Università degli Studi di Padova
Scuola di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Autenticazione Continua su Smartwatch Tramite Accelerometro e Sensore Cardiaco

Relatore
Prof. Mauro Migliardi

Candidato
Gregorio Morin
N 1234010

Anno Accademico 2021–2022

Sommario

L'obiettivo di questa tesi è quello di studiare la possibilità su Smartwatch di mantenere l'utente autenticato al suo dispositivo prolungando una sessione precedentemente creata da quest ultimo tramite l'utilizzo del sensore cardiaco e dell'accelerometro.
Nella prima parte dell'elaborato, dopo una breve introduzione, vengono trattate tematiche relative alle vulnerabilità degli Smartwatch e successivamente proposte delle possibili soluzioni per rinforzare la sicurezza del dispositivo.
Nella seconda parte, invece, viene spiegato nel dettaglio il funzionamento dell'applicazione sviluppata e, in particolare, come rilevare movimenti sospetti e come accertarsi della presenza dell'utente. Infine, dopo le conclusioni, vengono proposti possibili sviluppi futuri e miglioramenti per l'app.

Indice

Capitolo 1

Introduzione

Gli smartwatch oggigiorno stanno diventando sempre più presenti nella quotidianità delle persone e si stima che il mercato di questi ultimi avrà un valore di oltre 93 miliardi nel 2027. La richiesta per questa tecnologia è aumentata di molto negli anni[**BMI20**] e ciò è dovuto al fatto che nuove funzionalità, sempre più accattivanti, vengono di anno in anno sviluppate. Per citare alcuni esempi oggi col proprio smartwatch è possibile ricevere e fare chiamate, controllare messaggi ed e-mail, effettuare pagamenti on-line. Tutte queste novità, nonostante possano essere molto utili, portano gli smartwatch ad accumulare inevitabilmente una grande quantità di dati personali esponendo l'utente a possibili attacchi malevoli. In un primo momento la sicurezza dello smartwatch era gestita dal dispositivo al quale era accoppiato, che solitamente era uno smartphone, e non erano necessarie altre misure di prevenzione e/o difesa. Tuttavia, negli ultimi periodi, grandi compagnie hanno lavorato per rendere il dispositivo sempre più indipendente, in particolare Google ed Apple. Lo smartwatch è cosi passato da essere un dispositivo secondario ad essere un dispositivo primario capace di funzionare da identity proxy e comunicare con altri dispositivi legati al mondo dell Internet of Things.
I principali strumenti di autenticazione implementati negli smartwatch per proteggere il dispositivo sono pin a 4 cifre o APL (Android Lock Pattern) e sono usati sia come semplice blocco dello schermo sia per azioni più sensibili come eventuali pagamenti. Appare dunque chiaro che ottenere tali sequenza di sblocco non comporterebbe solo la possibilità di utilizzare liberamente il dispositivo ma anche la possibilità di compiere azioni più dannose come ad esempio la sottrazione di denaro. L'intento ultimo di questo elaborato è quello di fornire un'idea per un sistema di protezione contro tali azioni bloccando lo smartwatch in caso di furto.
Nella prima parte di questa tesi verranno mostrate alcune tecniche avanzate di attacco nei confronti degli smartwatch e successivamente alcune tecniche di difesa. Nella seconda parte verrà invece trattato il tema dell'autenticazione continua con conclusioni finali.

Capitolo 2

Vulnerabilità Smartwatch

L'indipendenza raggiunta dallo smartwatch è stata sicuramente un passo avanti per questa tecnologia. Questo ha comportato che una grande quantità di informazioni personali dell'utente vengano raccolte dal dispositivo. I dati disponibili possono essere di vario tipo. Da semplici mail o dati riguardanti l'attività fisica dell'utente, fino a dati per poter pagare con carte prepagate o carte di credito.
La conseguenza naturale di tale disponibilità di informazioni è la possibilità di ricevere attacchi. Nella prima sezione verranno descritte le principali vulnerabilità degli smartwatch, ossia shoulder surfing e side channel attacks, e, successivamente, mostrate tre tecniche di attacco sviluppate da alcune università: la prima riguarderà un attacco di tipo side-channel mentre la seconda uno shoulder surfing. La terza tecnica è particolare e non ricade in queste due categorie.

2.1 Shoulder Surfing e Side-Channel Attacks

I più comuni sistemi di difesa degli smartwatch sono pin a 4 cifre e APL e permettono all'utente di identificarsi sul dispositivo e di accedere alle sue funzionalità. Questi metodi però risultano poco robusti contro alcuni tipi di attacchi. Il primo tipo di attacco discusso è chiamato shoulder surfing ed è autoesplicativo: consiste nello osservare lo user mentre sblocca il dispositivo così da ottenere le credenziali. In questo modo, se il dispositivo venisse sottratto, il ladro avrebbe a disposizione la possibilità di accedere ai dati contenuti in esso e di utilizzarlo liberamente. Nonostante la banalità della tecnica essa è risultata negli anni molto efficace tanto da indurre studi su password in grado di resistere a tale attacco. La soluzione solitamente risulta rendere gli step di log-in lunghi e macchinosi ma efficaci per preservare la sicurezza del dispositivo.[**Wie+06**]
Il secondo tipo di attacco è più sofisticato. Esso è indipendente dal software e non sfrutta una debolezza di quest ultimo. Un side-channel attack[**Nah16**] monitora il modo in cui il sistema operativo del dispositivo accede all'hardware su cui è in esecuzione e ne analizza il comportamento. Parlando di smartwatch nello specifico, è possibile risalire, tramite l'ascolto dei sensori di movimento, a eventuali pin usati per bloccare il dispositivo o scoprire le sequenze degli APL. A differenza degli smartphone, la ridotta dimensione dello schermo rende tale

compito più difficile. Nella prima immagine si può notare come sia il tapping che lo swiping non siano uniformi e rendono difficile riconoscere le singole cifre.

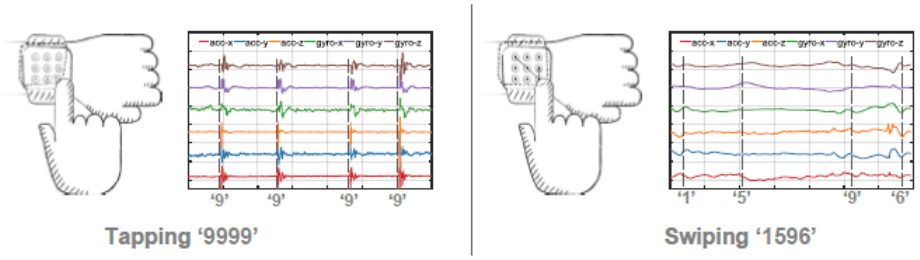


Figura 2.1: Tapping e Swiping su smartwatch

Nella seconda , invece, si analizza l'SNR proveniente da rispettivamente accelerometro a sinistra e giroscopio a destra. Si nota che il rumore è da 20 a 40 dB peggiore rispetto a quello di un comune smartphone.

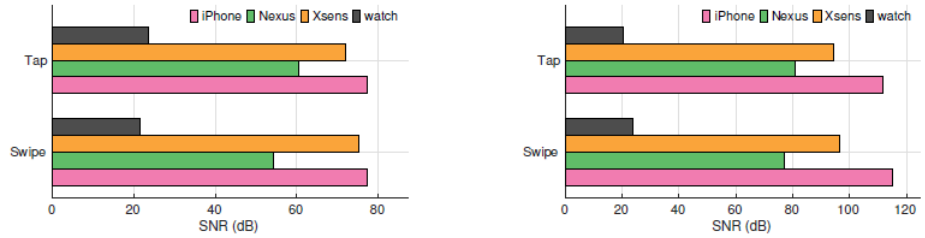


Figura 2.2: SNR Accelerometro e Giroscopio

2.2 Attacco tramite l'ascolto dei sensori

Snoopy[**Lu+18**] è il primo sistema che dimostra la possibilità di intercettare pin immessi negli smartwatch sfruttando l'ascolto dei sensori di questi ultimi. Inoltre Snoopy è in grado anche di captare le APL che risulta essere una sfida molto più complessa.
Prima di descrivere il funzionamento di Snoopy è necessario introdurre alcune informazioni preliminari. Innanzitutto lo smartwatch ha come principali metodi di autenticazione dei meccanismi che come input permettono di toccare lo schermo o farci scorrere il dito sopra. Per piattaforme IoS la password di default è composta da un pin di quattro cifre mentre Andorid lascia di dafault la scelta fra pin e APL. I pin a 4 cifro offrono un totale di 9999 possibili combinazioni. Le APL, invece, offrono una quantità di combinazioni decisamente maggiore, limitata solo dalla fattibilità delle traiettorie, che si aggira intorno alle 320 mila.
Solitamente i tentativi per indovinare la sequenza o la password per accedere al dispositivo sono 3: una volta terminati lo smartwatch rimane inattivo per qualche minuto.

Intuitivamente, le azioni necessarie ad inserire le credenziali, che siano tapping o swiping, inducono un'applicazione di forza e un cambio di orientazione nel dispositivo che sono misurabili dai sensori presenti. Questa è la base di partenza che garantisce una involontaria fuoriuscita di informazioni da parte dello user che saranno poi sfruttate da Snoopy.
Prima di parlare dell'effettivamento funzionamento della tecnica fornisco una overview generale di Snoopy.
Innanzitutto affinchè l'attacco avvenga con successo, si assume che lo user scarichi all'interno del dispositivo Snoopy, che non è altro che un Trojan difficilmente riconoscibile da una normale app. Snoopy richiede l'accesso ai sensori di movimento ed ad Internet, che in Andorid o IoS non richiedono specifici permessi. La quantità di dati che invia sono molto pochi (circa 3kByte) per riuscire ad evitare notifiche a schermo che mostrano l'utilizzo di banda.
Gli obiettivi dell'attacco sono due: il primo è riuscire a raccogliere una grande quantità di dati collegata all'immisione delle password e successivamente riuscire a risalire a queste ultime.
L'architettura di Snoopy è composta da una parte di front-end e da una di back-end. La parte di front-end risulta simile ad una normale app di fitness e una volta installata ha lo scopo di acquisire dati relativi ai sensori di movimento. La frequenza di campionamento è mantenuto molto bassa per non destare sospetti al sistema operativo. Ogni volta che un evento viene valutato come possibile candidato, Snoopy raccoglie informazioni anche dal giroscopio. Una volta registrato l'evento quest'ultimo viene classificato con un ulteriore analisi come papabile evento di inserimento password o meno. Se la classificazione lo identifica come tale, i dati raccolti sono inviati alla parte responsabile del back-end ed analizzati ulteriormente.
La parte di back-end di Snoopy ha lo scopo di analizzare i dati ottenuti ulteriormente e da questi provare a risalire alla password. Per fare ciò viene usato un approcio di deep-learning

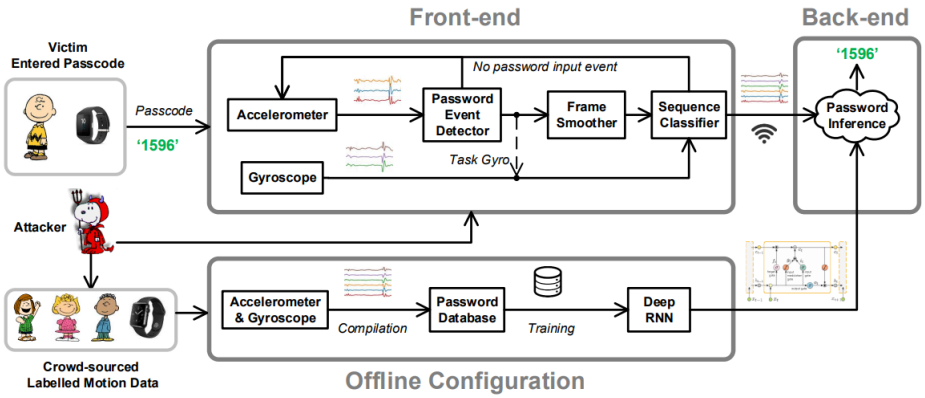


Figura 2.3: Overview Snoopy

Nonostante le operazioni fatte da Snoopy possano sembrare molto onerose, il consumo di batteria che ha è molto basso. In particolare Snoopy opera in 3

modalità: ascolto passivo, monitoraggio immissione password e estrazione dati dei sensori. La maggior parte del tempo Snoopy la passa in ascolto passivo. Durante questa modalità le sue funzioni sono limitate al semplice ascolto dei dati raccolti dall'accelerometro. Quando rileva la possibilità di interazione da parte dello user con il dispositivo Snoopy passa alla seconda modalità ed aumenta la frequenza di raccolta dati. Se si accorge della possibilità che una password venga inserita, passa alla terza modalità ed attiva anche il campionamento sui dati raccolti dal giroscopio. I dati sono poi salvati, classificati ed in caso passati al back-end. Segue una tabella con i consumi di batteria di Snoopy

Model	SoC	RAM	Battery Cap.	Task	CPU load (avg/max)	Current delta	Battery Usage
Sony SW3	Qualcomm APQ8026 SD 400	512MB	420 mAh	Detection	9.2%/17.5%	8.9 mA	2% per hr
				Smoothing	7.2%/15.2%	3.1 mA	
				Uploading	2.4%/9.9%	18.9 mA	
Samsung Gear Live	Qualcomm MSM8226 SD 400	512MB	300 mAh	Detection	8.1%/19.4%	11.4 mA	3% per hr
				Smoothing	15.6%/29.3%	6.2 mA	
				Uploading	2.1%/19.3%	25.1 mA	
Moto 360 Sports	Qualcomm MSM8926 SD 400	512MB	300 mAh	Detection	8.3%/26.2%	16.1 mA	3% per hr
				Smoothing	7.3%/22.4%	3.8 mA	
				Uploading	2.3%/26.1%	22.3 mA	

Figura 2.4: Utilizzo CPU e batteria Snoopy

Come si può osservare, la quantità di batteria usata è veramente poca, circa 2,3 percento all'ora.
Per conlcudere, Snoopy è in grado di estrarre i dati relativi all'inserimento delle passowrd nel 98% dei casi e avendo a disposizione dieci tentativi riesce a ottenere nel 95% dei casi le APL nell'86% dei casi i PIN.

2.3 APL in 5 tentativi via shoulder surfing

La seconda tecnica di attacco proposdta ricade nella categoria dello shoulder surfing. Il funzionamento base si divide in cinque step. Inizialmente la vittima viene filmata dall'attaccante mentre sblocca il proprio dispositivo. Questo video viene poi editato per riprendere solo l'atto in cui la sequenza di sblocco viene tracciata e vengono marcate due aree, ossia la supperfice dello schermo e il dito usato per lo sblocco. Successivamente un algoritmo di computer vision viene utilizzato per identificare in ogni frame la posizione del dito. Questi frame servono poi per ricostruire la traiettoria di quest ultimo lungo lo schermo. La linea ottenuta va considerata in funzione dell'angolo con cui il video è stato fatto che viene calcolato nel terzo step. Arrivati al quarto step la quantità di informazione raccolta è sufficiente per stilare una lista di pattern canditati che verranno poi testati nel quinto ed ultimo step.
La miglior qualità di questa tecnica è che non richiede che la videocamera sia puntata direttamente sullo schermo, ma può riprenderlo da varie angolazioni e ad una distanza di oltre due metri. Inoltre l'efficiacia è del 95% su circa 120 casi studiati. Essa arriva al 97% se i tentativi a disposizione sono cinque.

Nelle seguenti immagini viene mostrato prima come si corregge la sequenza candidata tenendo in conto l'angolo e poi ci si riconduce alla corretta sequenza.

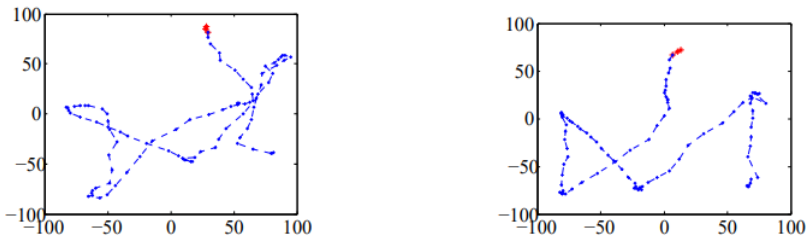


Figura 2.5: Calibrazione Videocamera

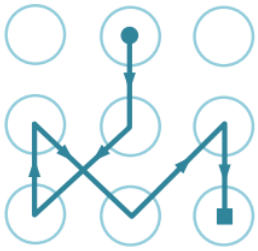


Figura 2.6: Pattern Corretto

2.4 Smudge Attack

In questo strategia d'attacco l'espediente usato è la traccia d'olio lasciata dalle mani dello user sul touch screen del dispositivo[**Avi+10**]. E' infatti possibile risalire a quali siano pin o sequenze di sblocco analizzando le macchie lasciate sullo schermo. Lo studio di tale meccanismo come strategia d'attacco si basa su tre punti fondamentali. Primo, le macchie sono persistenti nel tempo. Secondo, cambiare involontariamente le macchie lasciate inserendo il telefono in tasca o pulendolo è complicato. Terzo e ultimo punto, raccogliere e analizzare i residui d'olio sui dispositivi è molto semplice. Bastano infatti una telecamera ed un computer. Nello studio proposto dall'università delle Pennsylvania nel 92% dei casi analizzzati, era possibile riconoscere quasi del tutto le sequenze e i pin utilizzati e nel 68% dei casi riconoscerli del tutto. Le condizioni sperimentali sotto le quali sono stati eseguiti gli attacchi erano le seguenti: l'attaccante aveva il dispositivo in proprio possesso e si trovava in un posto dove gli era permesso illuminare in vari modi lo schermo e disponeva di un computer. La strategia per rilevare le macchie lasciate sullo schermo consiste nel illuminare quest ultimo da varie angolazioni e scattare delle foto, successivamente passate ad un pc che le processa. La combinazione di più immagini permette di risalire alle sequenze di sblocco usate o algi eventuali pin a quattro cifre.

Nonostante le condizioni in cui gli esperimenti sopno stati svolti siano a tutti gli effetti favorevoli, la possibilità di mettere in pratica questo attacco è possibile anche all'aria aperta. L'unica difficoltà riscontrata riguarda la difficoltà di scattare foto sfruttando corrette angolazioni con una corretta illuminazione.

2.5 Resoconto e Considerazioni Finali

Questo capitolo preliminare al lavoro su cui si basa la tesi ha lo scopo di mostrare come gli smartwatch siano vulnerabili a diversi tipi di attacchi e che la sicurezza di questi ultimi è un aspetto a cui prestare attenzione data la quantità di informazioni che possono essere sottratte. Nel prossimo capitolo si affronterà una tematica opposta ossia tecniche di difesa che provano a ridurre l'efficacia degli attacchi sopra citati.

Capitolo 3

Tecniche di Difesa nella fase di log-in

Il precedente capitolo ha messo in luce la facilità con la quale sia possibile superare i sistemi di difesa implementati negli smartwatch ed accedere alle informazioni contenute in essi. La sfida più difficile è rappresentata dal fatto che l'utente medio è pigro e sistemi troppo complessi, codici da memorizzare troppo lunghi e una quantità elevata di passaggi per ottenere una difesa migliore contro eventuali attacchi non sono adatti. L'idea è quella di trovare un compromesso tra velocità e facilità nella fase di log-in mantenendo un buon livello di sicurezza.
Prima di parlare di possibili soluzioni è necessario dire che l'autenticazione di uno user si basa su almeno uno dei tre meccanismi che seguono: "qualcosa che so", "qualcosa che ho", "qualcosa che sono".
Per "qualcosa che so" si intende solitamente una password, qualcosa di segreto che solo io conosco e che mi permette di autenticarmi sul dispositivo inserendo l'informazione segreta che verrà poi controllata. Nel caso degli Smartwatch, come detto in precedenza, questo meccanismo è solitamente implementato tramite uso di PIN o APL. "Qualcosa che ho", invece, esplora la possibilità di escludere la memorizzazione di una informazione che potrebbe essere dimenticata e di sostituirla con altro: per esempio una carta magnetica da far scorrere su un lettore, un autorizzazione da un dispositivo primario ad uno secondario. In questo caso, se uno Smartwatch è connesso ad uno Smartphone, qualcosa che ho risulta come un secondo livello di sicurezza. Infatti se inserisco il mio Pin sullo smartwath posso richiedere una seconda verifica tramite smartphone. Infine per "qualcosa che sono" si intende una caratteristica unica dello user che lo distingue dagli altri e che gli permette di autenticarsi facendo leva sulla sua unicità: un esempio è il classico finger print su uno smartphone o un lettore di retina.
Nelle successive due sezioni vengono proposte delle soluzioni per aumentare l'efficacia dell'autenticazione specificatamente per smartwatch.

3.1 Qualcosa che sono: Biometrics Behavior

L'idea alla base di questa tecnica di difesa[**Lu+17**] è che ogni utente abbia un suo modo di inserire le credenziali. Come già detto prima, tapping e swiping agiscono sul dispositivo imprimendo una certa forza e facendogli assumere una certa inclinazione. Recenti studi[**SB14**] hanno dimostrato che il metodo di inserimento delle credenziali varia di molto da persona a persona e la ricca quantità di sensori presenti negli smartwatch ci può permettere di capire le abitudini dello user, apprenderle ed usarle a nostro vantaggio come secondo livello di sicurezza. L'insidia più grossa per la realizzazione di tale meccnismo è rappresentata dal fatto che i livelli di rumore nei dati raccolti su smartwatch sono molto elevati e rendono il campionamento dei dati difficile. Inoltre, per riuscire nell'intento è necessario salvare i dati raccolti e le effettive passowrd e mandarle in clud per elabolarle. Questa operazione da sola solleva problemi di privacy da gestire.
Per addestrare l'orologio a riconoscere l'unicità del proprio proprietario si usano tecniche di machine learning, in particolare reti neurali. Nell'immagine proposta viene mostrato come i sensori ottengano misure diverse da due differenti utenti. Nonostante il pin inserito sia corretto in entrambi i casi solo con il primo inserimento il dispositivo verrà sbloccato.

Figura 3.1: Differenze rilevate nell'inserire il PIN

Ovviamente l'efficacia di tale approcio risulterà migliore in base alla qualità dell'addestramento fornito alla rete neurale.

3.2 Qualcosa che so: Two Gesture Pin

Il secondo meccanismo di difesa di cui voglio parlare è il two gesture pin[**Gue+20b**] che sfrutta le feature TEE del dispostivo e concilia sicurezza e facilità di utilizzo. Prima di descriverne il funzionamento vorrei porre l'attenzione sulla sigla TEE. Per TEE (trusted execution environment) si intende un ambiente esecutivo parallelo al sistema operativo che fornisce strumenti di difesa aggiuntivi. In particolare il TEE combina elementi hardware e software contemporaneamente per garantire la sicurezza del dispositivo. Inoltre il TEE ha a disposizione una

serie di risorse utilizzabili da lui che possono essere in caso condivise (ad esempio con dei sensori). Il TEE inoltre ha un proprio sistema crittografico cablato nella ROM.
Fatte queste premesse il funzionamento del Two Gesture Pin è tanto semplice quanto geniale. Osservando la figura è possibile notare che sullo schemro appaiono due moduli circolari uno indipendente dall'altro: il modulo esterno è fisso e segue un ordine crescente per la disposizione delle cifre mentre il modulo interno è composto da dieci cifre che vanno da zero a nove.

Figura 3.2: Interfaccia Two Gesture Pin

Usando la corona dell'orologio è possibile ruotare il modulo interno e formare la prima metà del PIN nel primo step e la seconda metà nel secondo step. In particolare nella seconda figura la prima parte del PIN è 73 mentre la seconda è 40.

Figura 3.3: Funzionamento Two Gesture Pin

Il two-gesture pin è in grado di evitare alcune forme di attacco precendentemente descritte. Innanzitutto, se si pensa al side-channel attack, twp gesture pin ci permette di non inviare dati campione in quanto non ci sono movimenti di tapping swiping. Allo stesso tempo, ci protegge da smudge attack in quanto è la corona a fungere la meccanismo per inserire le credenziali. Infine, lo shoulder surfing risulta inefficace perchè la combinazione nonostante sia visibile, risulta "nascosta" in mezzo ad altre nove papabili combinazioni. Come detto all'inizio del capitolo la sfida per questi sistemi di difesa è quella di essere sostenibili nell'uso ossia lo user non deve essere appesantito dal loro utilizzo. Two gesture PIN è l'esempio perfetto di questa filosofia.

Capitolo 4

Sviluppo Applicazione

Nei due capitoli precedenti sono state trattate tematiche di attacco e di difesa per quando riguarda il mondo degli smartwatch. Le considerazioni che ho fatto sono state diverse. Innanzitutto è pericoloso inserire le credenziali in un luogo non sicuro, ossia frequentato da gente che non conosco e sul quale non ho controllo, perchè potrei essere vittima di shoulder surfing. Inoltre inserire spesso i vari PIN o tracciare spesso le APL è altrettanto rischioso perchè se qualcuno sta prendendo informazione dai miei sensori più volte inserisco le mie credenziali più campioni fornisco all'attaccante. Per quanto riguarda invece i sistemi di difesa, credo che sia furbo munirsi di sistemi atipici e non utilizzare invece i classici proposti di default.
Ricapitolando il tutto, uno user necessità di inserire poche volte le credenziali e che esse siano facili da ricordare e da inserire. Questa tesi studia la possibilità di fornire una soluzione a queste necessità. L'idea di base è la seguente: lo user si autentica una sola volta in un luogo sicuro, ad esempio casa propria quando idossa lo smartwatch, e lo smartwatch apre una sessione all'interno della quale lo user è loggato e può compiere le azioni che desidera. Per tenere aperta e sicura questa sessione il meccanismo usato è quello di monitorare, tramite l'uso dei sensori, che lo smartwatch non venga sottratto. Se questo viene sottratto la sessione viene chiusa. Il fatto che la sessione venga chiusa implica che chi ha ora il dispositivo deve loggarsi di nuovo per avere accesso alla sue funzionalità e poichè le credenziali vengono inserite una volta sola ed in un luogo protetto rende più difficile che l'attaccante disponga di esse.
Nelle successive sezioni verrà approfondito meglio come funziona l'app che simula il prolungamneto della sessione: in particolare verrà spiegato come rilevare movimenti sospetti e come assicurarsi che lo user abbia ancora l'orologio.

4.1 General Overview

In questa sezione viene mostrato nello specifico come funziona l'app. Per prima cosa viene creata l'activity principale. In questo caso si suppone che lo user si sia loggato nel dispositivo e che la sessione sia stata creata correttamente. Successivamente viene inizializzato un service eseguito in sottofondo: il suo scopo è

quello di monitorare i movimenti a cui è sottoposto il dispositivo tramite l'uso dell accelerometro.
All'interno del service ci sono due meccanismi che si alternano. Alla prima invocazione dell'OnCreate() vengono abilitati i sensori dell'accelerometro all'ascolto. I valori campionati sono tre e corrispondono alle tre coordinate spaziali. Asse x,y e z. A seconda di come muoviamo il dispositivo otteniamo dei valori: se per esempio, dopo aver disposto lo smartwatch su un tavolo lo spingiamo da sinistra (lui si muove a destra) il valore dell'asse x aumenterà e sarà positivo. Nell'app i valori sono salvati all'interno di un array e controllati con un ciclo if. Se il loro valore assoluto (siccome posso ottenere anche valori negativi converto in valore assoluto) supera una certa soglia si attiva la seconda parte di codice.
Nella seconda parte di codice viene misurata la presenza di battito cardiaco per accertarsi che lo user indossi ancora il dispositivo. Testando il funzionamento dell'app mi sono accorto che la misurazione a volte non è riconosciuta istantaneamente quindi lascio un certo lasso di tempo per fare diverse misurazioni. In particolare, l'orologio ogni tot tempo effettua un'acquisizione di un evento (in questo caso si cerca di sentire il battito) e lo fa per un numero massimo di volte che ho impostato come limite (35).
Effetuate le misurazioni se c'è battito si ritorna al primo pezzo di codice, ossia rilevazione di movimenti sospetti. Se invece non viene rilevato il battito cardiaco la sessione viene chiusa. Nel caso specifico dell'app, per essere funzionale al testing, viene cambiato il valore di una variabile boolean e le funzioni di monitoring da parte dei vari sensori sono interrotte. Dalla main activity cliccando il pulsante REFRESH si invia un messaggio al service che risponde con un codice. Se il codice è EXIT1 la sessione è aperta, se il codice è EXIT2 la sessione è chiusa. Nel caso in cui l'app non dovesse essere stata testata sarebbe stato sufficiente inviare un messaggi da parte del service all'activity e innescare delle righe di codice per tornare ad una ipotetica fase di login che avrebbe di fatto interrotto la sessione.

4.1.1 Life Cycle Activity e Service

Per capire appieno come funziona l'app, reputo sia necessario fare una digressione e spiegare il funzionamento dei life cycle delle activities e dei services.
Innanzitutto l'activity è un blocco fondamentale per la programmazione in Android e sostanzialmente permette allo user di interfacciarsi con il dispositivo: infatti a differenza del service, l'activity implementa quasi sempre una UI con la quale interagire. Alla sua creazione il pezzo di codice che viene eseguito una sola volta è OnCreate() e permette di eseguire operazioni chiavi per quella specifica activity. Una volta eseguito OnCreate() e successivamente OnStart() l'activity è funzionante. Come si può vedere nell'immagine ci sono altri comandi che possono modificare il funzionamento di quest'ultima.
Una volta che l'activity è in esecuzione ci possono essere due situazioni che si presentano: una nuova activity prende il suo posto e quindi quella di partenza viene fermata tramite il comando OnStop(), oppure un'activity si sovrappone

momentaneamente ad essa e l'activity di partenza è messa in pausa. Una volta messa in pausa tramite il comando OnResume() torna a funzionare. In questo caso è da far notare che OnResume() salta il blocco funzionale OnStart() che solitamente insieme ad OnCreate() contiene informazioni di inizializzazione per l'activity. Se invece l'activity viene fermata del tutto ci sono tre possibili scenari. Nel primo caso viene invocato OnDestroy() e l'activity viene completamente cancellata. Nel secondo tramite il comando OnRestart() si torna al blocco OnStart() e l'activity riprende il suo funzionamento. Nel terzo caso invece quando un activity è bloccata da OnStop() se esiste un'altra app o più app a priorità maggiore che chiedono risorse al dispositivo l'activity viene distrutta.

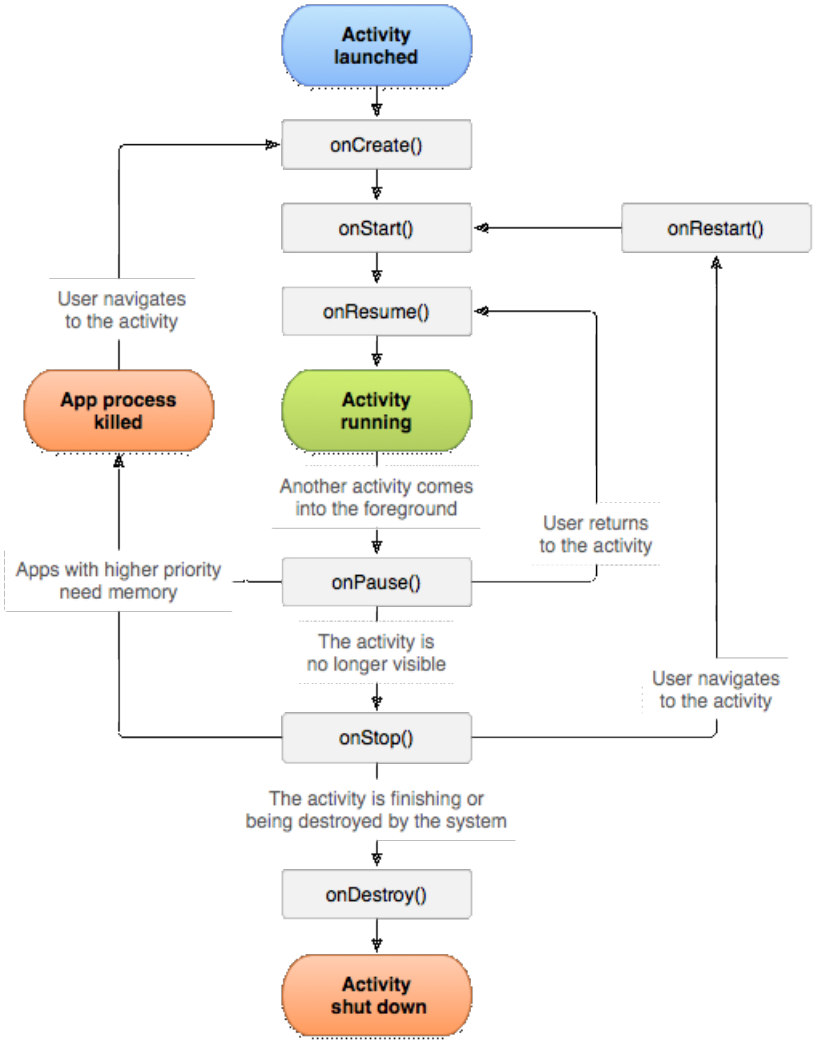


Figura 4.1: Lifecycle Activity

Per quanto riguarda il service la situazione è differente. Il service può essere avviato solo da una activity e non da un altro service e il service non può avviare una activity (questa funzionalità era una volta permessa ma è stata cambiata). All'avvio del service ci possono essere due situazioni distinte: il service non è

legato all'activity chiamante(parte a sinistra della figura), oppure il service è legato all'activity tramite la funzionalità OnBind().

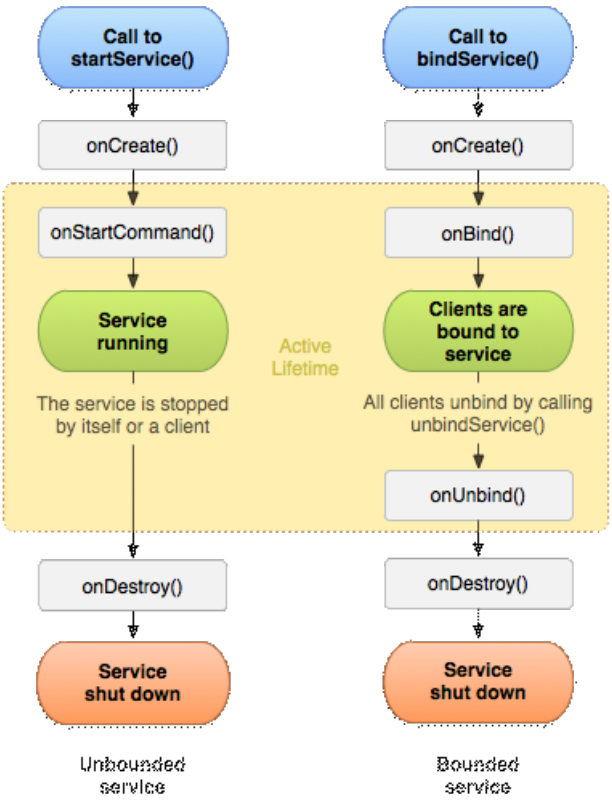


Figura 4.2: Funzionamento App

Anche in questo caso OnCreate() è la parte di codice utilizzata per inizializzare il service e viene eseguita una sola volta ossia alla creazione. OnStartCommand(), invece, viene eseguito dopo OnCreate() se il service non è legato all'activity chiamante. Per non legato si intende che non può interagire con la UI dell'activity. Esistono vari tipi di service. Quello usato nel mio codice è un foreground service e non è legato all'attività chiamante.
Come per le activities i services terminano solitamente dopo la chiamata del metodo OnDestroy(). Nell'app da me implementata per la fase di monitorin (nella prima fase di test) ho usato un service che lavora in background: questo permette che i sensori funzionino anche se l'app viene ridotta o se lo schermo del dispositivo si spegne per un breve lasso di tempo.

4.2 Movimento Sospetto

Per movimento sospetto si intende un qualsiasi movimento che da come risultato una misurazione da parte dei sensori (accelerometro) del dispositivo che oltre-

passa il limite numerico che ho imposto.
Il limite è stato scelto dopo aver fatto diverse prove ed aver raccolto dati svolgendo una serie di attività con l'orologio al polso misurando e registrando i valori ottenuti dall'accelerometro.
In particolare, il valore numerico è stato scelto a ribasso, nel senso che è preferibile rilevare più movimenti sospetti ed accertarmi del battito cardiaco piuttosto che "mancare" una misurazione che potrebbe essere sospetta.
In seguito fornisco una tabella che riporta dei dati raccolti secondo la seguente disposizione:

- Prima colonna: attività svolta
- Seconda colonna: misurazione media asse X
- Terza colonna: misurazione media asse Y
- Quarta colonna: misurazione media asse Z
- Quinta colonna: verifica se almeno una volta un valore oltre alla soglia sia stato misurato

Camminare	15.300	13.500	14.000	MISURATO
Guidare	12.000	14.300	5.600	NON MISURATO
Usare il PC	8.500	14.700	6.100	NON MISURATO
Cucinare	12.500	16.700	8.100	MISURATO
Mangiare	10.900	18.100	7.500	MISURATO

Questi dati sono stati raccolti campionando circa una ventina di volte ogni azione. Le misurazioni sono state fatte tenendo l'orologio sul polso sinistro di una persona alta circa 170 cm. Inoltre le azioni scelte per il campionamento sono le azioni più comuni svolte durante la giornata da una persona nella media.
Oltre a questi dati, ho fatto delle altre prove nelle quali muovevo il polso in maniera rapida come se il dispositivo mi stesse venendo sottratto. I valori erano ogni volta (almeno uno dei tre) superiori a 20 che è la soglia che ho deciso di imporre all'orologio come limite durante la rilevazione del movimento sospetto.
Nonostante solo uno dei valori riportati in tabella si avvicini a 20, scegliere un valore più basso come limite comportava l'attivazione dei sensore di battito cardiaco troppe volte durante la fase di prova. Proprio per questo la soglia è stata alzata da 19 a 20. Altre soglie valide potrebbero essere 22 e 25 ma preferisco catturare più movimenti sospetti che tralasciarne alcuni.

4.3 Utilizzo Sensori

In Android esiste una classe specifica per gestire l'utilizzo dei sensori ossia Android SensorManager. Abbinata ad essa, la nostra classe che estende Activity o Service, deve implementare SensorEventListener che permette di ricevere notifiche da parte di SensorManager. Le notifiche non sono altro che dei SensorEvent che , a seconda del tipo di sensore che stiamo ascoltando, rachiudono certi tipi

di informazione.

In figura è mostrato come preparare i sensori all'ascolto e alla ricerca di Sen-

```
private void prepareAccSensors(){
    mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mSensorManager.registerListener( listener: this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
}

private void prepareHSensor(){
    hSensorManager = (SensorManager)this.getSystemService(Context.SENSOR_SERVICE);
    hHeartBeat = hSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
}
```

Figura 4.3: Preparazione Sensori

sorEvent a cui siamo interessati. In questo caso con le prime tre linee di codice creo un manager di eventi per quanto riguarda il sensore dell'accelerometro e lo predispongo all'ascolto usando il metodo registerListener(). Le altre linee di codice servono solo ad inizializzare il manager per il sensore cardiaco che verrà utilizzato effettivamente solo una volta rilevato un movimento sospetto.
Ogni volta che la nostra classe implementa SensorEventListener deve implementare due metodi ossia onAccuracyChanged e onSensorChanged. Segue l'esempio di codice di dafault fornito da Andorid.

```
public class SensorActivity extends Activity implements SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NOR
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    public void onSensorChanged(SensorEvent event) {
    }
}
```

Figura 4.4: Esempio di default su come usare i sensori

Questi due metodi da implementare hanno la seguente funzione: onAccuracyChanged ci permette di ricalibrare i sensori se durante una misurazione si sfasano,

mentre onSensorChanged ci permette di gestire i dati raccolti dal sensorManager che sta utilizzando il sensore che al momento vogliamo.

```
public void onSensorChanged(SensorEvent event){
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
        mAccelerometerValues = event.values;
        int asse1 = (int) Math.abs(mAccelerometerValues[0]);
        int asse2 = (int) Math.abs(mAccelerometerValues[1]);
        int asse3 = (int) Math.abs(mAccelerometerValues[2]);
        if(asse1>5 || asse2>5 || asse3>5){
            exitCode1();
        }
    }
    else if (event.sensor.getType() == Sensor.TYPE_HEART_RATE){
        int valore = (int) event.values[0];
        exitCode2(valore);
    }
}
```

Figura 4.5: Gestione eventi app

Questo codice è quello usato nell'app. Il funzionamento è il seguente: se stiamo usando come sensore l'accelerometro gli eventi che registriamo sono di tipo TYPE.ACCELEROMETER e quindi entriamo nel primo if dove. una volta salvati i valori controlliamo che siano inferiori alla soglia stabilita. In questo caso la soglia era cinque e serviva per testare velocemente che l'app funzionasse correttamente. Se gli eventi sono di tipo TYPE.HEARTRATE si entra nel secondo if, si acquisisce il valore numerico misurato e lo si utilizza per capire se c'è battito o no.

4.4 Testing e Limiti App

La fase di testing è stata composta da principalmente due attività differenti. La prima consisteva nel verificare l'efficacia del meccanismo, ossia valutare quanto bene funzionasse il passaggio da accelerometro a sensore caridaco. La seconda, invece, consisteva nel testare nel lungo periodo l'applicazione, valutando la quantità media di eventi registrati e i consumi della batteria.
Per quanto riguarda la prima fase di testing, la bontà del codice è stata testata nella seguente maniera: il limite imposto come movimento sospetto è stato portato ad un valore di 1. Questo accorgimento fà in modo che ogni volta che viene effettuata una misura dall'accelerometro essa figuri come pericolosa in quanto sorpassa la soglia imposta. Così facendo veniva innescata la seconda parte di codice che misurava il battito cardiaco.
Questa prima fase di sperimentazione è stata utile anche per calibrare il lasso temporale concesso allo smartwatch per acquisire i dati sul battito cardiaco. In particolare, il numero di acquisizioni è stato impostato cercando un compromesso tra tempo impiegato e percentuale di successo.
Nella prima immagine il grafico mostra la percentuale di successo associata al numero di misurazioni

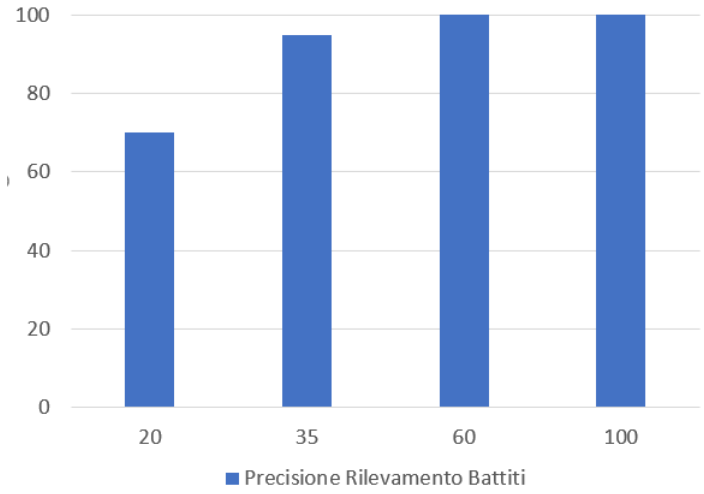


Figura 4.6: Percentuale di Successo di Rilevamento Battiti

Dalla figura si può notare che se il numero di campioni per cercare il battito supera i 35 eventi più di 9 volte su 10 il battito è trovato.
La pericolosità nel misurare per troppo tempo la presenza del battito risiede nel fatto che se l'orologio venisse sottratto, l'autore del furto avrebbe a disposizione una finestra temporale considerevole per indossare il dispositivo che, trovando il battito, manterrebbe la sessione aperta. Proprio per questo motivo è stato necessario trovare un trade-off sul numero di eventi da campionare.
Nella seconda fase di testing, l'obiettivco principale è stato quello di utilizzare l'applicazione per un periodo di tempo considerevole, reimpostando il limite per la rilevazione di movimenti sospetti a 20.
Qui sono sorti alcuni problemi. In particolare, Andorid ha limitato enormemente la possibilità di sviluppare applicazione che eseguono operazioni in background per grandi quantità di tempo. La ragione di questa limitazione sta nel fatto che si vuole in tutti i modi evitare sprechi energetici da parte della batteria. Questa limitazione impediva di fatto all'applicazione di funzionare correttamente nel tempo. Infatti, dopo essere stata avviata, se lo schermo del dispositivo si spegneva o l'app veniva ridotta ad icona manteneva le sue funzionalità per un lasso di tempo di circa uno o due minuti per poi essere messa in pausa dal sistema.
Per superare questa limitazioni ho adottato la seguente strategia: il service che creo all'intenro dell'applicazione viene eseguito in foreground e non più in background e l'activity che inizializza il service mantiene lo schermo del dispositivo sempre acceso. Nonostante sia una soluzione brute force, funziona e permette di testare effettivamente il funzionamento dell'applicazione.
Passando alla vera e propria fase di testing, l'applicazione inizialmente è stata testata in finestre temporali di circa dieci minuti per stimare una media di quante volte la misurazione del battito viene eseguita. Per ogni movimento sospetto rilevato è stata inserita una casella rossa all'interno del grafico (circa al minutaggio in cui è stato acquisito l'evento) e una verde se la sessione è stata prolungata correttamente. Alla fine dei dieci minuti l'orologio veniva tolto dal

polso e successivamente si verificava che la sessione venisse chiusa correttamente. Se la chiusura della sessione fallisce viene inserita una casella blu.

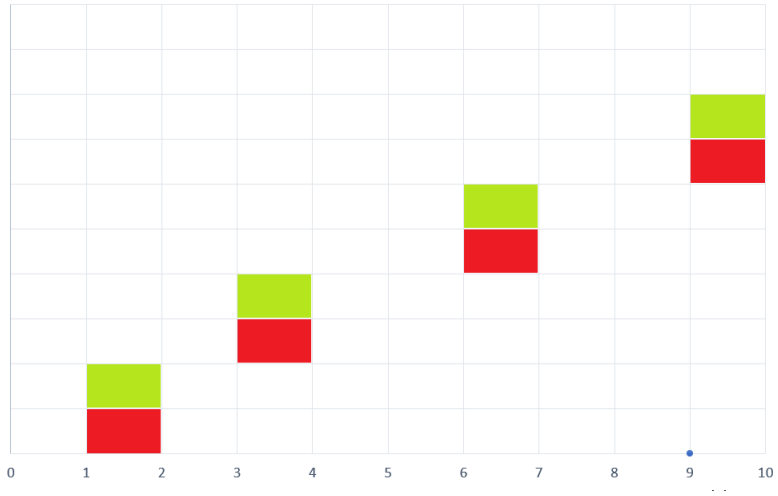


Figura 4.7: Test 1

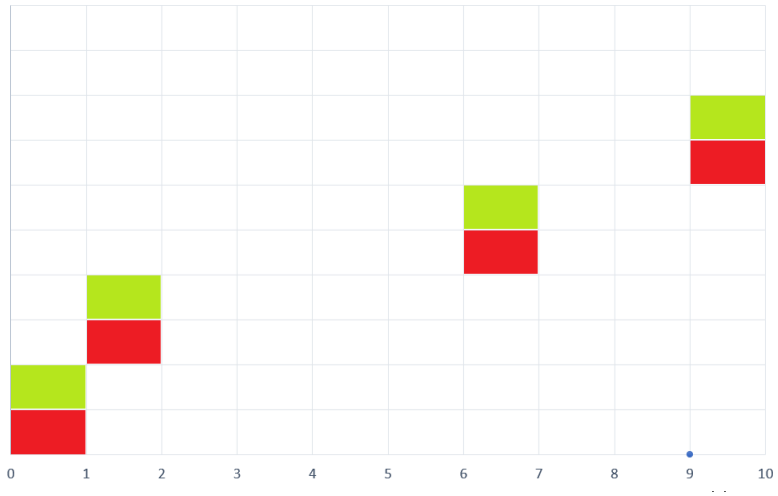


Figura 4.8: Test 2

Questi due grafici sono il risultato di venti minuti di monitoring eseguito durante la stesura della tesi. I movimenti sospetti misurati sono stati effettivamente pochi (3 per sessione più il quarto per verificare la chiusura di quest'ultima) in quanto nessun genere di attività motoria intensa è stata fatta durante la registrazione. Questo è una prova di come il limite imposto a 20 sia effettivamente ponderato correttamente.
Nella fase di testing solo una volta la sessione non è stata chiusa correttamente. Questo "fallimento" si è verificato perchè a volte il rilevatore di battito non è precisissimo.
Di seguito viene riportato il grafico della sessione, all'interno del quale il rile-

vamento fallimentare del battito è stato riportato in blu. Tale situazione si è verificata solo una volta su più di 50 sessioni.

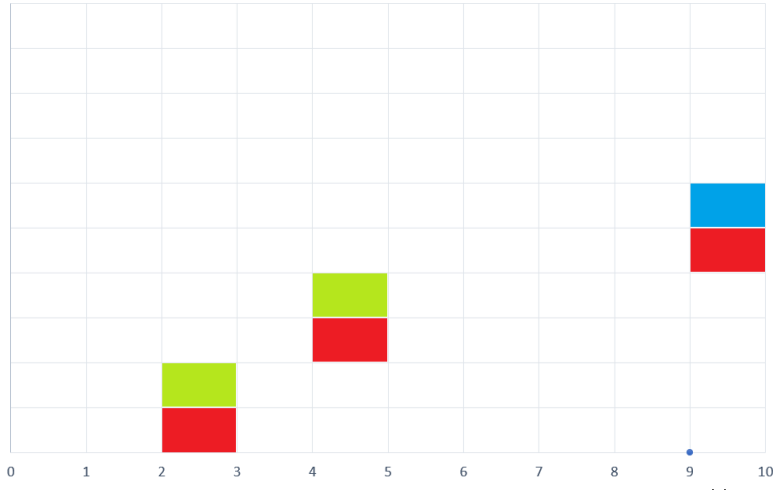


Figura 4.9: Test 3

Successivamente l'applicazione è stata testata per lassi temporali maggiori. Il seguente grafico riporta una fase di testing di circa un'ora. In questo caso gli eventi sono stati segnati come dei puntini all'interno del grafico. Poichè la sessione è stata prolungata con successo per tutto il tempo il colore degli eventi è stato tenuto lo stesso su tutto il grafico. Anche il questo caso l'asse delle x rappresenta il minutaggio.

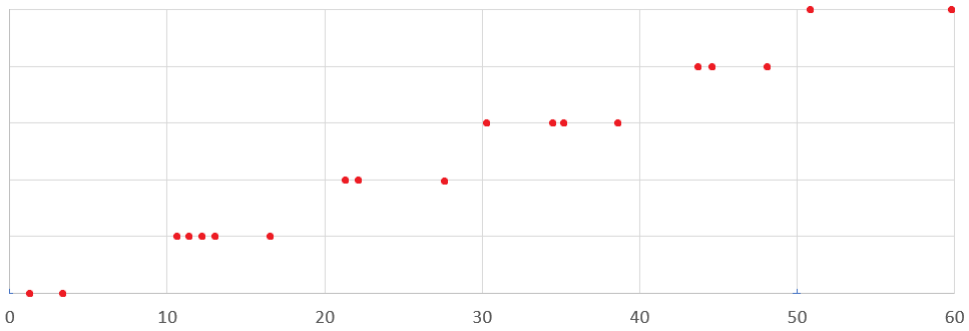


Figura 4.10: Test di Un'ora

Nei punti dove si raggruppano di puntini di fila, c'è stato un tipo di azione che sorpassava i limiti imposti dall'accelerometro. Ergo dopo una misurazione fatta con battito trovato ne è iniziata subito dopo un'altra.
Anche in questo caso il limite imposto a 20 sembra aver funzionato bene in quanto gli eventi campionati si sono mantenuti su una buona media.

4.5 Consumo Batteria

Negli anni precedenti è stato fatto un lavoro simile a questo che sfruttava solo il battito cardiaco come garante della presenza dello user. In pratica il battito veniva costantemente misurato per mantenere la sessione aperta e appena quest'ultimo veniva a mancare la sessione veniva interrotta. Il problema maggiore di questo approccio è che il consumo della batteria è esagerato.
Nell'approcio da me proposto la maggior parte del lavoro è fatto dall'accelerometro che consuma, secondo l'API di Android, mediamente dieci volte in meno rispetto agli altri sensori e dallo schermo che rimane acceso durante l'intero processo.
La batteria del Galaxy Watch4 dura circa un giorno e mezzo se tenuto su un tavolo fermo senza compiere azioni e dura circa 20 ore se tenuto al polso senza aprire applicazione che usano in modo intensivo la batteria.
Durante la fase di testing ho tenuto traccia dei consumi della batteria. Per ridurre il consumo ho lasciato lo sfondo dello schermo di color nero e le scritte presenti sull'app in bianco.
La quantità media di batteria consumata varia ovviamente anche in base alla frequenza con cui viene misurato il battito. Nella sessione sopra campionata dalla durata di un'ora la batteria è scesa di circa il sei percento. Questi sono dati molto positivi e mostrano in parte la fattibilità di questo approcio anche a livello energetico.

Capitolo 5

Conclusioni

L'indipendenza raggiunta dagli Smartwatch li rende soggetti a diversi attacchi. In particolare shoulder surfing e side-channel attack. Per difendersi da questo tipo di minacce una possibile soluzione è quella di loggarsi sul dispositivo una volta e mantenere la sessione aperta. In questo elaborato è stata sviluppata un'app che esplora questa possibilità.
I risultati ottenuti sono molto incoraggianti. Innanzitutto, l'applicazione riesce con successo a riconoscere movimenti sospetti, che potrebbero portare alla sottrazione del dispositivo, e a misurare successivamente il battito cardiaco, garante del fatto che lo smartwatch sia ancora al polso dello user. Inoltre l'applicazione da un punto di vista energetico è sostenibile nel senso che il consumo di batteria permette all'applicazione di funzionare per periodi di tempo molto lunghi.
Per colcudere, mi sento di dire come la realizzabilità del meccanismo qui studiato sia valido e che una implementazione direttamente nel s.o. possa in futuro rivelarsi un ottimo meccanismo per la sicurezza degli Smartwatch.

5.1 Sviluppi Futuri

Uno dei limiti di questo lavoro sta nel fatto che l'unico modo per determinare se lo user sta subendo un furto è quello di controllare dei valori numerici ottenuti dall'accelerometro senza tener conto di alcuni possibili scenari. Per esempio, se lo user si sfila l'orologio volontariamente per appoggiarlo e questo viene sottratto o se la forza impressa per prendere il dispositivo non supera la soglia imposta non c'è il controllo del battito ergo la sessione rimarrebbe aperta lasciando pieno controllo sullo smartwatch a furto avvenuto.
Una possibile soluzione a questo sarebbe implementare tramite Machine Learning la possibilità di riconoscere i movimenti a cui viene sottoposto il dispositivo. In particolare l'idea sarebbe quella di addestrare una rete neurale.
In questa tesi questo ulteriore step non è stato fatto in quanto lo scopo era quello di effettuare un primo studio sulla possibilità di mantenere aperta una sessione tramite uso di accelerometro e sensore cardiaco.

Appendice A

Scelta Smartwatch

Per lo sviluppo della tesi si è reso necessario l'acquisto di uno smartwatch. In particolare questa sezione giustificherà la scelta del modello (questa sezione è stata fatta prima dell'acquisto del dispositivo e prima di sapere quali sensori sarebbero serviti quindi la tabella riportata contiene dati extra). La prima caratteristica sulla quale abbiamo puntato è la presenza di Wear Os. Wear Os è il sistema operativo "indossabile" di Google che porta Android anche all'interno degli smartwatch. Questa scelta è giustificata dal fatto che Wear Os sia "programmer friendly" ed inoltre esiste un ambiente di sviluppo integrato per lo sviluppo per la piattaforma Android ossia Android Studio.
Dopo una ricerca per trovare gli smartwatch più adatti allo scopo i candidati erano tre: Galaxy Watch 4, TicWatch Pro 3 e TicWatch E2. Tutti questi modelli montano Wear Os.
Le altre caratteristiche ricercate riguardano la parte sensoristica dello smartwatch, ossia sensore per la frequenza cardiaca, accelerometro, barometro, giroscopio ecc.
Particolare attenzione è stata posta anche su hardware e batteria per ragioni di velocità di risposta, fluidità durante il funzionamento e per riuscire a svolgere test su periodi di tempo prolungati. Infatti, essendo Wear Os è famoso per i consumi energetici elevati, una batteria capiente è l'ideale.
Per confrontare i modelli segue una tabella:

Modello	Galaxy Watch 4	TicWatch Pro 3	TicWatch E2
Processore	Exynos W920	Snapdragon 4100	Snapdragon 2100
Ram	1,5 GB	1 GB	0.512 GB
Batteria	461 mAh	577 mAh	415 mAh
Memoria Interna	16 GB	8 GB	4 GB
NFC	Presente	Presente	Assente
Freq. Cardiaca	Presente	Presente	Presente
Accelerometro	Presente	Presente	Presente
Girocsopio	Presente	Presente	Assente
Google Pay	Presente	Presente	Presente

Dalla tabella appare chiaro che le due alternative più valide sono Galaxy Watch 4 e TicWatch Pro 3 in quanto presentano tutte le caratteristiche richieste. La

differenza sostanziale tra i due modelli sta nel processore.
Segue un confronto più accurato tra i due processori

	Qualcomm Snapdragon Wear 4100	Samsung Exynos W920
	Qualcomm	Samsung
	ARM Cortex-A53	ARM Cortex-A55
Series: ARM Cortex-A53	**Qualcomm Snapdragon Wear 4100+** 0 - 1.7 GHz 5 / 5 ARM Cortex-A53 **» Qualcomm Snapdragon Wear 4100** 0 - 1.7 GHz 4 / 4 ARM Cortex-A53 **Samsung Exynos 9110** 1.15 GHz 2 / 2 ARM Cortex-A53	
	<=1700 MHz	1180 MHz
	4 / 4	2 / 2
	12	5
	LPDDR3-750 Memory Controller, eMMC	Mali-G68MP2, qHD 960x540 Display Support, LPDDR4 (12Gb), FHD 30fps video
iGPU	Qualcomm Adreno 504 (320 MHz)	ARM Mali-G68 MP2
Architecture	ARM	ARM
	= 447 days old	= 6 days old
	Qualcomm Wear 4100	

Figura A.1: Confronto Processori

Nonostante il numero di core e la frequenza del Samsung Exynos W920 siano inferiori, esso risulta il miglior processore tra i due grazie alla sua iGPU e alla sua RAM (DDR4 vs DDR3). Seguono dei benchmarks per mostrare effettivamente la differenza tra i due

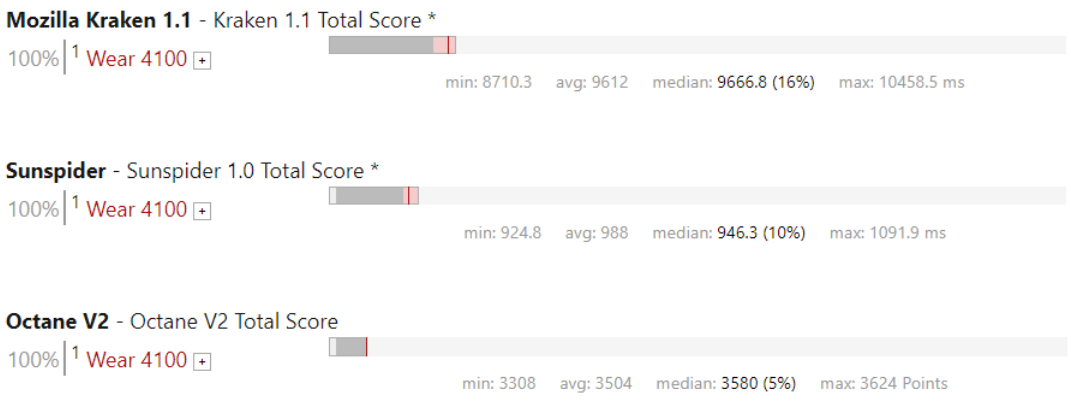


Figura A.2: Benchmarks Processori

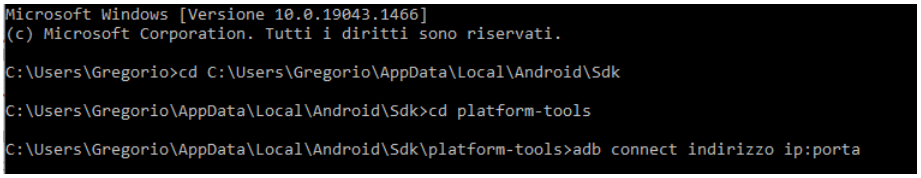
Per concludere Galaxy Watch 4 risulta essere lo smartwatch più adatto a questa tesi per caratteristiche e prestazioni.

Appendice B

How To: connettere Galaxy Watch 4 ad Android Studio

Solitamente per testare il codice su un dispositivo Android è necessario connettere quest ultimo ad Android Studio. Una volta connesso basta runnare il codice e modificarlo a piacimento per ottenere il risultato desiderato.
La maggior parte dei dispositivo si connettono tramite USB al PC e vengono rilevati in automatico da Android Studio. Nel caso del Galaxy Watch4 questo procedimento non è possibile in quanto l'USB che si trova nella scatola del dispositivo ha come altra estremità una base di ricarica magnetica. Quest'ultima non permette il passaggio dei dati.
Per ovviare a questo problema è necessario compiere le seguenti azioni:

- prendere il dispositivo ed abilitare la modalità sviluppatore
- connettere il dispositivo allo stesso Wi-Fi che utilizza il PC
- aprire da terminale l'SDK manager (se non si conosce il percorso basta andare su Android Studio – Tool – SDK)
- aprire platform-tool
- a questo punto per connettere il dispositivo basta usare il comando adb connect e inserire l'indirizzo IP e la corretta porta che viene displayata sul dispositivo (esempio adb connect 192.12.34.1:5555)
- una volta seguiti questi punti il dispositivo è connesso ad Adroid Studio



Appendice C

Codice Applicazione

```
1  package com.hfad.applicazionetesi;
2
3  import android.app.Activity;
4  import android.content.BroadcastReceiver;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.content.IntentFilter;
8  import android.os.Bundle;
9  import android.view.View;
10 import android.view.WindowManager;
11 import android.widget.TextView;
12
13 import androidx.core.content.ContextCompat;
14
15 import com.hfad.applicazionetesi.databinding.ActivityMainBinding;
16
17 public class MainActivity extends Activity {
18
19     private TextView stringa;
20     private ActivityMainBinding binding;
21     private static final String EXIT1 = "La sessione    aperta";
22     private static final String EXIT2 = "La sessione    terminata
    ";
23     private static final String REFRESH = "Check";
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         binding = ActivityMainBinding.inflate(getLayoutInflater()
    );
29         setContentView( R.layout.activity_main );
30         stringa = (TextView) findViewById(R.id.verbose);
31         getWindow().addFlags(WindowManager.LayoutParams.
    FLAG_KEEP_SCREEN_ON);
32         setActionFilters();
33         startService();
34     }
35
36     public void startService() {
```

```
37          Intent serviceIntent = new Intent(this, SensoriBackground
    .class);
38          startForegroundService(serviceIntent);
39      }
40      private final BroadcastReceiver myActionReceiver = new
    BroadcastReceiver(){
41          @Override
42          public void onReceive(Context context, Intent intent) {
43              switch(intent.getAction()) {
44                  case EXIT1:
45                      onCase1();// dopo aver refreshato so che la
    sessione    aperta
46                      break;
47                  case EXIT2:
48                      onCase2();// dopo aver refreshato la sessione
        chiusa e cancello il service
49                      break;
50              }
51          }
52      };
53
54      private void setActionFilters() {
55          IntentFilter filter = new IntentFilter();
56          filter.addAction(EXIT1);
57          filter.addAction(EXIT2);
58          registerReceiver(myActionReceiver, filter);
59      }
60
61      public void onCase1(){
62          stringa.setText("Refresh: sessione aperta");
63      }
64
65      public void onCase2(){
66          stringa.setText("Refresh: sessione chiusa");
67          Intent stop_service_intent = new Intent(this,
    SensoriBackground.class);
68          stopService(stop_service_intent);
69      }
70
71
72      public void onCLickRefresh(View view){
73          Intent refresh = new Intent();
74          refresh.setAction(REFRESH);
75          sendBroadcast(refresh);
76      }
77
78  }
```

Listing C.1: Main

```
1  package com.hfad.applicazionetesi;
2
3  import android.app.Notification;
4  import android.app.NotificationChannel;
5  import android.app.NotificationManager;
```

```
6  import android.app.PendingIntent;
7  import android.app.Service;
8  import android.content.BroadcastReceiver;
9  import android.content.Context;
10 import android.content.Intent;
11 import android.content.IntentFilter;
12 import android.hardware.Sensor;
13 import android.hardware.SensorEvent;
14 import android.hardware.SensorEventListener;
15 import android.hardware.SensorManager;
16 import android.os.Build;
17 import android.os.IBinder;
18
19 import androidx.core.app.NotificationCompat;
20
21
22 public class SensoriBackground extends Service implements
       SensorEventListener {
23
24
25     private static final String EXIT1 = "La sessione    aperta";
26     private static final String EXIT2 = "La sessione    terminata
       ";
27     private static final String REFRESH = "Check";
28     private SensorManager mSensorManager = null;
29     private Sensor mAccelerometer = null;
30     private SensorManager hSensorManager = null;
31     private Sensor hHeartBeat = null;
32     float[] mAccelerometerValues = null;
33     boolean flag = true;
34     int i = 0;
35     private String CHANNEL_ID = "NOTIFICATION_CHANNEL";
36
37     @Override
38     public void onCreate() {
39         super.onCreate();
40     }
41
42     @Override
43     public int onStartCommand(Intent intent, int flags, int
       startId) {
44         createNotificationChannel();
45         Intent pintent = new Intent(this, MainActivity.class);
46         PendingIntent pendingIntent = PendingIntent.getActivity(
       this,0,pintent,0);
47         Notification notification = new NotificationCompat.
       Builder(this,"ChannelID")
48                                         .setContentIntent(
       pendingIntent)
49                                         .build();
50         startForeground(1,notification);
51         setListenFilters();
52         prepareAccSensors();
53         prepareHSensor();
```

```
54          return START_STICKY;
55      }
56
57      private void createNotificationChannel() {
58          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
59              NotificationChannel serviceChannel = new
    NotificationChannel(
60                      "ChannelID", "foreground notification",
    NotificationManager.IMPORTANCE_DEFAULT);
61              NotificationManager manager = getSystemService(
    NotificationManager.class);
62              manager.createNotificationChannel(serviceChannel);
63          }
64      }
65
66      private void prepareAccSensors(){
67          mSensorManager = (SensorManager)getSystemService(
    SENSOR_SERVICE);
68          mAccelerometer = mSensorManager.getDefaultSensor(Sensor.
    TYPE_ACCELEROMETER);
69          mSensorManager.registerListener(this, mAccelerometer,
    SensorManager.SENSOR_DELAY_NORMAL);
70      }
71
72      private void prepareHSensor(){
73          hSensorManager = (SensorManager)this.getSystemService(
    Context.SENSOR_SERVICE);
74          hHeartBeat = hSensorManager.getDefaultSensor(Sensor.
    TYPE_HEART_RATE);
75      }
76
77      public void onSensorChanged(SensorEvent event){
78          if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
79              mAccelerometerValues = event.values;
80              int asse1 = (int) Math.abs(mAccelerometerValues[0]);
81              int asse2 = (int) Math.abs(mAccelerometerValues[1]);
82              int asse3 = (int) Math.abs(mAccelerometerValues[2]);
83              if(asse1>20 || asse2>20 || asse3>20){
84                  exitCode1();
85              }
86          }
87          else if (event.sensor.getType() == Sensor.TYPE_HEART_RATE
    ){
88              int valore = (int) event.values[0];
89              exitCode2(valore);
90          }
91      }
92
93      public void onAccuracyChanged(Sensor sensor, int accuracy) {
94      }
95
96      private void exitCode1(){
97          mSensorManager.unregisterListener(this);
98          hSensorManager.registerListener(this, hHeartBeat,
```

```
    SensorManager.SENSOR_DELAY_NORMAL);
99   }
100
101  private void exitCode2(int value){
102      if(value<5)
103      {
104          i++;
105      }
106      if(i==35)
107      {
108          hSensorManager.unregisterListener(this);
109          flag = false;
110      }
111      if(value>5)
112      {
113          hSensorManager.unregisterListener(this);
114          i = 0;
115          mSensorManager.registerListener(this, mAccelerometer,
     SensorManager.SENSOR_DELAY_NORMAL);
116      }
117  }
118
119  private final BroadcastReceiver myBroadcastReceiver = new
    BroadcastReceiver() {
120      @Override
121      public void onReceive(Context context, Intent intent) {
122          switch (intent.getAction()) {
123              case REFRESH:
124                  onCheck();// do task 1 request from Activity
125                  break;
126              default:
127                  // do nothing
128                  break;
129          }
130      }
131  };
132
133  public void onCheck(){
134      if(flag){
135          Intent exit1 = new Intent();
136          exit1.setAction(EXIT1);
137          sendBroadcast(exit1);
138      }
139      else{
140          Intent exit2 = new Intent();
141          exit2.setAction(EXIT2);
142          sendBroadcast(exit2);
143      }
144  }
145
146  private void setListenFilters() {
147      IntentFilter filter = new IntentFilter();
148      filter.addAction(REFRESH);
149      registerReceiver(myBroadcastReceiver, filter);
```

```
150     }
151
152     @Override
153     public void onDestroy() {
154         super.onDestroy();
155     }
156     @Override
157     public IBinder onBind(Intent intent) {
158         return null;
159     }
160 }
```

Listing C.2: Service

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/
       android"
3      package="com.hfad.applicazionetesi">
4
5      <uses-permission android:name="android.permission.WAKE_LOCK"
       />
6      <uses-feature android:name="android.hardware.type.watch" />
7      <uses-permission android:name="android.permission.
       BODY_SENSORS" />
8      <uses-permission android:name="android.permission.
       FOREGROUND_SERVICE"/>
9
10     <application
11         android:allowBackup="true"
12         android:icon="@mipmap/ic_launcher"
13         android:label="@string/app_name"
14         android:supportsRtl="true"
15         android:theme="@android:style/Theme.DeviceDefault">
16
17         <service
18             android:name=".SensoriBackground"
19             android:enabled="true"
20             android:exported="true"
21             android:foregroundServiceType="dataSync"></service>
22
23         <uses-library
24             android:name="com.google.android.wearable"
25             android:required="true" />
26         <!--
27                 Set to true if your app is Standalone, that is, it
       does not require the handheld
28                 app to run.
29         -->
30         <meta-data
31             android:name="com.google.android.wearable.standalone"
32             android:value="true" />
33
34         <activity
35             android:name=".MainActivity"
36             android:exported="true"
```

```
37              android:label="@string/app_name">
38              <intent-filter>
39                  <action android:name="android.intent.action.MAIN"
    />
40
41                  <category android:name="android.intent.category.
   LAUNCHER" />
42              </intent-filter>
43          </activity>
44      </application>
45
46  </manifest>
```

Listing C.3: Manifest

Bibliografia

[Aon+20] Simone Aonzo et al. «Low-Resource Footprint, Data-Driven Malware Detection on Android». In: *IEEE Transactions on Sustainable Computing* 5.2 (2020), pp. 213–222. DOI: 10.1109/TSUSC.2017.2774184.

[Avi+10] Adam J Aviv et al. «Smudge attacks on smartphone touch screens». In: *4th USENIX Workshop on Offensive Technologies (WOOT 10)*. 2010.

[BMI20] Andreea Barbu, Gheorghe Militaru e Savu Ionut. «Factors affecting the use of smartwatches». In: *FAIMA Bus. Manag. J* 5 (2020), p. 202044.

[GMM18] Meriem Guerar, Alessio Merlo e Mauro Migliardi. «Completely Automated Public Physical test to tell Computers and Humans Apart: A usability study on mobile devices». In: *Future Generation Computer Systems* 82 (2018), pp. 617–630. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.03.012. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17303709.

[Gue+20a] Meriem Guerar et al. «CirclePIN: A Novel Authentication Mechanism for Smartwatches to Prevent Unauthorized Access to IoT Devices». In: *ACM Trans. Cyber-Phys. Syst.* 4.3 (mar. 2020). ISSN: 2378-962X. DOI: 10.1145/3365995. URL: https://doi.org/10.1145/3365995.

[Gue+20b] Meriem Guerar et al. «Securing PIN-based authentication in smartwatches with just two gestures». In: *Concurrency and Computation: Practice and Experience* 32.18 (2020). e5549 cpe.5549, e5549. DOI: https://doi.org/10.1002/cpe.5549. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5549. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5549.

[Lu+17] Chris Xiaoxuan Lu et al. «VeriNet: User Verification on Smartwatches via Behavior Biometrics». In: *Proceedings of the First ACM Workshop on Mobile Crowdsensing Systems and Applications*. CrowdSenSys '17. Delft, Netherlands: Association for Computing Machinery, 2017, pp. 68–73. ISBN: 9781450355551. DOI: 10.1145/

3139243.3139251. URL: https://doi.org/10.1145/3139243.3139251.

[Lu+18] Chris Xiaoxuan Lu et al. «Snoopy: Sniffing Your Smartwatch Passwords via Deep Sequence Learning». In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1.4 (gen. 2018). DOI: 10.1145/3161196. URL: https://doi.org/10.1145/3161196.

[Nah16] Ani Nahapetian. «Side-channel attacks on mobile and wearable systems». In: *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2016, pp. 243–247. DOI: 10.1109/CCNC.2016.7444763.

[SB14] Elizabeth Stobert e Robert Biddle. «The password life cycle: user behaviour in managing passwords». In: *10th Symposium On Usable Privacy and Security (SOUPS 2014)*. 2014, pp. 243–255.

[UA16] Emmanuel Sebastian Udoh e Abdulwahab Alkharashi. «Privacy risk awareness and the behavior of smartwatch users: A case study of Indiana University students». In: *2016 Future Technologies Conference (FTC)*. 2016, pp. 926–931. DOI: 10.1109/FTC.2016.7821714.

[Wie+06] Susan Wiedenbeck et al. «Design and Evaluation of a Shoulder-Surfing Resistant Graphical Password Scheme». In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '06. Venezia, Italy: Association for Computing Machinery, 2006, pp. 177–184. ISBN: 1595933530. DOI: 10.1145/1133265.1133303. URL: https://doi.org/10.1145/1133265.1133303.