Questo punteggio viene assegnato in base a un insieme di criteri stabiliti dall'analisi automatizzata condotta da MobSF. Un punteggio più alto indica un'applicazione più sicura, mentre un punteggio più basso suggerisce un maggiore rischio per la sicurezza.

Tenendo traccia di tutti i punteggi di sicurezza, sarà possibile ottenere, nella fase successiva di visualizzazione, il punteggio di sicurezza globale. Questo valore fornisce agli utenti un'indicazione generale dello stato di sicurezza del proprio dispositivo. Questo viene determinato calcolandone la media dei singoli punteggi di sicurezza delle app installate.

## 4.7 Visualizzazione dei risultati

Una volta completata l'analisi di tutti gli APK, viene mostrata all'utente una finestra che rappresenta l'analisi complessiva. I dati sono ottenuti elaborando i risultati della scansione, attraverso la classe `AnalysisResult`. Utilizzando queste informazioni, l'utente può accedere a 3 pagine selezionabili tramite un menu:

**Dashboard** Questa pagina fornisce una panoramica rapida dell'analisi, includendo il punteggio di sicurezza globale e il numero di vulnerabilità individuate suddivise per categoria (alta, media, info e sicura) per ogni applicazione analizzata. I dati della dashboard sono visualizzati tramite grafici che offrono una rappresentazione visiva dell'overview, rendendo più facile comprendere le informazioni. Il punteggio di sicurezza globale è visualizzato tramite un grafico ad anello, dove ogni valore è associato a un colore specifico: rosso per punteggi inferiori a 15, rosso chiaro per punteggi inferiori a 40, arancione per punteggi inferiori a 70, e verde per punteggi pari o superiori a 70. Questa categorizzazione consente agli utenti di comprendere in modo più chiaro lo stato di sicurezza del dispositivo.

**Vulnerabilità** In questa pagina è elencata la lista di tutte le vulnerabilità individuate nelle applicazioni. Ogni vulnerabilità è descritta con un titolo e specifica se è inclusa nella OWASP Mobile Top, l'OWASP MASVS relativo e fornisce un pulsante per il reindirizzamento al repository ufficiale dello standard di verifica della sicurezza delle applicazioni mobili.

**Dettagli** Questa pagina presenta una tabella che dettaglia il punteggio di sicurezza per ogni applicazione. Inoltre, offre un pulsante per l'apertura diretta del report PDF generato automaticamente da MobSF. Questo report fornisce una visione dettagliata delle informazioni su quella specifica applicazione, delle sue componenti, delle vulnerabilità rilevate, dei permessi e altro ancora.

## 4.8 Risultati

Di seguito viene documentato in dettaglio lo strumento sviluppato in questo progetto di tesi, applicandolo in un caso reale su un dispositivo HUAWEI POT-LX1T, appartenente a un utente reale.

Poiché si trattava di un dispositivo fisico e non di un emulatore, è stato necessario abilitare, direttamente sul dispositivo mobile, la modalità Debug USB. Questo è stato fatto attraverso il seguente procedimento: l'utente ha dovuto accedere alle impostazioni del telefono, precisamente alla sezione denominata "Info telefono". Una volta lì, è stato necessario toccare ripetutamente la voce corrispondente a "Numero di build" fino a che non è comparso un messaggio chiave: "Ora sei uno sviluppatore".

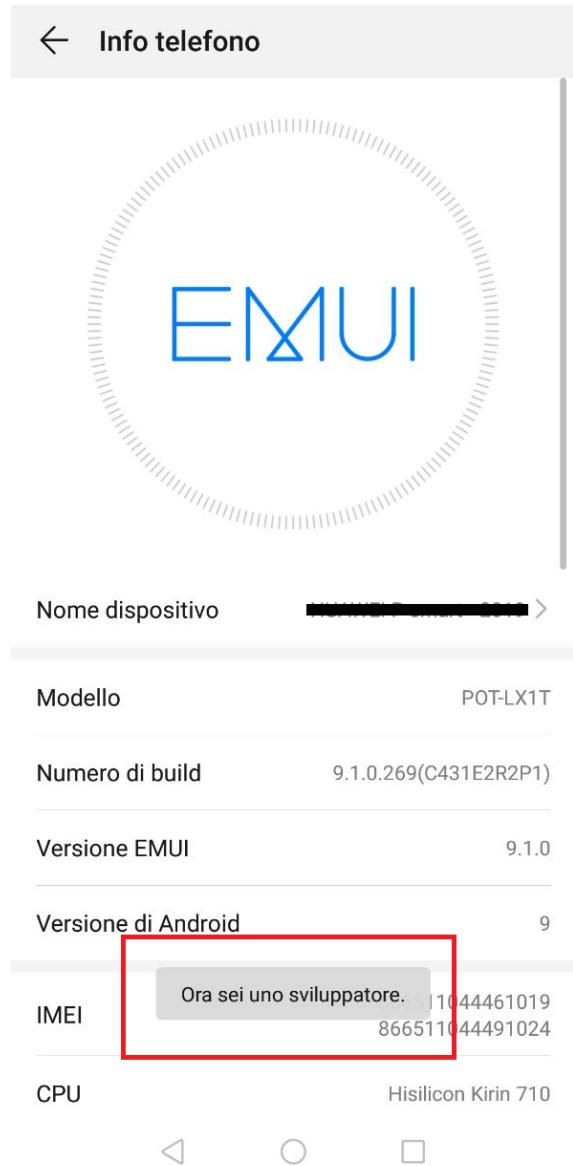


Figura 17: Attivazione modalità sviluppatore

Successivamente, si accede alle “Opzioni sviluppatore” dove si potrà abilitare il Debug USB.

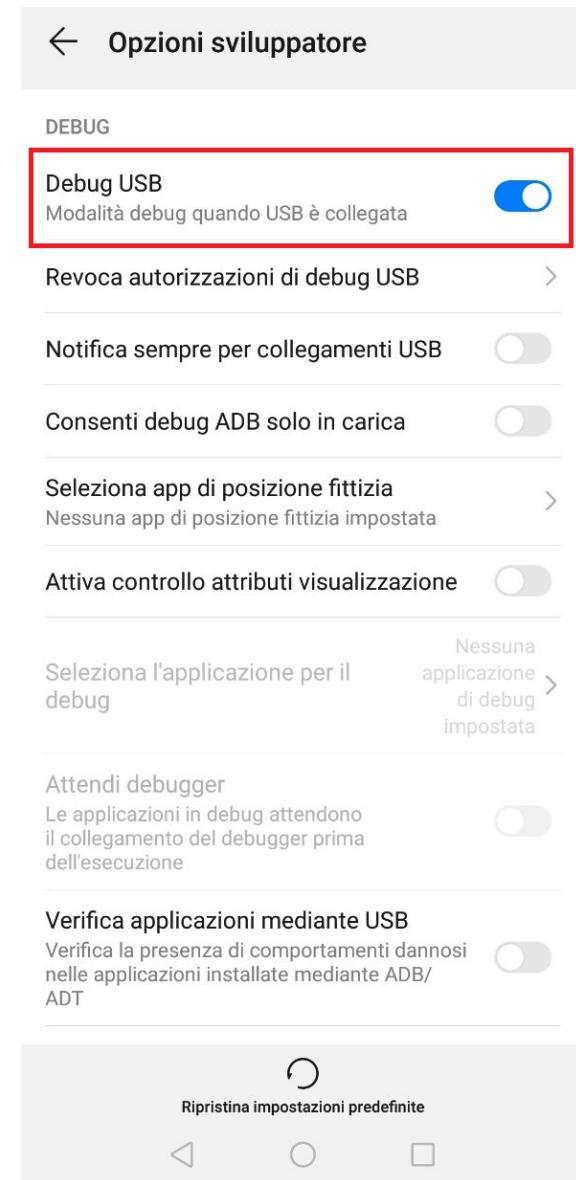


Figura 18: Abilitazione Debug USB

A questo punto si procederà con il collegare il dispositivo alla macchina tramite USB, per poi avviare l'eseguibile dell'applicazione:

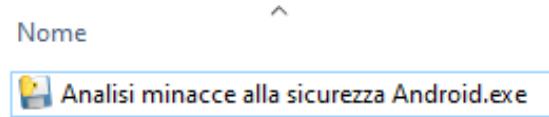


Figura 19: File eseguibile dell'applicazione

Come illustrato nei capitoli precedenti, non sarà necessario che l'utente scarichi alcuna libreria o strumento esterno. Grazie alla libreria pyinstaller è tutto incluso in un unico file eseguibile. Una volta eseguito quest'ultimo, inizierà la prima fase dell'applicazione: la ricerca del dispositivo:

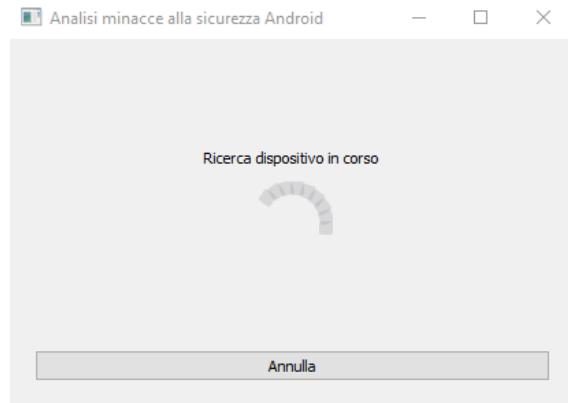


Figura 20: Ricerca del dispositivo

La prima volta che si connette il dispositivo con un nuovo computer, sarà richiesta l'autorizzazione per consentire la comunicazione con il PC, come mostrato nella figura 21.



Figura 21: Autorizzazione per eseguire il Debug USB

Dopo aver ottenuto l'autorizzazione, l'applicazione riuscirà ad individuare il dispositivo collegato visualizzando la schermata principale contenente alcune informazioni sul dispositivo e il pulsante di avvio dell'analisi.

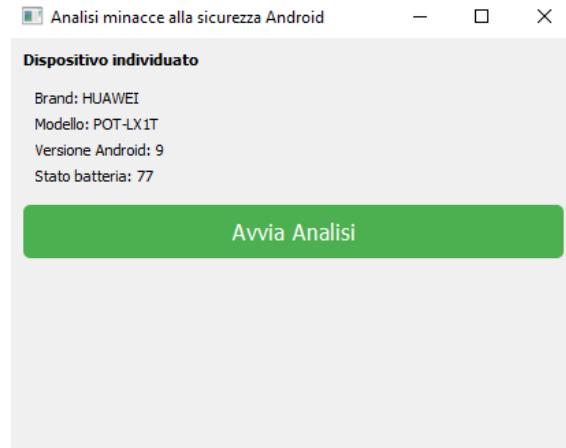


Figura 22: Dispositivo individuato

Premendo il pulsante di avvio, partirà l'intero processo. Quindi l'applicazione prima procederà nell'estrare le applicazioni installate (e quindi i file APK) e successivamente li analizzerà:

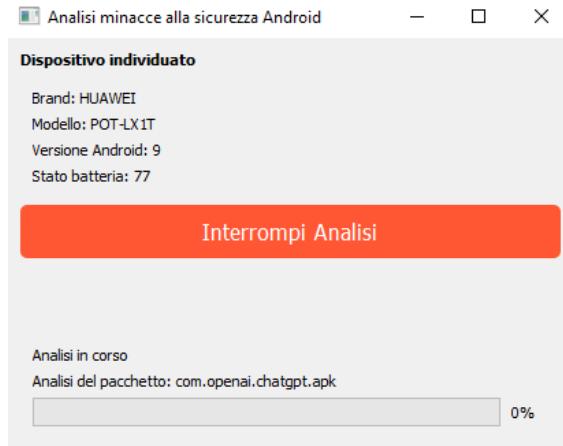


Figura 23: Analisi del pacchetto “com.openai.chatgpt.apk”

Al termine dell’analisi, vengono mostrati all’utente i relativi risultati. Questi sono stati presentati attraverso diverse schermate.

Nell’overview, come accennato precedentemente, abbiamo un primo grafico a torta che indica il punteggio di sicurezza globale del dispositivo. Un punteggio più alto indica un’applicazione più sicura, mentre un punteggio più basso suggerisce un maggiore rischio per la sicurezza.

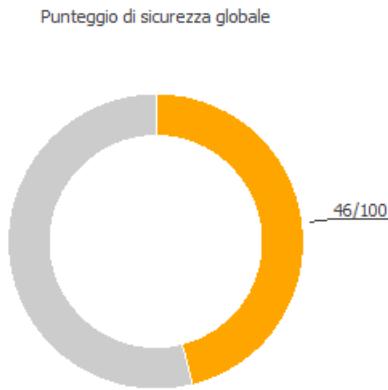


Figura 24: Punteggio di sicurezza globale

Nella seconda parte della schermata di overview abbiamo invece un grafico stacked, che mostra il numero di vulnerabilità individuate per ogni applicazione, suddivise per criticità.

In questo grafico sono presenti due tipi di visualizzazione. Il primo basato sull'ordinamento per numero di vulnerabilità, esprime quali sono le applicazioni con più problemi mostrato nella figura 4.8, mentre il secondo tipo di visualizzazione, mostrata nella figura 26, si basa su un ordinamento per criticità, ponendo in primo piano quelle di maggiore gravità e in maggior numero, ovvero quelle con criticità high.

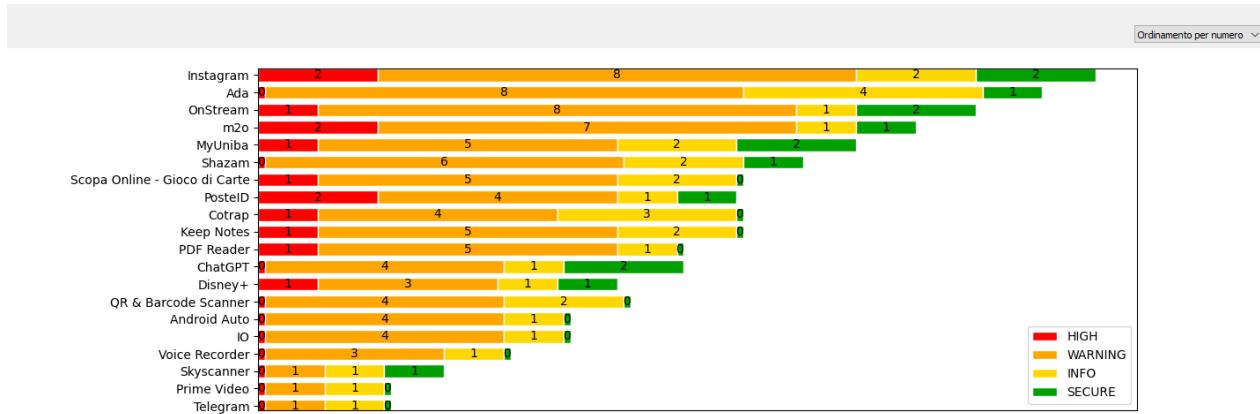


Figura 25: Numero di vulnerabilità individuate ordinate per quantità

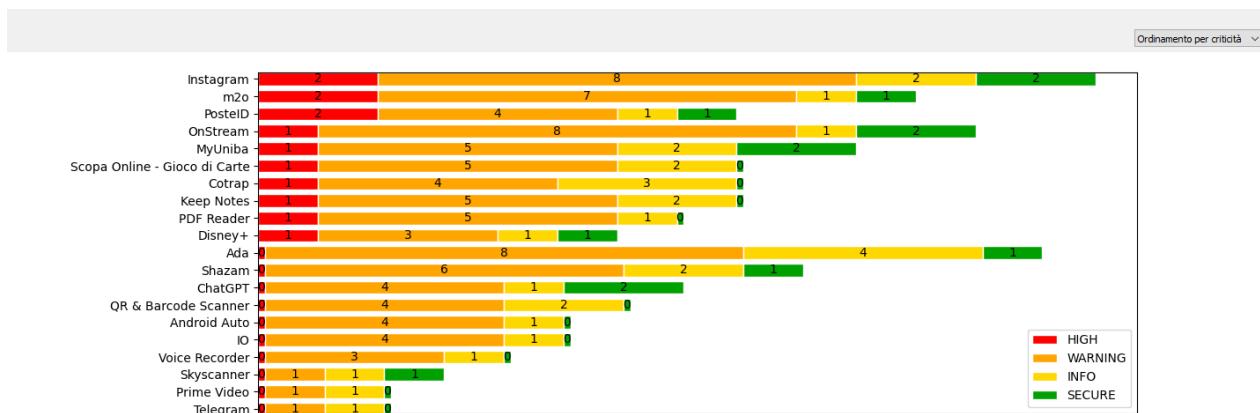


Figura 26: Numero di vulnerabilità individuate ordinate per criticità

In questo caso, il punteggio di sicurezza globale è 46/100. Il grafico stacked mostra come l'applicazione con più vulnerabilità individuate è Instagram, con un maggior numero di criticità

“Warning 4” e solamente 2 di tipo “High”

L'altra schermata che è possibile visualizzare è quella relativa alla lista delle vulnerabilità. Qui vengono elencate in maniera più dettagliata tutte le vulnerabilità a cui il dispositivo potrebbe essere a rischio.

Lista vulnerabilità individuate				
	Titolo	OWASP Mobile Top 10	OWASP MASVS	Reindirizzamento
1	Android Logging		MSTG-STORAGE-3	<a href="#">Maggiori info</a>
2	Android Hardcoded	M9: Reverse Engineering	MSTG-STORAGE-14	<a href="#">Maggiori info</a>
3	Android Insecure Random	M5: Insufficient Cryptography	MSTG-CRYPTO-6	<a href="#">Maggiori info</a>
4	Android Sql Raw Query	M7: Client Code Quality		<a href="#">Maggiori info</a>
5	Android Clipboard Copy		MSTG-STORAGE-10	<a href="#">Maggiori info</a>
6	Android Read Write External	M2: Insecure Data Storage	MSTG-STORAGE-2	<a href="#">Maggiori info</a>
7	Android Hiddenui	M1: Improper Platform Usage	MSTG-STORAGE-7	<a href="#">Maggiori info</a>
8	Android Write App Dir		MSTG-STORAGE-14	<a href="#">Maggiori info</a>
9	Android Ssl Pinning		MSTG-NETWORK-4	<a href="#">Maggiori info</a>
10	Android Temp File	M2: Insecure Data Storage	MSTG-STORAGE-2	<a href="#">Maggiori info</a>
11	Android Sha1	M5: Insufficient Cryptography	MSTG-CRYPTO-4	<a href="#">Maggiori info</a>
12	Android Webview Debug	M1: Improper Platform Usage	MSTG-RESILIENCE-2	<a href="#">Maggiori info</a>
13	Android Ip Disclosure		MSTG-CODE-2	<a href="#">Maggiori info</a>
14	Android Detect Root		MSTG-RESILIENCE-1	<a href="#">Maggiori info</a>
15	Android Webview	M1: Improper Platform Usage	MSTG-PLATFORM-7	<a href="#">Maggiori info</a>
16	Android Insecure Ssl	M3: Insecure Communication	MSTG-NETWORK-3	<a href="#">Maggiori info</a>
17	Android Md5	M5: Insufficient Cryptography	MSTG-CRYPTO-4	<a href="#">Maggiori info</a>
18	Android Sql Cipher		MSTG-CRYPTO-1	<a href="#">Maggiori info</a>
19	Android Webview Ignore Ssl	M3: Insecure Communication	MSTG-NETWORK-3	<a href="#">Maggiori info</a>
20	Android Aar Jar Debug Enabled	M1: Improper Platform Usage	MSTG-RESILIENCE-2	<a href="#">Maggiori info</a>
21	Android Aes Ecb	M5: Insufficient Cryptography	MSTG-CRYPTO-2	<a href="#">Maggiori info</a>
22	Android Su Detect		MSTG-RESILIENCE-1	<a href="#">Maggiori info</a>
23	Cbc Padding Oracle	M5: Insufficient Cryptography	MSTG-CRYPTO-3	<a href="#">Maggiori info</a>
24	Android Clipboard Listen		MSTG-PLATFORM-4	<a href="#">Maggiori info</a>
25	Android World Readable	M2: Insecure Data Storage	MSTG-STORAGE-2	<a href="#">Maggiori info</a>

Figura 27: Lista vulnerabilità individuate

Per ogni vulnerabilità viene specificato se è nella OWASP Mobile TOP 10, viene specificato se è nell'OWASP MASVS e infine viene data la possibilità all'utente di poter visualizzare ancora più nel dettaglio la vulnerabilità con il pulsante “Maggiori info”, dove l'utente verrà reindirizzato al riferimento ufficiale OWASP di quella vulnerabilità. Un esempio di reindirizzamento è mostrato nella figura 28.

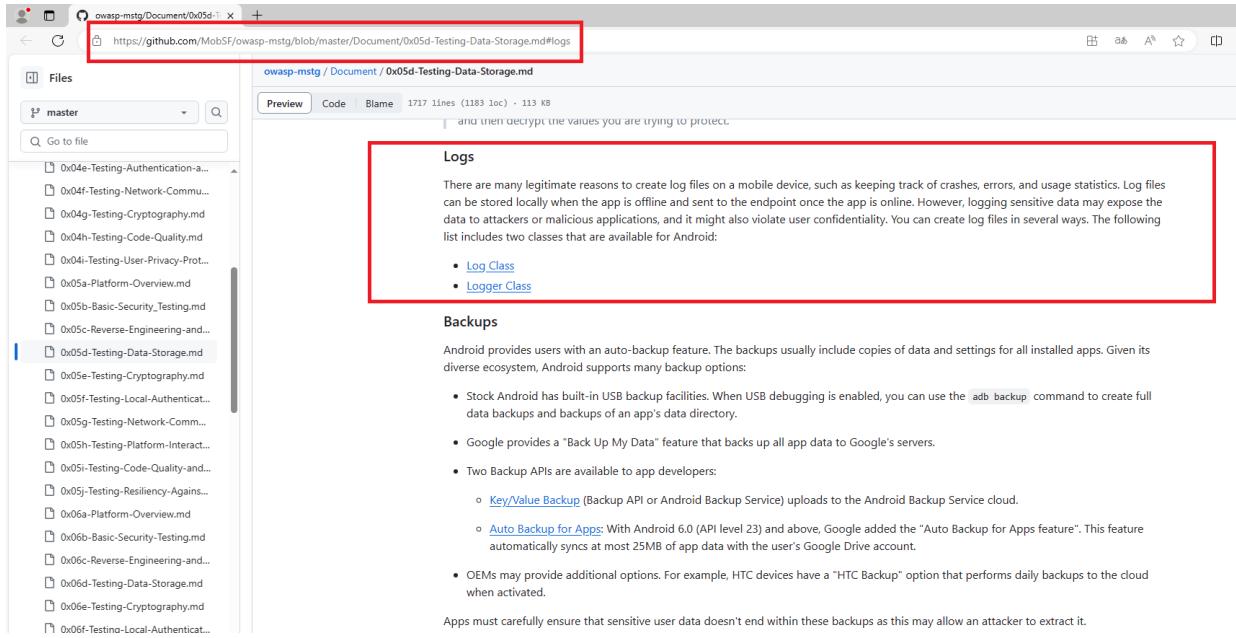


Figura 28: Reindirizzamento al OWASP MASTG per la vulnerabilità “Android Logging”

In questo caso, notiamo che il dispositivo è affetto da 25 vulnerabilità.

Infine è presente la schermata dettagli, la quale mostra per ogni applicazione installata sul dispositivo il relativo punteggio di sicurezza.

Dettagli			
	Applicazione	Punteggio di sicurezza	Dettagli
1	Ada	44/100	<a href="#">Dettagli</a>
2	OnStream	49/100	<a href="#">Dettagli</a>
3	m2o	44/100	<a href="#">Dettagli</a>
4	Shazam	53/100	<a href="#">Dettagli</a>
5	Keep Notes	47/100	<a href="#">Dettagli</a>
6	ChatGPT	55/100	<a href="#">Dettagli</a>
7	Cotrap	46/100	<a href="#">Dettagli</a>
8	MyUniba	49/100	<a href="#">Dettagli</a>
9	PostelD	43/100	<a href="#">Dettagli</a>
10	Voice Recorder	43/100	<a href="#">Dettagli</a>
11	QR & Barcode Scanner	39/100	<a href="#">Dettagli</a>
12	IO	41/100	<a href="#">Dettagli</a>
13	Telegram	45/100	<a href="#">Dettagli</a>
14	Prime Video	57/100	<a href="#">Dettagli</a>
15	Skyscanner	51/100	<a href="#">Dettagli</a>
16	Scopa Online - Gioco di Carte	37/100	<a href="#">Dettagli</a>
17	Android Auto	48/100	<a href="#">Dettagli</a>
18	Instagram	49/100	<a href="#">Dettagli</a>
19	Disney+	45/100	<a href="#">Dettagli</a>
20	PDF Reader	35/100	<a href="#">Dettagli</a>

Figura 29: Dettagli delle applicazioni analizzate

Viene inoltre messo a disposizione per l'utente la possibilità di visualizzare il report completo PDF dell'analisi MobSF fatta su quell'applicazione, che quindi conterrà tutti i dettagli quali ad esempio informazioni generiche sull'applicazione, le sue componenti, i servizi che utilizza, i permessi che necessita, ed altro ancora. Un esempio delle prime pagine di un report PDF è mostrato nella figura 30.

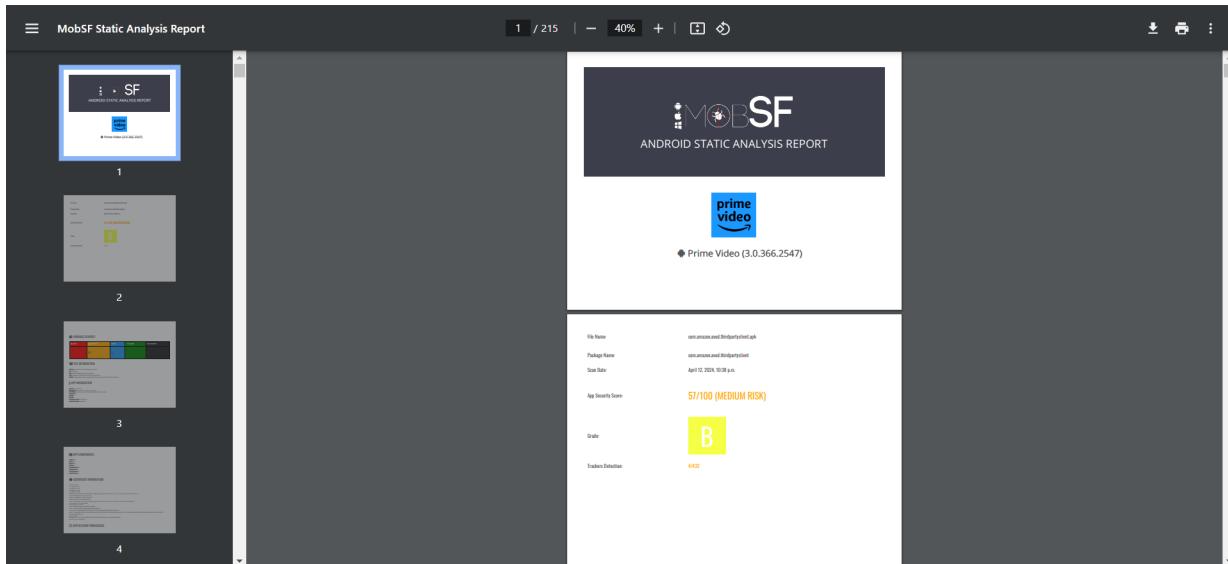


Figura 30: Esempio prime pagine di un report PDF per l'app “Prime Video”

In questo caso notiamo che delle 20 applicazioni installate dall’utente, quella con il punteggio più basso (e quindi quella potenzialmente più vulnerabile) è l’applicazione PDF Reader. Quella invece che ha il punteggio più alto (quindi più sicura) è l’applicazione Prime Video, con un punteggio di 57/100.

Una cosa importante da tenere a mente, soprattutto per gli utenti meno esperti, è che questi strumenti possono generare dei falsi positivi, segnalando erroneamente la presenza di vulnerabilità che in realtà non esistono. Bisogna quindi adottare un approccio critico nell’interpretazione dei risultati ottenuti.

## 4.9 Conclusioni

L'aumento dell'uso degli smartphone e il loro ruolo sempre più centrale nella vita quotidiana hanno evidenziato sempre più l'importanza della sicurezza. Rispetto al passato, dove le informazioni erano limitate a rubrica e messaggi, ora gli smartphone contengono una vasta gamma di dati sensibili, come informazioni bancarie, foto, password e email. Inoltre, l'interesse crescente degli aggressori per i dispositivi mobili e la diffusione di software dannosi, favorita dalla possibilità di installare app da store di terze parti, aumentano il rischio di compromissione dei dispositivi.

Per affrontare queste sfide, gli strumenti di analisi statica si rivelano particolarmente utili, consentendo di identificare potenziali rischi e vulnerabilità nelle applicazioni.

Questo progetto di tesi si è concentrato sulla realizzazione di uno strumento di analisi delle applicazioni Android. A differenza delle soluzioni attualmente disponibili sul mercato, che non offrono metodi di visualizzazione orientati ad utenti meno esperti nell'esplorazione dei risultati da essi prodotti, o non supportano l'analisi simultanea di più applicazioni, il progetto fornisce agli utenti un quadro complessivo dello stato di sicurezza del dispositivo. Questo strumento mira a essere un supporto per gli utenti Android, semplificando il processo di valutazione della sicurezza e permettendo loro di comprendere meglio i rischi e le vulnerabilità delle app installate sui loro dispositivi.

È importante sottolineare però, che questi strumenti possono generare dei falsi positivi, segnalando erroneamente la presenza di vulnerabilità che in realtà non esistono. Questo è un aspetto da tenere in considerazione, sia per gli utenti che per gli sviluppatori. Gli utenti, soprattutto quelli meno esperti, devono adottare un approccio critico nell'interpretazione dei risultati ottenuti, evitando di basarsi esclusivamente su questi strumenti. D'altra parte, è importante sensibilizzare gli sviluppatori ad integrare gli strumenti di analisi statica fin dalle fasi iniziali del progetto, includendo la sicurezza come parte integrante del processo di progettazione. Questo permette di individuare e correggere le vulnerabilità sin dalle prime fasi dello sviluppo, riducendo così il rischio di esposizione a potenziali attacchi e garantendo un livello di sicurezza più elevato per gli utenti finali.

In conclusione, la sicurezza dei dispositivi Android è una sfida in continua evoluzione. Gli aggressori diventano sempre più sofisticati e il panorama delle minacce è in costante cambiamento. È quindi fondamentale per gli utenti e gli sviluppatori prendere misure preventive

per salvaguardare i dispositivi e le informazioni personali dall'evolversi delle minacce nel panorama digitale.

## 4.10 Sviluppi futuri

Questo progetto di tesi ha portato alla realizzazione di un utile strumento a supporto degli utenti Android per poter analizzare diverse applicazioni insieme, semplificando il processo di analisi della sicurezza delle applicazioni Android e rendendolo accessibile anche a coloro che non hanno competenze informatiche. Benché questo sia un buon punto di partenza, possono essere applicate ulteriori migliorie, quali ad esempio:

- Ridurre i tempi di analisi: Valutare eventuali ottimizzazioni per ridurre i tempi di analisi e migliorare l'efficienza complessiva.
- Integrazione con altre piattaforme di analisi: Integrazione con altri strumenti e piattaforme di analisi e sicurezza, consentendo agli utenti di sfruttare un'ampia gamma di strumenti e risorse per valutare la sicurezza delle proprie applicazioni.
- Implementazione Web App: Implementare la visualizzazione dei risultati in una Web App a sé stante, per garantire maggiore flessibilità nella creazione e gestione dei widget grafici.
- Implementazione grafici interattivi: Implementare grafici interattivi che consentano all'utente di esplorare e comprendere ancora meglio le informazioni fornite dall'analisi. Questo permetterà di visualizzare in modo più dettagliato e intuitivo i risultati, migliorando la comprensione senza compromettere la semplicità d'uso dell'applicazione.

# Bibliografia

- [1] Statista. *Number of mobile internet users worldwide from 2020 to 2029*. URL: <https://www.statista.com/forecasts/1146312/mobile-internet-users-in-the-world>. (Ultimo accesso: 11.03.2024).
- [2] Pew Research Center. *Demographics of Mobile Device Ownership and Adoption in the United States*. URL: <https://www.pewresearch.org/internet/fact-sheet/mobile/>. (Ultimo accesso: 11.03.2024).
- [3] MobiLoud. *What Percentage of Internet Traffic is Mobile?* URL: <https://www.mobiloud.com/blog/what-percentage-of-internet-traffic-is-mobile>. (Ultimo accesso: 17.03.2024).
- [4] S. F. Afroze F. H. Shezan e A. Iqbal. «Vulnerability detection in recent android apps: an empirical study». In: *International Conference on Networking, Systems and Security (NSysS)* (2017), pp. 55–63.
- [5] Bank my cell. *Percentage of Android vs iPhone Users Worldwide*. URL: <https://www.bankmycell.com/blog/how-many-android-users-are-there>. (Ultimo accesso: 11.03.2024).
- [6] G. McWilliams S. Y. Yerima S. Sezer e I. Muttik. «A new android malware detection approach using bayesian classification». In: *27th IEEE International Conference on Advanced Information Networking and Applications (AINA)* (2013), pp. 121–128.
- [7] A. K. Singh N. Chiluka e R. Eswarawaka. «Privacy and security issues due to permissions glut in android system». In: *International Conference on Inventive Research in Computing Applications (ICIRCA)* (2018), pp. 406–411.
- [8] B. Mueller. «Mobile application security verification standard (masvs)». In: *OWASP Standard* (2017).

- [9] R. Baldoni e R. De Nicola. «Il futuro della cybersecurity in italia: Ambiti progettuali strategici». In: *CINI-Consorzio Interuniversitario Nazionale* (2018).
- [10] StatCounter. *Mobile Operating System Market Share Worldwide*. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide>. (Ultimo accesso: 11.03.2024).
- [11] Axelle Apvrille e Tim Strazzere. «Reducing the window of opportunity for Android malware Gotta catch 'em all». In: *Journal in Computer Virology* 8 (2012), pp. 61–71. DOI: <https://doi.org/10.1007/s11416-012-0162-3>.
- [12] Ajin Abramo. *MobSF: Mobile sicurezza Framework*. URL: <https://github.com/MobSF/Mobile-Security-Framework-MobSF>. (Ultimo accesso: 20.03.2024).
- [13] Cristiano Kisutsa. *MARA Framework*. URL: <https://github.com/xtiankisutsa/MARA%20Framework>. (Ultimo accesso: 20.03.2024).
- [14] Dr. Ilia Kolochenco. *ImmuniWeb*. URL: <https://www.immuniweb.com>. (Ultimo accesso: 20.03.2024).
- [15] Yu-Cheng Lin. *AndroBugs Framework*. URL: [https://github.com/AndroBugs/AndroBugs\\_Framework](https://github.com/AndroBugs/AndroBugs_Framework). (Ultimo accesso: 20.03.2024).
- [16] Ryan B. Joseph, Minhaz F. Zibran e Farjana Z. Eishita. «Choosing the Weapon: A Comparative Study of Security Analyzers for Android Applications». In: *2021 IEEE/A-CIS 19th International Conference on Software Engineering Research, Management and Applications (SERA)* (2021), pp. 51–57. DOI: <https://doi.org/10.1109/SERA51205.2021.9509271>.
- [17] Wikipedia. *Usage share of operating systems*. URL: [https://en.wikipedia.org/wiki/Usage\\_share\\_of\\_operating\\_systems](https://en.wikipedia.org/wiki/Usage_share_of_operating_systems). (Ultimo accesso: 17.03.2024).