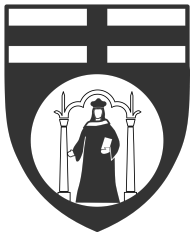
UNIVERSITÀ DEGLI STUDI DI GENOVA

Facoltà di Ingegneria



Corso di Laurea Magistrale in Ingegneria Informatica

TITOLO

HideDroid: App-level Runtime Data Anonymization on Mobile

Relatore:

Prof. Alessio Merlo

Correlatore:

Dott. Davide Caputo

Candidati:

Francesco Pagano
Matricola: 4192013

Giovanni Bottino
Matricola: 4249940

Anno Accademico 2019 – 2020

Indice

Acronimi

adb Android Debug Bridge

Ads Advertising

API Application Programming Interface

CA Certificate Authorities

DA Data Anonymization

DGH Domain Generalization Hierarchy

DP Differential Privacy

EI Explicit Identifiers

GDPR General Data Protection Regulation

GPS Global Positioning System

IMEI International Mobile Equipment Identity

IP Internet Protocol

NPT Not Perturbative Techniques

OS Operating System

PT Perturbative Techniques

QI Quasi-Identifiers

SD Sensitive Data

SDK Software Development Kit

UE Unione Europea

VPN Virtual Private Network

Elenco delle figure

Elenco delle tabelle

Elenco degli script

1 Introduzione

Nella prima metà del 2020, report statistici hanno evidenziato come l'utilizzo di smartphone da parte della popolazione mondiale sia in costante crescita[1]. Ad oggi, le persone che possiedono uno smartphone, sono circa 3.5 miliardi, cioè il 44.81% della popolazione mondiale. Questo numero è aumentato notevolmente rispetto al 2016, quando solamente 2.5 miliardi di persone ne possedevano uno[2]. Tuttavia, non aumenta solo il numero di persone che utilizzano uno smartphone, ma anche il numero di applicazioni disponibili negli store. Ad oggi, infatti, si contano circa 4.5 milioni di applicazioni di cui 2.56 M Android e 1.85 M iOS[3]. Nonostante ciò, per la prima volta negli ultimi anni, a partire dal 2018, il numero di applicazioni disponibili per i dispositivi Android si è ridotto[4] e le motivazioni di tale fenomeno sono principlamente due.

La prima è sicuramente legata al grande lavoro di controllo svolto da Google, che solo nell'ultimo anno[5] ha rimosso circa 1700 applicazioni dal Play Store, perchè malevoli o non conformi alle nuove norme sulla sicurezza e sulla privacy. La seconda, invece, è legata al crescente monopolio, sul mercato delle applicazioni mobile, da parte di grandi aziende come Google, Facebook, SuperCell, Ubisoft, che non lasciano spazio alla concorrenza, convogliando la quasi totalità dell'utenza verso i propri prodotti. Queste aziende, inoltre, hanno introdotto schemi pubblicitari e funzionalità in grado di adattarsi alle abitudini dell'utente, attraverso l'impiego di algoritmi che raccolgono ed elaborano dati personali, con l'obiettivo di personalizzare e migliorare costantemente l'esperienza utente. Queste funzionalità permettono agli sviluppatori di app di ricevere continui feedback sul modo in cui l'utenza interagisce con le loro piattaforme, al fine di soddisfare pienamente le aspettative degli utenti. Ad oggi, questi strumenti di raccolta e analisi dei dati, vengono messi a disposizione anche agli sviluppatori di applicazioni mobili, sotto forma di librerie ed Application Programming Interface (API) specifiche per la profilazione. Queste librerie, infatti, consentono di visualizzare, attraverso specifiche dashboard, informazioni riguardanti i comportamenti degli utenti, sotto forma di eventi (ad esempio, l'insieme delle pagine più visitate o la cronologia degli acquisti effettuati), ma anche informazioni personali sull'utente e sul dispositivo in uso (ad

[1] https://www.oberlo.com/statistics/how-many-people-have-smartphones
[2] https://www.bankmycell.com/blog/how-many-phones-are-in-the-world
[3] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
[4] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
[5] https://tecnologia.libero.it/perche-google-ha-eliminato-piu-di-1700-applicazioni-quali-sono-33240

esempio, il numero identificativo del dispositivo International Mobile Equipment Identity (IMEI), la versione di Operating System (OS) o la posizione Global Positioning System (GPS)). Attualmente, le librerie di analytics più utilizzate sono Facebook Analytics[6] e Google Firebase Analytics[7], secondo una classifica delle librerie di analytics più utilizzate [1]. Tuttavia, l'utilizzo di queste librerie ha sollevato diverse preoccupazioni in merito alla privacy dell'utente, per diversi motivi [2, 3]. Innanzitutto, queste librerie sono incorporate all'interno delle applicazioni, e quindi, condividono gli stessi privilegi e l'accesso alle stesse risorse, non garantendo nessun meccanismo di salvaguardia della privacy [3, 4, 5]. Infine, l'utente non ha alcun controllo sui dati raccolti da queste librerie: sebbene possa concedere o negare il permesso al raccoglimento dei propri dati personali, tuttavia, non può decidere quali dati vengano raccolti, né, tantomeno, applicare tecniche di anonimizzazione su di essi. Paradossalmente, questo significa che il compito di gestione delle informazioni personali viene affidato agli sviluppatori di app, piuttosto che all'utente stesso, il quale ne è l'unico legittimo proprietario. L'obiettivo degli sviluppatori e, principalmente delle aziende, è quello di massimizzare l'utilità dei dati a discapito della privacy dell'utente, al fine di ottenere informazioni sull'utilizzo dei loro prodotti, in modo da migliorare i loro servizi. Sebbene Google Play abbia recentemente introdotto alcune linee guida sulla privacy e sul trattamento dei dati personali, raccolti dagli sviluppatori di app, molte delle applicazioni, ancora presenti nello store, non sembrerebbero essersi adeguate correttamente. Infatti, in lavori precedenti [6] è stato evidenziato come oltre il 95% delle app analizzate acceda a dati sensibili dell'utente, ma solo un sottoinsieme trascurabile (1%) è pienamente conforme alle linee guida sulla privacy del Google Play Store. A tale scopo, in Europa, nel maggio del 2018, è entrato in vigore il General Data Protection Regulation (GDPR). Quest'ultimo sancisce, in materia di trattamento dei dati personali e di privacy, l'obiettivo di rafforzare la protezione dei dati personali dei cittadini dell'Unione Europea (UE), unificando e rendendo omogenea la normativa sulla privacy. Sosteniamo, quindi, che l'utente debba essere tenuto al corrente dei dati raccolti ed essere libero di scegliere il trade-off tra utilità e privacy dei propri dati, prima che questi vengano consegnati a terzi.

[6] https://analytics.facebook.com/
[7] https://firebase.google.com/docs/analytics

1.1 Contributo della tesi

Lo scopo della seguente tesi di ricerca, è quello di presentare una metodologia, chiamata **MobHide**, in grado di coinvolgere l'utente nello stabilire un livello di privacy dei propri dati e delle proprie informazioni, specifico per ogni app. Inoltre, per verificare la fattibilità, le performance e l'affidabilità dell'approccio proposto, si è implementato un prototipo della metodologia, in un app Android, chiamata **HideDroid**.
Per dimostrare l'efficacia, la fattibilità e le performance di HideDroid, abbiamo scelto di testarla sulle prime 2000 app più scaricate, presenti sul Google Play Store.

1.2 Struttura della tesi

La tesi viene organizzata come segue: dopo aver dato una breve introduzione sul lavoro che si intende svolgere, si definisce la metodologia sviluppata e gli strumenti utilizzati ai fini della sua implementazione, motivando le scelte sulla base dell'approccio presentato. Infine vengono esposti i risultati, le considerazioni finali ed eventuali sviluppi futuri. Nella Sezione 2 verranno introdotte alcune delle tematiche che si affronteranno nel corso dell'elaborato, facendo una panoramica sulle librerie di analytics, sulle tecniche di anonimizzazione e le recenti novità di sicurezza in Android. In Sezione 3 vengono descritti alcuni degli elaborati presenti all'attuale stato dell'arte nel campo della data privacy in ambito mobile. Nella Sezione 4 viene presentata la metodologia MobHide e nella Sezione 5 ne viene proposta un'implementazione, attraverso un'applicazione Android, chiamata HideDroid. Nella Sezione 6 vengono esposti i risultati, tratti dagli esperimenti effettuati sull'applicazione sviluppata e si discutono gli approcci implementativi e metodologici adottati. Infine, in Sezione 7, vengono riportate le conclusioni dell'elaborato sullo studio della metodologia proposta, delineando suoi possibili sviluppi futuri.

2 Background

2.1 Librerie di Analytics

Al fine di soddisfare pienamente le aspettative degli utenti, gli sviluppatori di app hanno la necessità di ricevere continui feedback sul modo in cui l'utenza interagisce con le loro piattaforme. Le **librerie di analytics** consentono di ottenere questi feedback, andando a registrare dati riguardanti l'utilizzo dell'app, informazioni sul dispositivo e sull'utente che la utilizza. In particolare, le librerie di analytics sono generalmente composte di due parti:

1. Software Development Kit (SDK) che può essere incluso dallo sviluppatore all'interno delle sue app. Una SDK, consiste in un insieme di strumenti tra cui API, ovvero librerie standard dotate di interfacce pubbliche, che consentono di interagire con i servizi offerti.

2. Un sistema di backend, che permette allo sviluppatore di poter accedere, in un secondo momento, ad informazioni statistiche di tipo grafico e/o testuale, ad esempio: l'utilizzo medio dell'applicazione da parte degli utenti, quanti utenti hanno installato l'applicazione, quanti invece l'hanno rimossa oppure la provenienza geografica dell'utenza. Il tutto tramite una dashboard di monitoraggio.

Come si nota in Figura 1, il dispositivo Android, che presenta al suo interno API di analytics (a), a run-time invia informazioni, attraverso la rete internet (b), al cloud dashboard dello sviluppatore (c), che è così in grado di monitorare l'utilizzo delle proprie piattaforme.

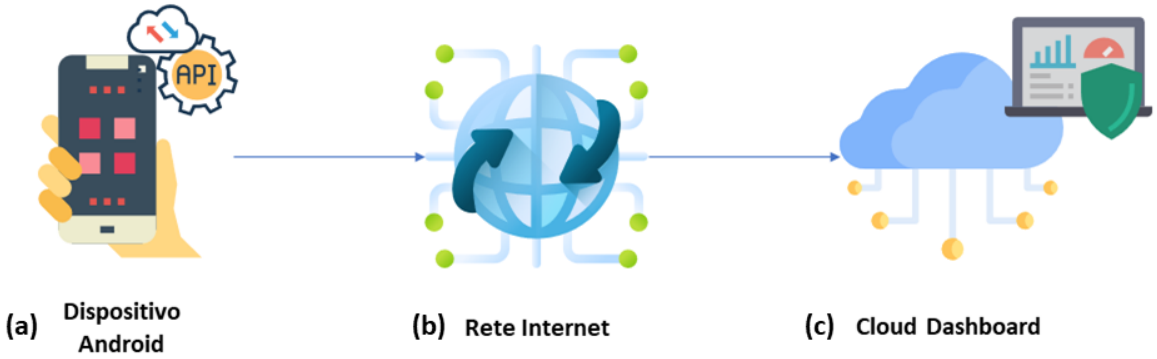


Figura 1: Esempio di architettura Android - API - Cloud

In Tabella 1 sono elencate le librerie di Analytics più utilizzate, secondo i dati statistici di Exodus [1].

Servizio	**Utilizzo complessivo (#app)**	**Utilizzo per API (#app)**	**API**
Google	134358	39728	Google Firebase Analytics
		36015	Google AdMob
		23300	Google CrashLytics
		19583	Google Analytics
		15732	Google Tag Manager
Facebook	64481	15732	Facebook Login
		14969	Facebook Share
		13978	Facebook Analytics
		10662	Facebook Ads
		9140	Facebook Places
Flurry	7697	7697	Flurry
Unity	6444	6444	Unity3d Ads
Moat	5495	5495	Moat
Twitter	4699	4699	Twitter MoPub

Tabella 1: Servizi di Analytics

Le librerie di analytics espongono API, che forniscono la possibilità di memorizzare informazioni sull'utente e/o sul dispositivo su cui è installata l'applicazione. Le interazioni dell'utente con le applicazioni, invece, vengono riportate sotto forma di **eventi**, che possono essere standard o custom a seconda delle scelte dello sviluppatore. Gli eventi standard, come *"app installing"* e *"app removing"*, *"app open"* e *"app close"*, sono comuni alla maggior parte dei servizi di analytics. Gli eventi custom, al contrario, vengono scelti dallo sviluppatore con uno scopo più preciso, sulla base anche dell'app di cui si vorrà monitorare l'andamento. Alcuni esempi possibili di *"custom events"* sono: *"add to cart"*, *"add payment info"* e *"purchase"* nel caso di applicazioni di *"e-commerce"*, oppure *"click to Ad"*, *"rewarded video"* nel caso di giochi mobile o app di apprendimento, che offrono ricompense a fronte della visione di spot pubblicitari.

2.2 Anonimizzazione dei Dati

Al fine di poter comprendere al meglio le tecniche di anonimizzazione, che verranno esposte nel corso della tesi, è opportuno soffermarsi su alcuni concetti chiave relativi ai

dati e alle loro caratteristiche. In **Data Anonymization (DA)**, gli eventuali attributi di un dato si possono, generalmente, dividere in:

- **Explicit Identifiers (EI)**, consentono di identificare univocamente un individuo a partire da informazioni quali, ad esempio, nome, cognome e codice fiscale.
- **Quasi-Identifiers (QI)**, permettono ad un ente terzo, in possesso di altre informazioni, di risalire all'identità di un individuo grazie ad informazioni quali, ad esempio, email, codice postale, numero di telefono, informazioni geografiche o demografiche.
- **Sensitive Data (SD)**, rappresentano tutte quelle informazioni come condizioni finanziare o sanitarie, che devono essere mantenute confidenziali e non ricollegabili all'individuo.

La tipologia di dati, con cui più frequentemente ci siamo confrontati durante il lavoro di ricerca, sono i **dati multidimensionali**. Un dato multidimensionale si presenta come una tabella, in cui ogni riga rappresenta un individuo, e ogni colonna rappresenta un'informazione associata a quell'individuo. Presa una riga, è possibile raggruppare un insieme di colonne per individuare i QI e i SD. Data la natura e varietà delle informazioni presenti, riga per riga, (un singolo attributo può presentarsi come un numero o una sequenza alfanumerica presa da un dominio specifico), è possibile costruire gerarchie sul singolo dato, variando la granularità dell'informazione portata da quest'ultimo. Data la categorizzazione delle informazioni in domini specifici, e alla varietà dei domini stessi, i dati multidimensionali sono una delle tipologie di dato più difficili da anonimizzare, poichè non esiste una funzione di trasformazione valida universalmente per tutti i domini. Questa loro proprietà rende impossibile stabilire a priori di quanta informazione sia in possesso un ipotetico terzo. Sebbene l'anonimizzazione di questo tipo di dato sia complicata, ad oggi esistono algoritmi di anonimizzazione molto promettenti, come la **K-Anonymity** [7].

2.2.1 Tecniche di anonimizzazione

Il processo di anonimizzazione ha come obiettivo quello di slegare l'identità dell'individuo dai suoi dati sensibili, ovvero gli EI e QI dai SD. Questo processo, affinchè sia valido, deve essere irreversibile, ovvero non deve essere possibile risalire all'identità della persona a partire dai suoi QI anonimizzati o dal loro arricchimento tramite

informazioni esterne. L'anonimizzazione può avvenire, generalmente, attraverso la rimozione, sostituzione e distorsione diretta degli EI come nome e codice fiscale, oppure attraverso la generalizzazione dei valori presenti all'interno di un attributo QI come data di nascita, codice postale o altro.
Un tale processo è di fondamentale importanza quando, per esempio, le aziende condividono i dati sui propri utenti con terze parti per effettuare indagini di mercato.
Ad oggi, le tecniche di DA possono essere suddivise, generalmente, in due macro gruppi: **Perturbative Techniques (PT)** e **Not Perturbative Techniques (NPT)**. Le prime consistono nell'aggiungere informazioni fittizie a ciascuno dei QI: un esempio può essere il rumore additivo su un QI numerico (es: *zip_code=16011* viene anonimizzato come *zip_code=16129* aggiungendo un rumore pari a *118*). Le tecniche NPT, al contrario, tentano di anonimizzare sfruttando metodologie atte alla generalizzazione degli attributi, alterando i dati in maniera accorta, al fine di preservare un livello di utilità maggiore rispetto alle PT.
Nel processo di anonimizzazione, il concetto chiave su cui occorre porre l'attenzione, è il trade-off tra **"Privacy-Utilità"**. Queste due misure, tra loro inversamente proporzionali, quantificano quanta informazione è stata oscurata dal dato e quanto il dato stesso, successivo all'anonimizzazione, sia utile in una sua fase successiva di analisi. Con il termine di **utilità** si intende la quantità numerica di interrogazioni eseguibili sul dato anonimizzato e sopprattutto la granularità delle interrogazioni stesse. Ad oggi, uno dei problemi degli algoritmi di DA è appunto la difficoltà nel selezionare un insieme di parametri algoritmici con cui bilanciare questo trade-off.
Da questo fatto, dunque, si deduce che le PT presentino problematiche sia dal punto di vista della privacy che dal punto di vista dell'utilità. Infatti, a livello di privacy, nelle PT, spesso è semplice, per un terzo, non in possesso di ulteriori informazioni, risalire al dato originale, tramite l'applicazione di un filtro contro il rumore inserito. Dal punto di vista dell'utilità del dato, nelle PT, il problema è dettato dal fatto che l'informazione viene così falsata, limitando quindi il range delle possibili interrogazioni.
Di seguito verranno introdotti i concetti chiave che caratterizzano due delle tecniche di anonimizzazione, che sono state impiegate all'interno della metodologia MobHide.

- **Generalizzazione**: è una delle tecniche NPT più utilizzate, specialmente per il trattamento di dati multidimensionali (ad es., database relazionali) e consiste nel rimpiazzare valori specifici di un insieme di attributi, appartenenti ad un

certo dominio, con alcuni più generici, preservando più semantica possibile. Nel dettaglio, dato un attributo A della tabella T, è possibile definire una gerarchia di generalizzazione del dominio (in inglese, **Domain Generalization Hierarchy (DGH)**) per A, come un insieme di n funzioni f_h: h = 0, . . . , n-1 tale che:

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} \ldots \xrightarrow{f_n - 1} A_n \tag{1}$$

Per esempio, in Figura 2, per un insieme Z_0 di codici postali (ZIP codes) possiamo definire una funzione di generalizzazione f_0 che sostituisca la prima cifra più a destra del codice, con il carattere *, rappresentando così zone geografiche più ampie. Per ridurre ulteriormente la quantità di informazioni dell'insieme Z_1, possiamo dunque iterare il procedimento tramite la definizione delle funzioni f_1, f_2, . . . , $f_n - 1$ fino al raggiungimento del dominio più generico Z_n, in cui tutti gli ZIP codes saranno mappati ad un unico valore. Ciò detto, si evince facilmente che, più funzioni di generalizzazione verranno invocate sul dato originale, maggiore sarà il livello di privacy ottenuto su quel dato, e di consenguenza, minore sarà la sua utilità, man mano che dati eterogenei si trasformeranno in un insieme sempre più ridotto di valori generici.

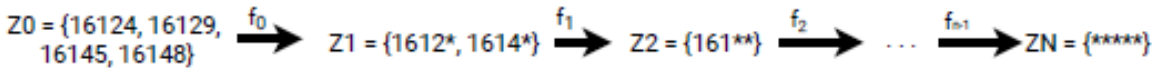


Figura 2: Un esempio di DGH per Codici Postali

Occorre precisare, però, che le tecniche di generalizzazione si adattano, per lo più, a dati multidimensionali semanticamente indipendenti, e poco, invece, all'anonimizzazione di sequenze di dati semanticamente correlate tra loro. Per questo motivo, mentre per l'anonimizzazione di attributi di singoli eventi verrà utilizzata la generalizzazione, per quanto riguarda le sequenze di eventi correlate fra loro ci si appoggerà a tecniche di **Differential Privacy (DP)**.

- **Differential Privacy**: La DP [8] è un modello di anonimizzazione, che consiste nell'alterazione della distribuzione originale di un insieme di dati correlati, perturbandoli mediante rumore randomico. Questo modello viene, solitamente,

applicato in un contesto dove è necessaria la riservatezza dell'informazione scambiata tra diversi pari, e dove l'identità dei destinatari, a cui vengono inviati tali dati, non è nota a priori. Questo contesto viene chiamato **locale** e differisce da un contesto **centralizzato** in cui, invece, i destinatari delle informazioni sono fidati ed essi stessi si occupano, tra le altre cose, dell'anonimizzazione dei dati prima di inviarli a terzi. In un contesto non fidato (locale), l'anonimizzazione dei dati deve essere fatta localmente, cioè il client stesso è responsabile dell'anonimizzazione dei dati prima di inviarli a terzi. L'insieme dei dati in esame, si presenta come una sequenza ordinata di eventi; l'obbiettivo della DP è quello di trasformare tale sequenza di eventi $(e_1, e_2, \dots e_n)$ in un'altra sequenza di eventi $(z_1, z_2, \dots z_n)$ mediante l'applicazione, evento per evento, di una specifica funzione di perturbazione (R), generalmente una distribuzione di probabilità, definita a priori: $z_i = R(e_i)$.

2.3 Sicurezza di Rete in Android

Una delle caratteristiche di sicurezza fondamentali in Android è la gestione delle comunicazioni di rete. Ogni applicazione si interfaccia verso API e server esterni per poter svolgere le sue funzioni. Mentre ad oggi la quasi totalità delle comunicazioni web, client-server, avviene mediante protocollo **HTTPS** [9], in Android, invece, è stato verificato [10], nel 2019, che su 125419 applicazioni analizzate, solo 16332 app adottavano questa funzionalità di sicurezza. HTTPS garantisce la confidenzialità del contenuto di richieste e risposte, scambiate tra il client e il server, utilizzando un canale, che di default non è sicuro: internet. Tale meccanismo si basa sulla ricezione, da parte del client (l'app), di un certificato, che attesti l'identità del server con cui sta entrando in contatto, in modo da verificare che le richieste vengano effettivamente inviate verso il destinatario previsto e che il loro contenuto rimanga confidenziale e accessibile solo ai diretti interessati, e quindi, non leggibile da terzi. Da quanto esposto, si evince che la validazione dei certificati ricevuti è il punto cardine di una comunicazione HTTPS. Le app Android, così come tutti i browser o client che comunicano utilizzando protocollo HTTPS, riconoscono un certificato come valido, solo se questo è stato precedentemente memorizzato sul dispositivo.

Al fine di forzare l'utilizzo di HTTPS, a partire da Android Nougat[8] (API 24), è stato

[8] https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html

modificato il modo in cui Android gestisce le **Certificate Authorities (CA)** attendibili [11], per fornire impostazioni predefinite più sicure per il traffico protetto delle app. Dall'API 24 di Android, le applicazioni installate su tali dispositivi non accettano più i certificati utente, a meno che lo sviluppatore dell'app non specifichi il contrario. Per consentire alle applicazioni di definire una propria configurazione di rete, in maniera sicura ed esplicativa, in Android, è stata introdotta, quindi, la funzionalità **Network Security Configuration**[9]. Tale funzionalità viene implementata tramite la definizione di un file di configurazione, evitando la modifica del codice sorgente dell'app. Tali configurazioni possono essere specifiche per singoli domini o applicazioni.
Le funzionalità chiave della Network Security Configuration sono:

- **Custom Trust Anchors**: permette allo sviluppatore di specificare quali certificati sono accettabili dall'applicazione per instaurare una connessione sicura; ad esempio, certificati **autofirmati (self-signed)**, o restringere l'insieme dei certificati accettabili.

- **Debug-only overrides**: consente allo sviluppatore di specificare quali connessioni accettare solo in fase di debug dell'applicazione.

- **Cleartext traffic opt-out**: questa opzione, a partire da Android 9[10], è stata disabilitata di default impedendo alle applicazioni di instaurare comunicazioni in chiaro (HTTP) con i destinatari.

- **Certificate Pinning**: normalmente un'app accetta tutti i certificati preinstallati sul dispositivo. Con questa opzione lo sviluppatore è in grado di restringere l'insieme dei certificati validi, ai soli necessari per il corretto funzionamento dell'app.

[9] https://developer.android.com/training/articles/security-config
[10] https://developer.android.com/training/articles/security-config

3 Stato dell'Arte

L'ampia adozione di librerie di analytics, da parte degli sviluppatori di app, ha suscitato, recentemente, l'attenzione della grande comunità di ricerca sulla sicurezza. Uno dei primi scandali che ha portato alla luce il problema della privacy in Internet, è stato lo scandalo relativo a *Facebook-Cambridge Analytica*[11]. Questo ha rappresentato uno dei maggiori scandali avvenuti all'inizio del 2018, quando fu rivelato che Cambridge Analytica aveva raccolto i dati personali di milioni di account Facebook senza il loro consenso e li aveva usati per scopi di propaganda politica.

Negli ultimi anni diversi lavori scientifici, e non, hanno analizzato e approfondito il tema della privacy sui dispositivi mobile. Il lavoro svolto da Chen et al. [4] è uno dei primi studi che si focalizza sui problemi della privacy, legati all'utilizzo di librerie di analytics su dispositivi mobile. Nel dettaglio, gli autori hanno dimostrato come un utente esterno, malintenzionato, possa estrarre informazioni sensibili riguardanti l'utente e il suo utilizzo delle applicazioni, sfruttando due servizi mobile di analytics, ovvero *Google Mobile App Analytics* e *Flurry*.

Vallina et al. [12] hanno identificato e mappato i domini di rete associati a servizi di *Advertising (Ads)* mobile e le librerie volte al tracciamento degli utenti, attraverso un ampio studio delle app Android più popolari. Oltre all'analisi del problema, vi sono diverse attività di ricerca che si focalizzano nel proporre nuovi approcci per garantire la privacy. Ad esempio, Beresford et al. [13] hanno proposto una versione modificata del sistema operativo Android, chiamata **MockDroid**, che impedisce alle app di accedere a risorse di sistema. MockDroid permette, così, all'utente di revocare i permessi di accesso delle app a specifiche risorse, a run-time, incoraggiandolo a tenere in considerazione il trade-off tra le funzionalità delle applicazioni e la divulgazione di informazioni personali.

Zhang et al. [14] hanno proposto **PrivAid**, una metodologia per applicare tecniche di DP sugli eventi generati dall'utente, collezionati dalle app mobile. Il tool sostituisce le originali API di analytics con una implementazione custom, che colleziona gli eventi generati e applica tecniche di DP. La strategia di anonimizzazione viene configurata direttamente dallo sviluppatore, che può così ricostruire, almeno con una buona aprossimazione, la distribuzione degli eventi originali.

Gli autori in [15] propongono un'app Android, chiamata **Lumen Privacy Monitor**,

[11] https://en.wikipedia.org/wiki/Facebook–Cambridge_Analytica_data_scandal

in grado di analizzare il traffico di rete dei dipositivi mobile. Questa, permette di avvisare l'utente se un'app colleziona e invia informazioni personali (ad es., IMEI, MAC, Phone Number), consentendogli, così, di bloccare le richieste indirizzate ad uno specifico endpoint. Per fare ciò, Lumen Privacy Monitor, richiede tutti i permessi Android al fine di raccogliere i dati utente ed eseguire la ricerca nelle richieste di rete che vengono inviate. Sfortunatamente, le soluzioni sopra citate non forniscono una adeguata anonimizzazione dei dati, proponendo così strategie o approcci **block-or-allow**, che bloccano richieste contenenti dati personali e del dispositivo utilizzato dall'utente, eliminando totalmente qualunque informazione utile agli sviluppatori per poter migliorare la qualità dei propri servizi. Inoltre, gran parte di esse richiedono modifiche invasive delle app stesse o dell'OS (ad es., custom OS e permessi root), e ciò difficilmente può essere adottato, poichè, mediamente, gli utenti non abilitano permessi specifici sul proprio dispositivo. Al meglio delle nostre conoscienze, HideDroid è la prima proposta che permetta all'utente di scegliere un livello di privacy, specifico per ogni applicazione, e, allo stesso tempo, garantisca la possibilità di esportare dati anonimizzati. Infine, il nostro prototipo è stato realizzato per garantire la minima invasività sul device, in modo da consentire a chiunque di poterlo utilizzare, senza la necessità di effettuare modifiche all'OS del dispositivo o ai permessi a cui esso ha accesso.

4 Metodologia

In questo capitolo verrà presentata la metodologia **MobHide**, raffigurata in Figura 3. I dati generati dalle varie applicazioni, installate sullo smartphone, vengono analizzati dal **Privacy Detector** (step 1), il cui compito è quello di riconoscere, secondo una determinata euristica, se una richiesta o flusso di dati appartiene a librerie di analytics. In caso affermativo, i dati della richiesta in questione verranno memorizzati in una tabella all'interno dell'**Event Buffer** (step 2) e mandati contemporaneamente al **Data Anonymizer** (step 3), che si occuperà prima della loro anonimizzazione (step 6,7,8,9), e successivamente del loro invio al server di analytics di destinazione, appoggiandosi al **Data Sender** (step 10,11). In caso contrario, invece, i dati verranno veicolati direttamente ai previsti destinatari, senza alcuna loro modifica (step 5). Nel prosieguo di questo capitolo verranno analizzati i vari moduli presenti all'interno della metodologia.

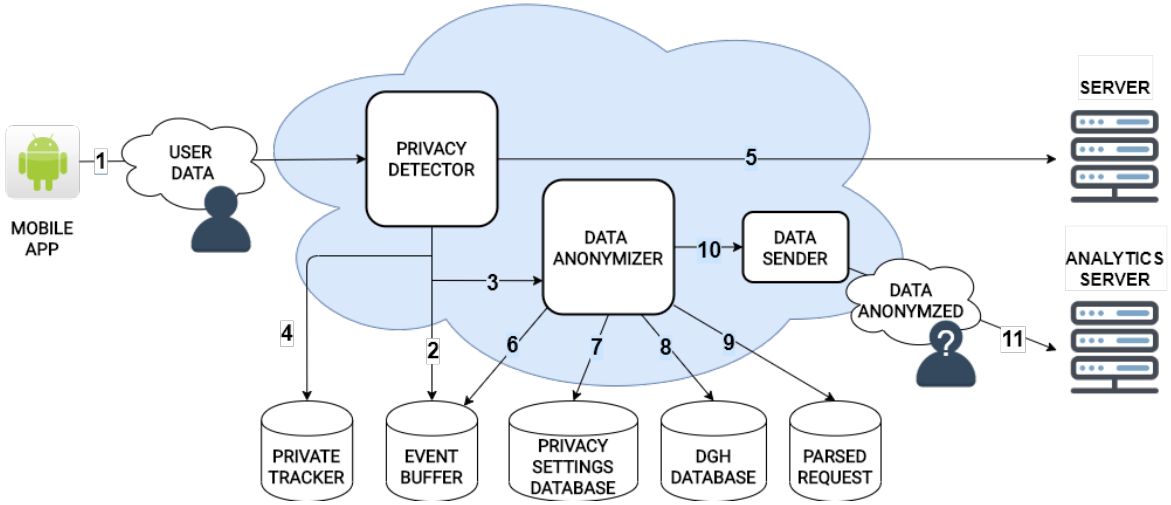


Figura 3: Metodologia MobHide

4.1 Privacy Detector

Il **Privacy Detector** ha il compito di analizzare e filtrare tutto il traffico proveniente dalle applicazioni installate sullo smartphone dell'utente (step 1). Il modulo esamina il traffico di rete con l'obiettivo di stabilire se una richiesta proviene o meno da librerie di analytics, basandosi sull'identità del destinatario (il suo dominio) e/o sul contenuto stesso della richiesta. Nel dettaglio, se il destinatario della richiesta appartiene ad una lista predefinita di host di analytics ben noti (ad es., *graph.facebook.com* di Facebook Analytics o *app-measurement.com* di Google Firebase Analytics), la richiesta viene

immediatamente memorizzata all'interno dell'Event Buffer Database (step 2) e inviata, parallelamente, al modulo Data Anonymizer (step 3) per poter essere anonimizzati. Se il dominio della richiesta non è conosciuto a priori, il Privacy Detector analizza i campi presenti all'interno della richiesta, con l'obiettivo di verificare la presenza o meno di attributi e parole chiave riconducibili a librerie di analytics (ad es., *device-name*, *device-id*, *device-info*, *event-name*, etc.). Se viene identificata almeno una di queste parole chiave, il Privacy Detector memorizza l'host in un'apposita tabella, denominata Private Tracker, (step 4), in modo tale da velocizzare il processo per le successive richieste. Il traffico non appartenente alle richieste di analytics viene inoltrato ai previsti destinatari (step 5).

4.2 Private Tracker

Il **Private Tracker** è un database, contenente l'insieme degli host di analytics, individuati dal Privacy Detector, a partire da un'analisi del contenuto di una richiesta (step 4).

4.3 Event Buffer

L'**Event Buffer** è un database, che mantiene al suo interno, le richieste intercettate dal Privacy Detector (step 2). Le richieste sono organizzate in una tabella e sono divise per applicazione (il client che le ha generate) e dominio del server di destinazione (il server della libreria di analytics).

4.4 Privacy Settings Database

All'interno del **Privacy Settings Database** vengono memorizzati i livelli di privacy, selezionati dall'utente, per ciascuna delle applicazione di cui si vuole ottenere un maggior livello di privacy. Ciascuna entry della tabella rappresenta, quindi, una coppia **application - privacy level**, permettendo così all'utente di definire con maggior granularità, strategie di anonimizzazione per ogni diversa applicazione. I livelli di privacy, selezionabili dall'utente, sono: **NONE, LOW, MEDIUM, HIGH**. Se il livello di anonimizzazione per un'app è impostato su NONE, il suo traffico di rete verrà escluso dal processo di anonimizzazione; viceversa, se il livello di anonimizzazione è impostato su HIGH, le strategie di generalizzazione e di DP anonimizzeranno i dati secondo le più

ristrettive regole, dando maggior peso alla privacy dell'utente piuttosto che all'utilità del dato.

4.5 DGH Database

La metodologia MobHide, nell'applicare le tecniche di anonimizzazione, basate su generalizzazione, si appoggia ad un terzo database, chiamato **Domain Generalization Hierarchy (DGH)**, che contiene regole di generalizzazione, specifiche per ogni dominio. All'interno della seguente tabella, per ciascuno degli attributi più comuni, individuati tra le librerie di analytics (ad es., *carrier*, *sdk*, *os*, *platform*, *system-id*, *model*, ...), sono state memorizzate gerarchie di generalizzazione specifiche per ogni livello. Inoltre, per alcuni dei servizi di analytics (come, ad esempio, *graph.facebook.com* di Facebook), sono state definite regole specifiche a parte. Al momento dell'anonimizzazione, il modulo Data Anonymizer verificherà la presenza di ciascun attributo della richiesta all'interno del DGH (step 8) e in caso affermativo, applicherà le regole di generalizzazione specifiche per quell'attributo, sulla base del livello di privacy selezionato; in caso contrario, invece, il Data Anonymizer applicherà un algoritmo di generalizzazione basato su euristiche, illustrato in Sezione 4.7.1.

4.6 Parsed Request

La struttura dati **Parsed Request** memorizza, al suo interno, le informazioni sulla struttura delle richieste di analytics (ad esempio: headers, body, host, ...), estratte attraverso il modulo Data Anonymizer (step 9). Questa struttura dati viene utilizzata come supporto nelle fasi di anonimizzazione e codifica della richiesta.

4.7 Data Anonymizer

Il **Data Anonymizer** è il cuore della metodologia MobHide. Questo modulo ha l'obiettivo di applicare strategie di anonimizzazione sui dati memorizzati dal Privacy Detector all'interno dell'Event Buffer (step 2). Le richieste vengono analizzate con l'obiettivo di riorganizzare i dati in apposite strutture dati, all'interno della tabella Parsed Request (step 9), utili alla fase di anonimizzazione. Come descritto nella Sezione 2.1, i dati, collezionati dalle librerie di analytics, includono sia le azioni svolte dall'utente sulle app (user app events), sia informazioni riguardanti l'utente stesso e il dispositivo

utilizzato. Per anonimizzare dati così eterogenei, il Data Anonymizer utilizza tecniche di generalizzazione, con l'appoggio di un DGH (step 8) e tecniche di DP. Entrambe le tecniche sono parametrizzate in base al livello di privacy selezionato dall'utente per una determinata applicazione (step 7). Al termine dell'anonimizzazione, i dati vengono codificati secondo la loro codifica originale e inoltrati al Data Sender (step 10).

4.7.1 Generalizzazione

Per anonimizzare le informazioni riguardanti l'utente e il dispositivo, il Data Anonymizer adotta tecniche basate sulla **generalizzazione** [16]. Il Data Anonymizer analizza ciascuna richiesta di rete per individuare ed estrarre tutti i dati al suo interno. Per ogni attributo, il modulo cerca, all'interno del DGH database, una regola di generalizzazione. Le regole di generalizzazione, che costituiscono il DGH, sono state individuate su base empirica per quegli attributi per cui era possibile definire tre diverse funzioni crescenti di generalizzazione, ciascuna per ogni livello di privacy selezionabile dall'utente. Se per un determinato attributo è stato possibile individuare una regola all'interno del DGH, il Data Anonymizer procederà nell'anonimizzare il valore secondo il livello di privacy selezionato dall'utente. Vicersa, se non è stata individuata alcuna regola, il modulo di Data Anonymizer si affiderà alla seguente euristica:

- se l'attributo è una **sequenza di caratteri alfanumerici**, la generalizzazione sostituisce gli ultimi p caratteri, con il carattere *'*'*. Il valore di p è dipendente dal livello di privacy selezionato ed è definito come segue:

$$p = \frac{stringLength * selectedPrivacyLevel}{\#PrivacyLevels - 1} \tag{2}$$

  dove *stringLength* è la lunghezza della sequenza di caratteri alfanumerici, $\#PrivacyLevels$ è il numero di livelli di privacy disponibili (ovvero 4 nella nostra implementazione) e *selectedPrivacyLevel* è il livello di privacy selezionato dall'utente (NONE=0, LOW=1, MEDIUM=2, HIGH=3). Se la sequenza di caratteri dovesse contenere solo caratteri numerici, la generalizzazione verrebbe applicata, comunque, secondo le regole illustrate nel successivo punto.

- se l'attributo è un **numero**, la generalizzazione arrotonda il valore alla cifra decimale p più significativa. Il valore di p è dipendente dal livello di privacy

selezionato ed è definito come segue:

$$p = \frac{\#digits * selectedPrivacyLevel}{\#PrivacyLevels - 1} \tag{3}$$

dove *#digits* è il numero di cifre che compongono il valore, mentre gli altri parametri sono definiti in maniera analoga come descritto precedentemente.

4.7.2 Differential Privacy

Le informazioni riguardanti le interazioni dell'utente con un'applicazione, vengono modellate come sequenze di eventi correlati fra loro. Per anonimizzare tali informazioni, il Data Anonymizer si affida all'utilizzo di euristiche basate sulla **DP** e al concetto di perturbazione locale dei dati. Questa euristica consente di anonimizzare le azioni dell'utente, preservando i dati strutturati per lo sviluppatore. Il processo di perturbazione locale dei dati, mira a modificare la distribuzione originale degli eventi **rimuovendo** quelli intercettati, **sostituendoli** o **inserendone** di nuovi fittizi. Per fare questo, il Data Anonymizer si affida ad un valore di soglia (*threashold*) definito come segue:

$$Threashold_{action} = 1 - \frac{selectedPrivacyLevel}{\#action + 1} \tag{4}$$

dove

$$action \in [inserimento, rimozione, sostituzione] \tag{5}$$

Il Data Anonymizer assegna, a ciascuno degli eventi intercettati, tre numeri pseudo-randomici (tra 0 e 1), che rappresentano la probabilità di esecuzione di ciascuna delle tre azioni di perturbazione (ovvero: inserimento, rimozione, sostituzione). Quindi, l'azione di perturbazione viene eseguita solo se la corrispondente probabilità è maggiore del valore di *threashold*.

4.7.3 Flusso di Esecuzione dell'Anonimizzazione

La procedura completa per l'anonimizzazione dei dati segue l'algoritmo descritto in Algorithm 1. Tale algoritmo viene utilizzato dal modulo Data Anonymizer per l'anonimizzazione delle richieste intercettate. L'algoritmo prende in input quattro parametri: *interceptedEvent*, ovvero l'evento da anonimizzare, *selectedPrivacyLevel*, ossia il livello di privacy selezionato dall'utente per quella specifica applicazione, l'*eventBuffer*, il

database contenente tutte le richieste intercettate dal Privacy Detector e *minNumberOfRequests*, un parametro scelto a priori da noi. Quest'ultimo rappresenta un valore di soglia che deve essere rispettato affinchè si possano eseguire correttamente le operazioni di inserimento e sostituzione. Successivamente l'algoritmo estrae le seguenti informazioni: l'*appName* (il nome dell'applicazione che ha generato tale richiesta), l'*hostName* (l'host a cui la richiesta è indirizzata) e infine *requestsWithSameAppAndHost* (l'insieme delle richieste intercettate tra i due end-point fino ad allora) (righe 4-6). Successivamente si procede al calcolo di tre numeri pseudo randomici: Pr_{inj}, Pr_{rem}, Pr_{rep} (righe 7-9). Se Pr_{inj} è maggiore del valore di $Threshold_{action}$ e il numero di richieste $requestsWithSameAppAndHost$ è maggiore del numero minimo di richieste *minNumberOfRequests* (righe 10-11), allora si procede all'inserimento di una nuova richiesta, presa randomicamente (sempre tra i due stessi end-point) dall'*eventBuffer*, che verrà subito anonimizzata (righe 12-13). Se Pr_{rep} è maggiore del valore di $Threshold_{action}$ e il numero di richieste $requestsWithSameAppAndHost$ è maggiore del numero minimo di richieste scelto da noi *minNumberOfRequests* (righe 16-17), allora il *Data Anonymizer* procederà alla sostituzione della richiesta corrente, con una presa randomicamente (sempre tra i due stessi end-point) dall'*eventBuffer*, che verrà subito anonimizzata (righe 18-21). Se, invece, Pr_{rem} è maggiore del valore di $Threshold_{action}$, allora si prosegue nella semplice rimozione della richiesta corrente (righe 24-25). Infine, se la richiesta corrente non è stata né sostituita né rimossa, allora questa viene semplicemente anonimizzata con le tecniche descritte precedentemente (righe 27-28). Al termine dell'esecuzione, l'algoritmo restituirà una lista di richieste (*anonymizedEvents*) che potrà risultare vuota o contenere una o più richieste anonimizzate.

Algorithm 1 Data Anonymization Pipeline

Input: $interceptedEvent$, $selectedPrivacyLevel$, $eventBuffer$, $minNumberOfRequests$
Output: $anonymizedEvents$

Initialize $anonymizedEvents \leftarrow list()$
Initialize $Threshold_{action} \leftarrow 1 - (selectedPrivacyLevel/4)$
Initialize $minNumberOfRequest \leftarrow 5$
$appName \leftarrow interceptedEvent.appName$
$hostName \leftarrow interceptedEvent.hostName$
$requestsWithSameAppAndHost \leftarrow eventBuffer.extractReqEndPoint(appName, hostName)$
$Pr_{inj} \leftarrow$ rand()
$Pr_{rem} \leftarrow$ rand()
$Pr_{rep} \leftarrow$ rand()
if $Pr_{inj} > Threshold_{action}$ **then**
    if $\#requestsWithSameAppAndHost \geq minNumberOfRequest$ **then**
        $newGenEvent \leftarrow$ generateNewGenEvent($selectedPrivacyLevel$)
        $anonymizedEvents$.add($newGenEvent$)
    end if
end if
if $Pr_{rep} > Threshold_{action}$ **then**
    if $\#requestsWithSameAppAndHost \geq minNumberOfRequest$ **then**
        $replEvent \leftarrow$ replaceEvent($interceptedEvent$)
        $replGenEvent.attributes \leftarrow$ generalizeEvent($replEvent.attributes$, $selectedPrivacyLevel$)
        $anonymizedEvents$.add($replGenEvent$)
        return $anonymizedEvents$
    end if
else if $Pr_{rem} > Threshold_{action}$ **then**
    deleteEvent($interceptedEvent$)
    return $anonymizedEvents$
end if
$originalGenEvent \leftarrow$generalizeEvent($interceptedEvent.attributes, selectedPrivacyLevel$)
$anonymizedEvents$.add($originalGenEvent$)
return $anonymizedEvents$

4.8 Data Sender

Il passo conclusivo della metodologia MobHide, qui proposta, si compie con l'inoltro dei dati ai previsti destinatari. L'incaricato di questa operazione è il modulo **Data Sender**, che vedremo in seguito, a livello implementativo, essere stato accorpato al modulo Data Anonymizer. Una volta che i dati sono stati anonimizzati, secondo le strategie previste dal Data Anonymizer, questi vengono incapsulati in pacchetti pronti ad essere inviati ai servizi di backend di analytics (step 10,11), simulando così le richieste orginali.

5 Implementazione

Per valutare la fattibilità e l'affidabilità della soluzione proposta, si è deciso di implementare la metodologia MobHide in un app per Android, chiamata **HideDroid**. L'app sviluppata risulta compatibile con tutte le versioni di Android a partire da Android Marshmallow [17], fino alle più recenti come Android 10 [18]. Il funzionamento dell'applicazione, raffigurato in Figura 4, è diviso in tre fasi: **Setup di HideDroid**, **Configurazione del livello di privacy per App** e **Anonimizzazione dei Dati**.

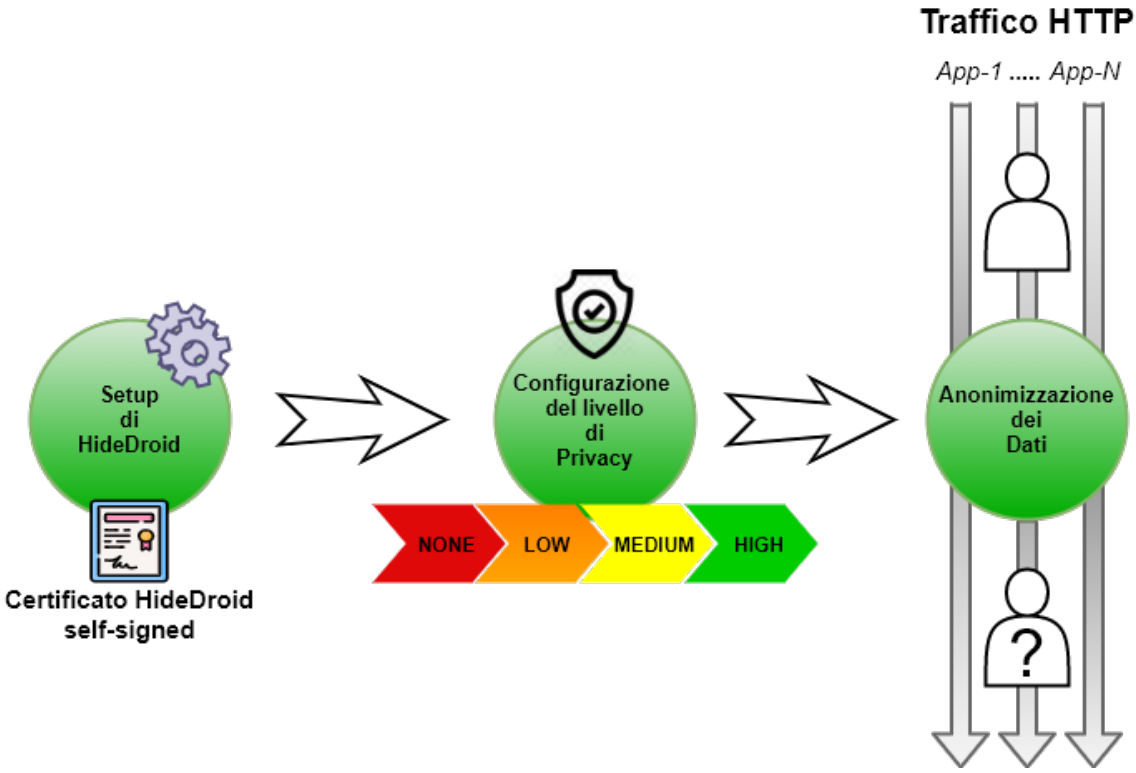


Figura 4: Le tre fasi in HideDroid

- **Setup di HideDroid.** Durante questa fase, HideDroid richiede, all'utente, il permesso di accedere allo storage del dispositivo e di installare il proprio certificato self-signed, nello store dei certificati utente. Terminati questi passaggi di configurazione, il servizio di anonimizzazione di HideDroid è, quindi, in grado di funzionare correttamente.

- **Configurazione del livello di privacy per App.** Dopo aver effettuato il setup di HideDroid, l'utente ha la possibilità di visionare la lista delle applicazioni installate sul dispositivo. Per ogni applicazione l'utente può scegliere uno tra i

seguenti livelli di privacy: NONE, LOW, MEDIUM e HIGH. Una volta selezionata l'applicazione e il livello di privacy, HideDroid verifica se siano necessari ulteriori operazioni di configurazione affinchè i dati possano essere intercettati e quindi successivamente anonimizzati. L'operazione di configurazione aggiuntiva eseguita da HideDroid è una procedura chiamata **repackaging**, che verrà descritta successivamente in questo capitolo.

- **Anonimizzazione dei Dati.** Questa fase viene effettuata quando l'utente utilizza una delle applicazioni selezionate nella fase precedente. Una volta attivato il servizio di HideDroid, denominato **Incognito Mode**, tutte le richieste dirette a servizi di analytics, provenienti dalle app selezionate, vengono anonimizzate e successivamente inoltrate al server di destinazione.

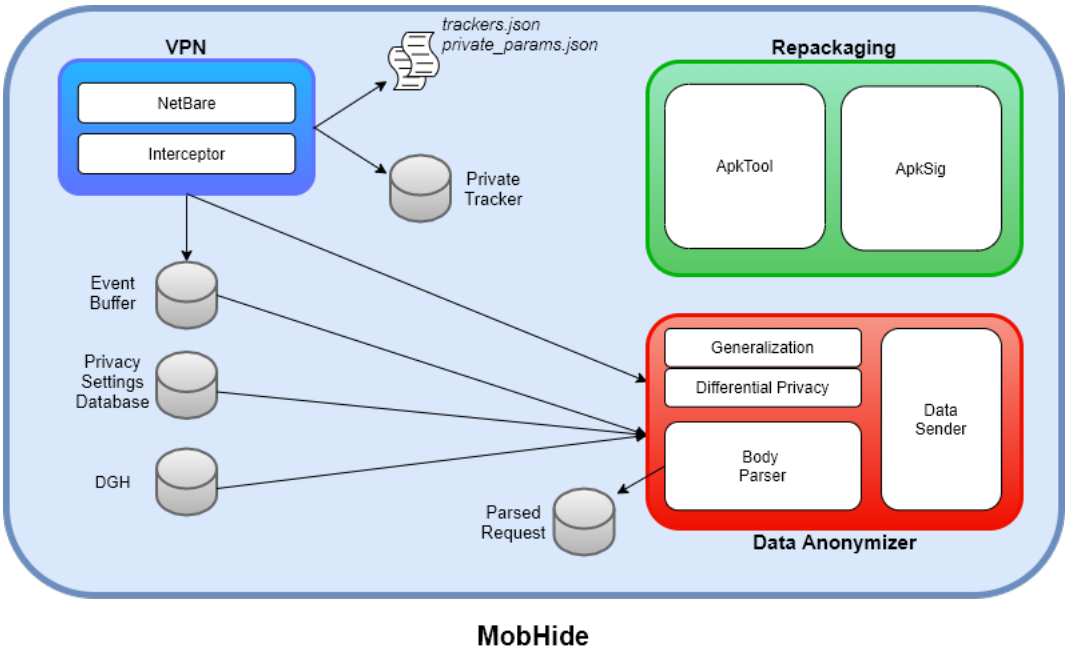


Figura 5: Moduli di implementazione in HideDroid

In questo capitolo, dunque, facendo riferimento allo schema di Figura 5, verranno analizzate e illustrate le scelte implementative fatte e i componenti che determinano il corretto funzionamento di HideDroid.

5.1 Setup di HideDroid

5.1.1 Certificato di HideDroid

Al primo avvio dell'applicazione, l'utente deve effettuare alcune operazioni di inizializzazione dell'app HideDroid necessarie al suo corretto funzionamento. Il permesso di accedere allo storage dei dati del dispositivo, Figura 6 (step (a)), risulta fondamentale per consentire ad HideDroid di memorizzare le informazioni sulle applicazioni da anonimizzare e per consentire la memorizzazione dei dati raccolti da HideDroid durante il suo utilizzo. Successivamente viene chiesto il permesso per generare e memorizzare un certificato self-signed. Questo certificato è necessario ad HideDroid per poter monitorare e analizzare il traffico di rete generato dalle applicazioni selezionate dell'utente.

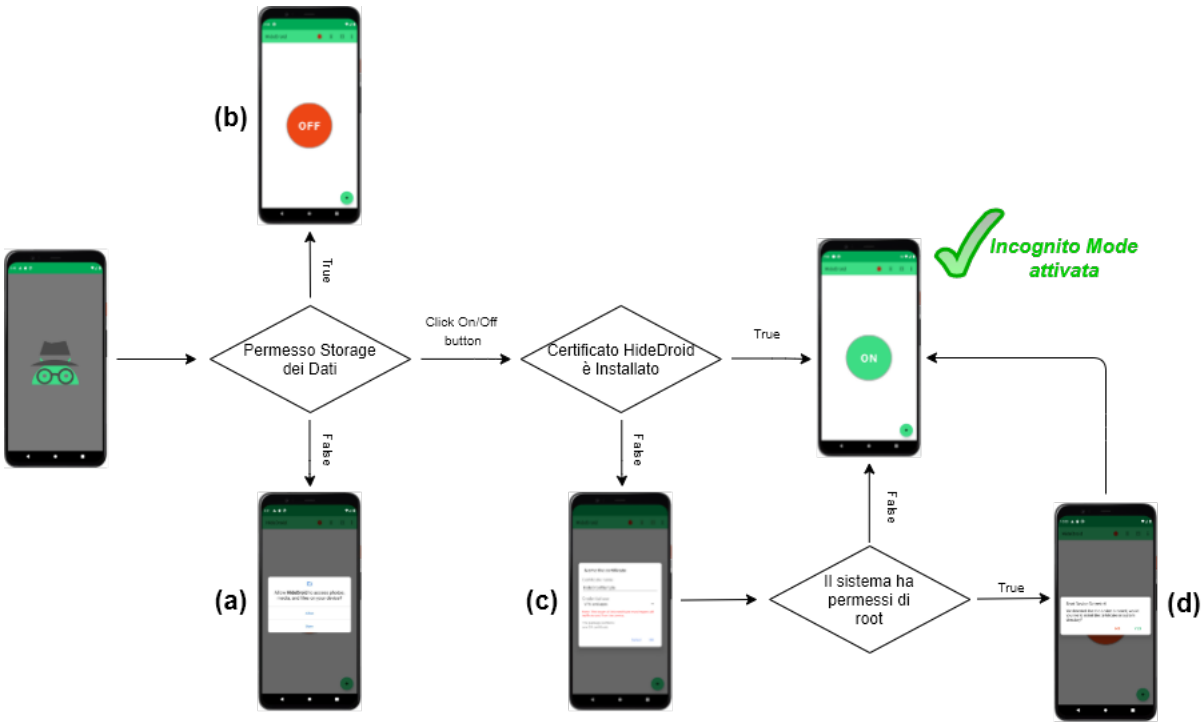


Figura 6: Fase di inizializzazione di HideDroid

Il certificato generato da HideDroid viene memorizzato all'interno dello store dei certificati utente all'interno del dispositivo, Figura 6 (step (c)). Successivamente, HideDroid esegue un'analisi del dispositivo per verificare la presenza dei permessi root [19], Figura 6 (step (d)). Questi permessi consentono all'utente di avere il controllo completo sul dispositivo, come ad esempio, il permesso di installare certificati nella directory di sistema la quale è inacessibile con i normali permessi utente. Se l'utente acconsente a

tale operazione, HideDroid memorizzerà il proprio certificato nella lista dei certificati di sistema.
Una volta compiute le seguenti operazioni, sarà possibile, per l'utente, attivare il servizio di anonimizzazione (Incognito Mode) di HideDroid, tramite l'apposito pulsante ON/OFF.

5.2 Configurazione del Livello di Privacy per App

5.2.1 Repackaging

Una volta configurata l'app HideDroid, l'utente ha la possibilità di visionare la lista delle applicazioni installate sul proprio dispositivo, Figura 7 (step (a)), per scegliere le app sulle quali vuole un maggiore livello di privacy.

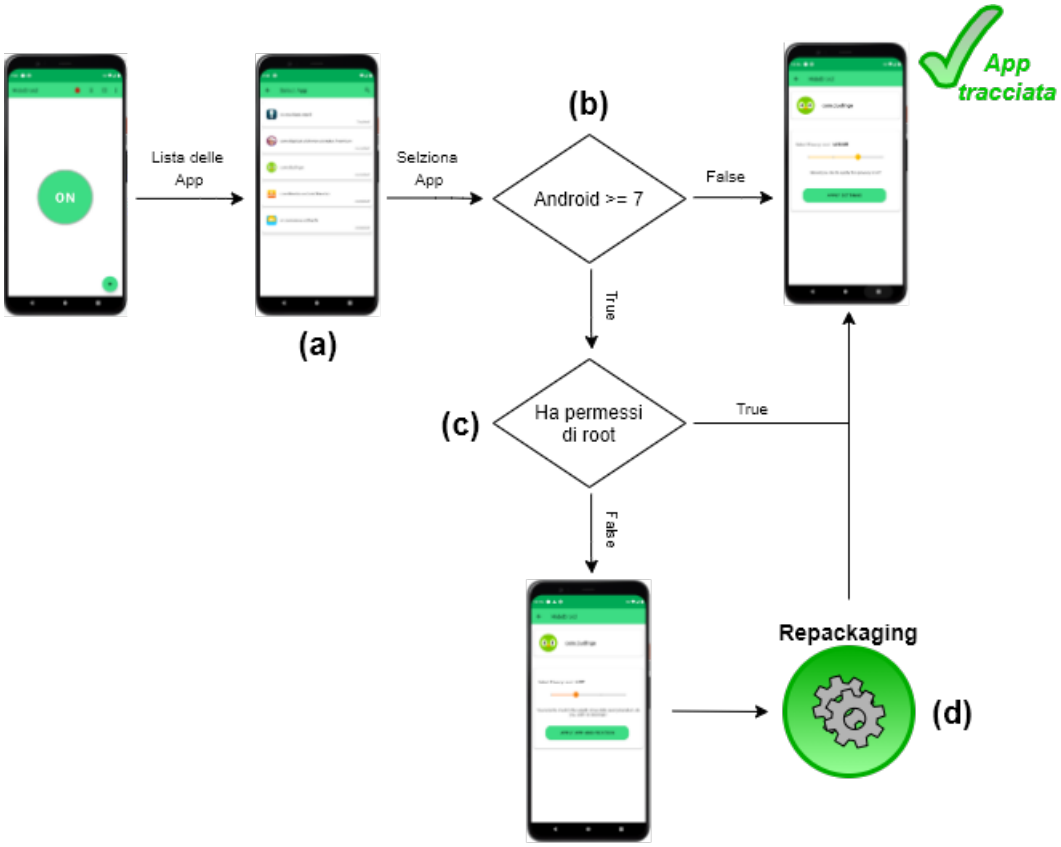


Figura 7: Fase di configurazione del livello di Privacy per App

Una volta che l'utente seleziona un'applicazione, HideDroid effettua un'analisi del sistema operativo per identificare la versione in esecuzione. Se la versione di Android è $\geq$ 7.0, Figura 7 (step (b)) e il dispostivo non presenta permessi di root (step (c)), allora

HideDroid evindenzierà e informerà l'utente della necessità di eseguire una ulteriore operazione preliminare di setup sull'applicazione selezionata. Tale operazione è una procedura definita di **repackaging** (step (d)) che consente di estrarre le risorse ed una rappresentazione del codice sorgente di una applicazione (file .apk), modificarli e successivamente ricreare il file apk. Questa procedura è necessaria per raggirare le nuove forme di sicurezza, descritte in Sezione 2.3, e consentire alle applicazioni di riconoscere come attendibili i certificati utente di un dispositivo. Per fare ciò, durante questa fase, vengono definiti e aggiunti nuovi **trust-anchors** all'interno del file di Network Security Configuration, come mostrato nello Script 1.

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <!-- Trust preinstalled CAs -->
            <certificates src="system" />
            <!-- Additionally trust user added CAs -->
            <certificates src="user" />
        </trust-anchors>
    </base-config>
</network-security-config>

```

Script 1: Esempio di configurazione di rete

Nel caso in cui l'applicazione non disponga di un file di configurazione, HideDroid ne creerà uno ad-hoc e lo aggiungerà tra le risorse dell'applicazione. Viceversa, se l'applicazione ha già dichiarato un file di configurazione di rete, allora si procederà a verificare la presenza dei trust-anchors necessari e in caso negativo verranno aggiunti. L'operazione di repackaging viene eseguita da HideDroid, mediante l'utilizzo del modulo **ApkTool**, Figura 8 (step (1)), che consente di disassemblare l'applicazione selezionata e decodificarne le risorse [20]. Una volta apportate le modifiche descritte, Figura 8 (step (2)), ApkTool compilerà nuovamente l'app generando un nuovo file apk, Figura 8 (step (3)). Per poter installare correttamente la nuova app generata, questa deve essere prima firmata, Figura 8 (step (4)), in modo tale da consentire al *PackageManager*[12] di Android di poter verificare l'intergrità dell'applicazione. Per firmare l'app, HideDroid utilizza la libreria **Apksig**. Se la versione dell'OS del dispositivo,

[12] https://developer.android.com/reference/android/content/pm/PackageManager

invece, è precedente ad Android 7.0 o se il dispositivo presenta permessi di root, allora l'utente potrà procedere direttamente alla selezione del livello di privacy per l'applicazione scelta. Al termine di questi passaggi, i dati generati dall'applicazione risulteranno anonimizzabili da HideDroid.

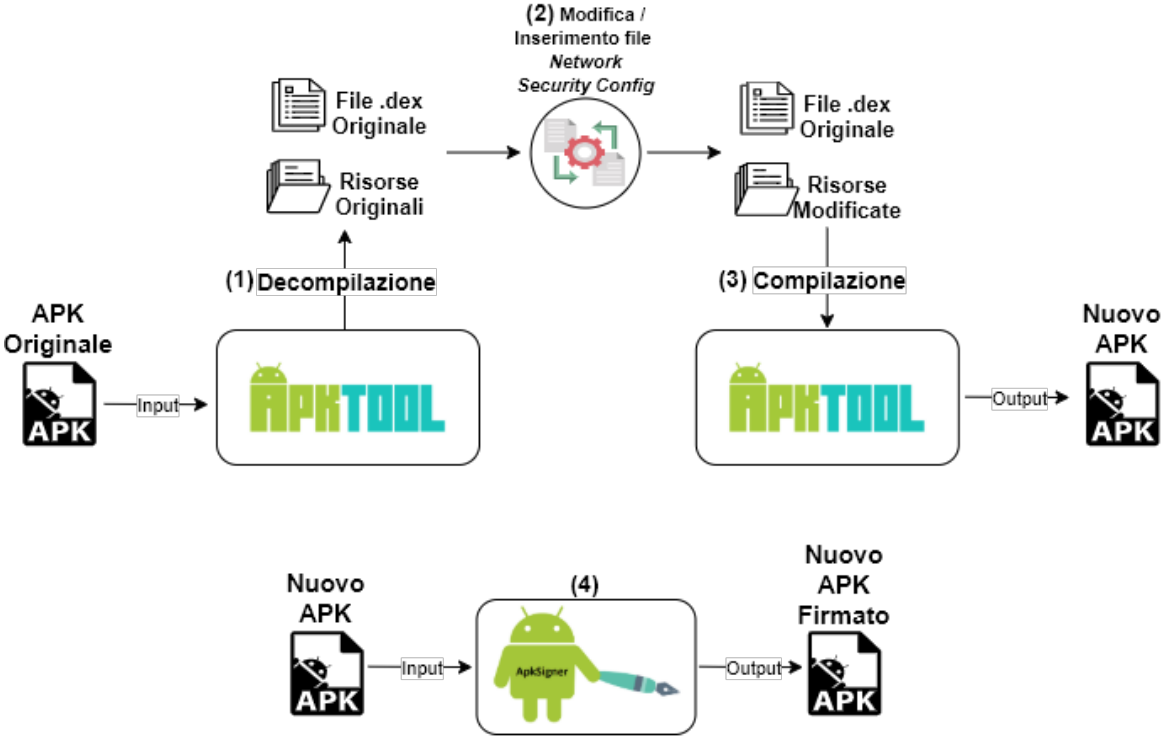


Figura 8: Fase di Repackaging

5.3 Anonimizzazione dei Dati

5.3.1 VPN

L'intercettazione del traffico, generato dalle applicazioni selezionate dall'utente, avviene grazie all'utilizzo di una **Virtual Private Network (VPN)**, implementata attraverso l'API VPN offerta da Android [21]. Questa componente svolge le funzioni del modulo *Privacy Detector*, definito all'interno della metodologia (Sezione 4.1). Essa genera un'interfaccia di rete virtuale basata sull'Internet Protocol (IP), configura indirizzi e regole di routing, e restituisce, infine, un file descriptor all'applicazione che la implementa. La peculiarità di tale servizio risiede nella trasparenza del server VPN rispetto alla comunicazione tra i due estremi, ovvero l'applicazione utilizzata e il server di destinazione. Infatti, come si vede in Figura 9, la comunicazione tra questi due end-

point non viene mai interrotta, ma intercettata dalla VPN, che, attraverso un proxy server, si interpone nella comunicazione. Essa instaura, così, due connessioni: una con il client (l'app) e una con il server di destinazione.

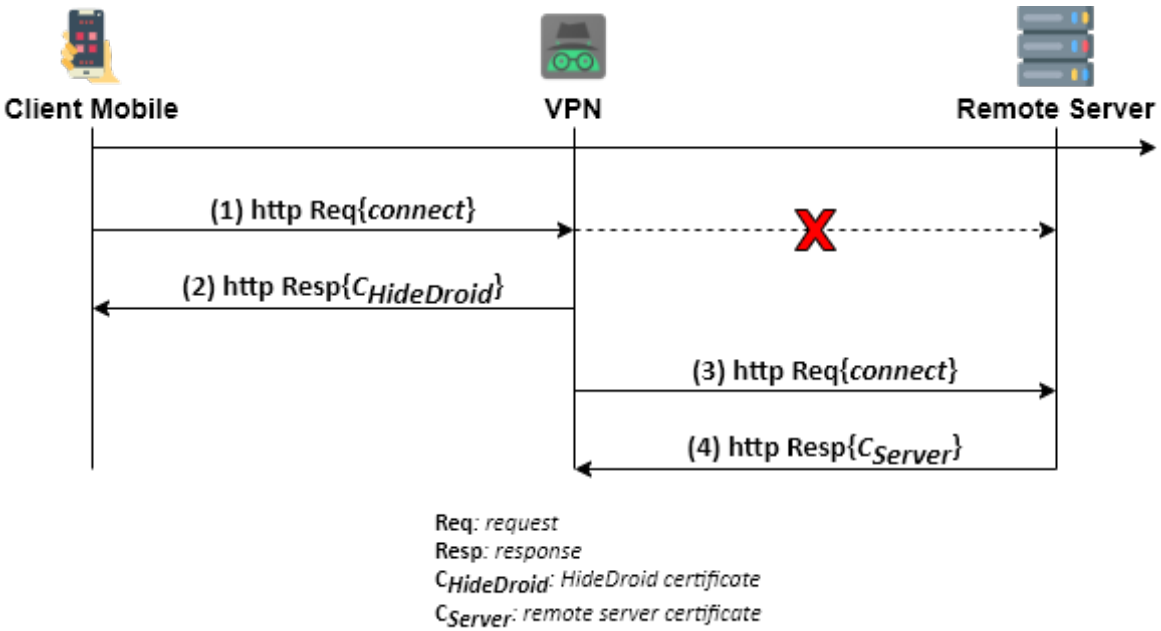


Figura 9: Sintesi del protocollo VPN

- Lato **applicativo.** La VPN, implementata da HideDroid, intercetta la richiesta di connessione (*http Req{connect}*) dell'applicazione verso il server remoto (step (1)) e le invia il proprio certificato autofirmato (step (2)), generato durante la fase di setup iniziale di HideDroid. L'applicazione riceve in risposta dalla VPN, il certificato, che verificherà correttamente in virtù delle operazioni svolte durante la fase di repackaging, descritta nella Sezione 5.2.1. Una volta terminata la verifica del certificato, viene instaurata la connessione tra l'applicazione e la VPN e tutte le richieste inviate dall'applicazione verranno così a lei inoltrate. Questa operazione è da considerarsi necessaria per ogni richiesta inviata dal client.

- Lato **server.** La VPN, a seguito della ricezione di una delle richieste di connessione (*http Req{connect}*), provenienti dalle applicazioni tracciate, provvederà ad inoltrarla al server di destinazione (step (3)). La VPN, riceverà, in risposta, il certificato del server destinatario (step (4)), e provvederà a verificarlo mediante la lista dei certificati installati sul dispositivo. Se la verifica ha successo, allora la connessione tra il server remoto e la VPN sarà stabilita correttamente.

Il servizio VPN è stato implementato attraverso l'utilizzo dei moduli **NetBare** ed **Interceptor**.

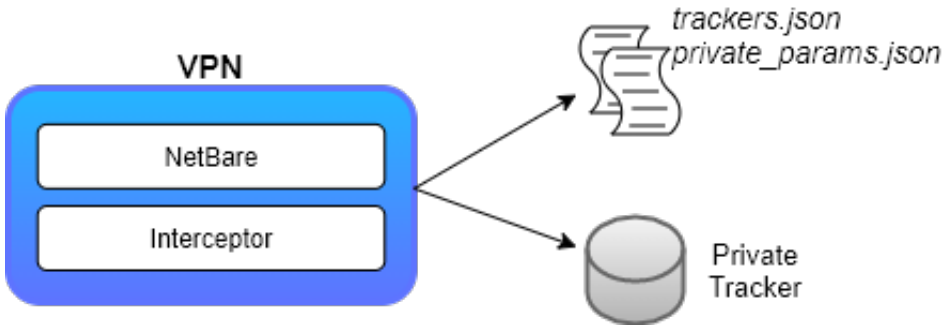


Figura 10: VPN module

5.3.2 Interceptor

Con il termine **Interceptor**, si fa riferimento ad una componente fondamentale della VPN, che ha l'obiettivo di effettuare lo *'sniffing'* delle richieste. Lo sniffing consiste nell'intercettazione di tutte le richieste e risposte inviate attraverso il protocollo HTTP in uscita e in ingresso ad un determinato client, come, ad esempio, un dispositivo Android. Le richieste intercettate possono essere, quindi, manipolate prima del loro invio o della loro ricezione, modificandone, ad esempio, gli headers e/o il body. Di seguito, riportiamo, in Figura 11, un'immagine esemplificativa che mostra il funzionamento dell'Interceptor, nell'interazione tra un generico client e un server remoto.

Figura 11: Esempio di funzionamento dell'Interceptor

HideDroid utilizza questo componente al fine di poter identificare quali richieste siano state generate da servizi di analytics. Inizialmente, l'Interceptor verifica il tipo di contenuto del pacchetto, accertandosi che questo non sia un'immagine, un video, un

audio o un font. Nel caso in cui non lo sia, si procederà alla verifica dell'appartenenza della richiesta al gruppo di servizi di analytics, mediante l'utilizzo di:

- un file `trackers.json`: in questo file è stato riportato un elenco dei principali host, riconducibili a servizi di analytics, memorizzandone alcune informazioni come nome, nome di dominio, indirizzo web. In tutto, questo file conta oltre 300 nomi di host appartenenti a svariati servizi di analytics.

- un file `private_params.json`: questo file nasce come supporto al precedente, che non è in grado di soddisfare a pieno il vasto numero di host di analytics esistenti al mondo. Questo file contiene alcuni degli attributi più utilizzati, presenti nelle richieste di analytics. Questi attributi sono stati individuati grazie all'osservazione dei dati raccolti durante la parte sperimentale e di testing dell'applicazione HideDroid.

- una tabella **Private Tracker**: HideDroid andrà a riempire questa tabella, definita all'interno della metodologia (Sezione 4.2), ogni qual volta un nuovo host di analytics, non presente nel file trackers.json, è stato riconosciuto come potenziale servizio di analytics, grazie alla presenza di uno o più attributi del file private_params.json.

L'Interceptor, all'arrivo di una nuova richiesta HTTP, verifica se l'host, definito nell'header del pacchetto, appartenga a uno degli oltre 300 servizi riportati all'interno del file `trackers.json`. Se l'interceptor non trova nessuna corrispondenza ricorrerà alla tabella Private Tracker, per verificare se l'host del pacchetto sia già stato riconosciuto, in precedenza, come potenziale servizio di analytics. Se anche in questo caso non dovesse trovare nessuna corrispondenza, allora, viene utilizzata l'euristica, descritta in Sezione 4.1 per determinare la provenienza della richiesta Se anche quest'ultimo controllo dovesse fallire, la richiesta verrà semplicemente inoltrata al server di destinazione originale; se invece almeno uno dei precendenti controlli ha avuto esito positivio, HideDroid memorizzerà la richiesta per procedere alla sua anonimizzazione.

5.3.3 Data Anonymizer

Il servizio di anonimizzazione, effettuato dal modulo **Data Anonymizer**, rappresenta il cuore dell'applicazione. Il suo ruolo, svolto dall'omonimo modulo presentato all'interno della metodologia (Sezione 4.7), è quello di anonimizzare le informazioni contenute

all'interno di ciascuna richiesta di analytics, intercettata dal modulo Interceptor. Per poter portare a termine il proprio compito, il Data Anonymizer, si basa su diversi moduli, come mostrato in Figura 12, ognuno con la sua funzione specifica.

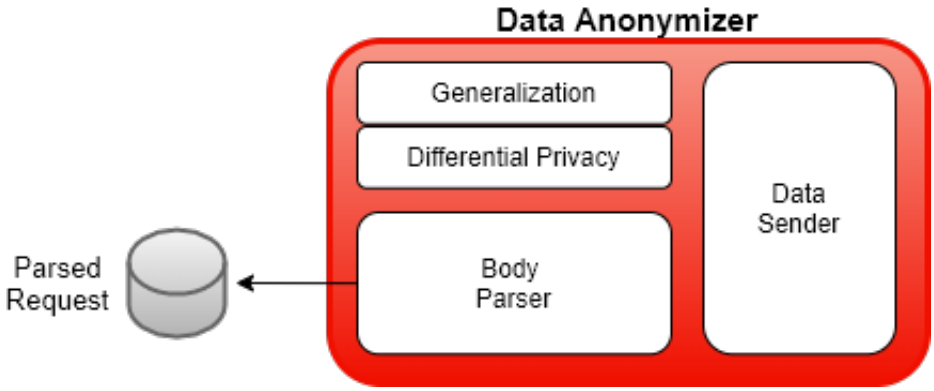


Figura 12: Data Anonymizer module

Dopo che l'utente ha avviato la modalità Incognito Mode, le richieste, in arrivo al Data Anonymizer, subiscono un processo di decodifica utile a riorganizzare le informazioni, presenti al loro interno, in apposite strutture dati. Tale operazione viene eseguita dal modulo **Body Parser**, che si occupa sia della loro decodifica, in fase di analisi, che della loro codifica, prima dell'invio. Il Body Parser è in grado di decodificare diverse tipologie di pacchetti, sulla base del valore di due parametri: l'header **Content-Type** e l'header **Content-Encoding**.
I valori di Content-Type che il Body Parser è in grado di decodificare sono:

- **Content-Type: application/x-www-form-urlencoded.** In questo caso, il body della richiesta si presenta nel formato "$chiave_1 = valore_1 \& chiave_2 = valore_2 \& \ldots \& chiave_n = valore_n$", dove il termine chiave, identifica univocamente la risorsa associata all'interno del body, come si può notare nello Script 2.

```
1        POST /test HTTP/1.1
2        Host: foo.example
3        Content-Type: application/x-www-form-urlencoded
4        Content-Length: 27
5
6        field1=value1&field2=value2&field3=value3
7
```

Script 2: Esempio di pacchetto application/x-www-form-urlencoded

- **Content-Type: application/json.** In questo caso il body della richiesta si presenta come una stringa in formato json. Mediante questo formato, è possibile serializzare oggetti, array, numeri, stringhe e booleani (Script 3).

```
1         POST /api/2.2/auth/signin HTTP/1.1
2         HOST: my-server
3         Content-Type:application/json
4         Accept:application/json
5
6         {
7           "credentials": {
8             "name": "administrator",
9             "password": "passw0rd",
10            "site": {
11              "contentUrl": ""
12            }
13          }
14        }
15
```

Script 3: Esempio di pacchetto application/json

- **Content-Type: multipart/form-data.** Questo tipo di formato viene solitamente utilizzato quando vengono inviati dati letti da file, il cui contenuto viene inserito nel body, sotto uno specifico header. Le richieste sono caratterizzate, all'interno del body, dalla presenza di un *boundary*, una sequenza di caratteri alfanumerica, che consente di separare headers di differenti risorse, identificate tramite una chiave univoca *name*. Di seguito, in Script 4, ne viene riportato un esempio.

```
1         POST /test HTTP/1.1
2         Host: foo.example
3         Content-Type: multipart/form-data;boundary="boundary"
4
5         --boundary
6         Content-Disposition: form-data; name="field1"
7
8         value1
9         --boundary
10        Content-Disposition: form-data; name="field2";
11        filename="example.txt"
```

```
12
13          value2
14          --boundary--
15
```

Script 4: Esempio di pacchetto multipart/form-data

Se la richiesta analizzata presenta una qualche forma di compressione dei dati, il Body Parser è in grado di identificarla analizzando l'header Content-Encoding. I tipi di compressione, gestiti dal Body Parser, sono:

- **Content-Encoding: gzip.** Tale formato di compressione utilizza l'algoritmo di Lempel-Ziv [22].
- **Content-Encoding: deflate.** Tale formato di compressione, si appoggia sulla struttura zlib [23], specificando l'algoritmo di compressione *deflate* [24].

Entrambi sono formati di compressione, che vengono utilizzati al fine di diminuire la banda trasmissiva, riducendo la congestione della rete. Una volta terminata la decodifica, il Data Anonymizer ne memorizzerà i risultati in una struttura dati, chiamata Parsed Request, come descritto all'interno della metodologia (Sezione 4.6). In Tabella 2 vengono riportate le informazioni contenute al suo interno. A questo punto il Data

Parsed Request	
Headers	gli headers della richiesta intercettata
Content-Type	una delle possibili codifiche gestite dal Body Parser (x-www-form-urlenconded, application/json, multipart/form-data)
Content-Encoding	una tra le possibili compressioni gestite dal Body Parser (gzip, deflate)
URL	l'indirizzo completo di destinazione della richiesta, composto da: metodo (POST e GET principalmente), host (nome del dominio), path
Body	il corpo della richiesta, contenente informazioni sull'utente o sul dispositivo
Applicazione	il nome dell'applicazione che ha generato la richiesta

Tabella 2: Tabella struttura dati Parsed Request

Anonymizer avvia il processo di anonimizzazione applicando le tecniche di Generalizzazione e di DP, descritte nelle Sezioni 4.7.1 e 4.7.2.
Terminato il processo di anonimizzazione, la richiesta subirà un ultimo processo di codifica da parte del Body Parser, il quale, appoggiandosi alle informazioni contenute all'interno della struttura dati Parsed Request, ricostruirà la codifica originale del pacchetto.
Infine la richiesta, così anonimizzata e codificata, verrà inoltrata al destinatario originale. Il modulo responsabile dell'invio è il Data Sender, come definito all'interno della metodologia (Sezione 4.8), il quale, sulla base della natura della richiesta, sceglie come inviare il pacchetto. In particolare, se il pacchetto in questione è lo stesso pacchetto di partenza, di cui si è anonimizzato il contenuto, allora il Data Sender provvede ad inviarlo tramite il modulo Interceptor, che continua a mantenere, con il destinatario, la comunicazione aperta. Se invece il pacchetto, è un pacchetto "fittizio", creato dal Data Anonymizer, allora il Data Sender provvederà da sé all'invio.

6 Risultati Sperimentali

Per dimostrare l'efficacia e la fattibilità della nostra implementazione, abbiamo testato HideDroid sulle prime 2000 app più scaricate, presenti sul Google Play Store. Gli esperimenti sono stati effettuati su una macchina virtuale Ubuntu; in Tabella 3, ne vengono riportate le specifiche.

OS	Ubuntu 20.04.1 LTS
Architecture	64-bit
Virtualization	VMware
CPU	AMD Ryzen ThreadRipper 3960x
RAM	32 GB

Tabella 3: Specifiche macchina virtuale Ubuntu

Le applicazioni selezionate sono state testate su un emulatore Android, privo di Google Play Services, creato con Android Studio 4.1.1. In Tabella 4 sono riportate le specifiche hardware dell'emulatore utilizzato.

OS	(Google APIs) Android 10
Resolution	1440 x 3040:560dpi
Architecture	x86
Model	Pixel 4 XL
Core	2-core
RAM	4 GB
Heap	3 GB
Internal Storage	15 GB

Tabella 4: Specifiche emulatore Android

Dell'insieme di app selezionate, in Tabella 5, riportiamo un esempio delle prime dieci, con il loro relativo numero di downloads.

Nome App	**#Downloads**
Facebook	5.000.000.000
Google Chrome	5.000.000.000
Flipboard	1.000.000.000
Candy Crush Saga	1.000.000.000
Dropbox	1.000.000.000
Facebook Lite	1.000.000.000
Instagram	1.000.000.000
Excel	1.000.000.000
OneDrive	1.000.000.000
TikTok	1.000.000.000

Tabella 5: Lista delle prime 10 App testate

Le 2000 applicazioni scelte, sono state testate in maniera automatica, utilizzando uno strumento per il black-box testing, chiamato **Droidbot** [25]. Questo tool, generando input di eventi, consente di testare le applicazioni Android, in maniera automatica. Per farlo, Droidbot implementa algoritmi di esplorazione delle applicazioni, trattandole come grafi, attuando una rapida e vasta esplorazione delle app. Inoltre, Droidbot fornisce la possibilità di definire file json di configurazione, in maniera personalizzata, utili a specificare determinate azioni da compiere al riconoscimento di un determinato stato dell'applicazione, come, ad esempio, una schermata di login o di registrazione. Il flusso di esecuzione dei test prevede l'automatizzazione di tre fasi:

1. **Installazione dell'App.** L'applicazione da analizzare viene installata sul dispositivo Android. Per poter installare l'applicazione, abbiamo utilizzato il tool offerto direttamente da Android: **Android Debug Bridge (adb)** [26].

2. **Configurazione dell'App selezionata.** Affinchè il traffico dell'app possa essere anonimizzato, l'app installata necessita di opportune configurazioni da parte di HideDroid. Per fare ciò, si è utilizzato il tool di Droidbot, fornendogli in input, un file json di configurazione ad hoc, per il riconoscimento di determinati stati dell'app HideDroid. Ad ogni stato riconosciuto, è stata associata, quindi, un'azione da eseguire, automatizzando, così, il processo di setup, come descritto

in Sezione 5.2.1. Droidbot, una volta avviata l'app HideDroid, automatizza i passaggi di configurazione dell'app da analizzare, descritti in Figura 7.

3. **Utilizzo dell'App.** Infine utilizzando sempre il tool Droidbot si è stimolata l'app in maniera automatica per un arco di tempo complessivo di 10 minuti.

Delle 2000 applicazioni scaricate, 1220 (il 61%), sono state testate e analizzate con successo utilizzando HideDroid (Figura 13). Le restanti 780 (il 39%) hanno fallito per motivi legati allo step di repackaging (necessario da HideDroid per intercettare il traffico), o per motivi legati alla incompatibilità dell'app con il dispositivo utilizzato per la fase di test. Le applicazione che hanno fallito per lo step di repackaging sono 246 (il 12%) e la motivazione è dovuta all'esistenza di tecniche di **anti-repackaging** [27]. Queste tecniche, infatti, evitano ad applicazioni modificate da terze-parti (ad es., utenti malevoli) di funzionare correttamente. Uno dei servizi maggiormente utilizzati è quello offerto da Google, chiamato **SafetyNet**[13]. Generalmente, è possibile ritrovare tali tecniche nelle applicazioni più scaricate e quindi più famose, come, ad esempio, Facebook, TikTok e Instagram. Le restanti 534 applicazioni (il 27%) non sono state analizzate, perchè non è stato possibile installarle sul dispositivo. Le cause di questo problema sono ritrovabili in tre aspetti:

1. Alcune applicazioni, di quelle selezionate, sono app di sistema, integrate nelle funzionalità del dispositivo stesso e quindi non riconoscibili come applicazioni esterne.

2. Un sottoinsieme delle app presentavano un file, in formato .apk, corrotto (il file non veniva riconosciuto con tale formato) o privo di firma. In quest'ultimo caso si è provato a firmare autonomamente le app, in modo da consentirne una corretta installazione sul dispositivo. In alcuni casi, però, anche l'operazione di firma non è andata a buon fine per problematiche legate all'integrità del file apk stesso.

3. L'architettura x86 del dispositivo utilizzato, non è compatibile con alcune delle app selezionate, sviluppate, invece, solo per l'architettura ARM.

[13] https://developer.android.com/training/safetynet/attestation

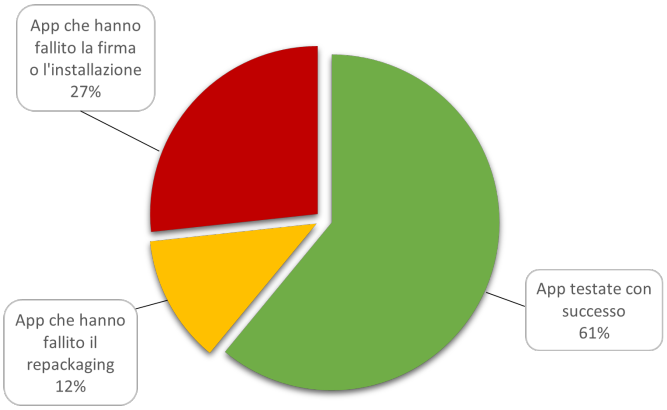


Figura 13: Diagramma delle applicazioni testate

6.1 Analisi dell'utilizzo delle librerie di analytics

Grazie all'analisi dei risultati ottenuti dai test sperimentali sulle 1220 app testate con successo, con HideDroid, è stato posssibile estrarre le seguenti informazioni statistiche. In Figura 14 (a), è possibile notare le percentuali di utilizzo dei servizi di analytics, 430 hosts (il 62%) ed altri servizi, 270 hosts (il 38%), generalmente utili al normale funzionamento dell'applicazione. Gli host di analytics, dal diagramma di Figura 14 (b), sono stati identificati da HideDroid, utilizzando le due tecniche descritte nelle sezioni precedenti: 153/700 (il 22%) utilizzando la lista di host relativi a servizi di analytics conosciuti a priori e 277/700 (il 40%), utilizzando l'euristica descritta in Sezione 4.1.

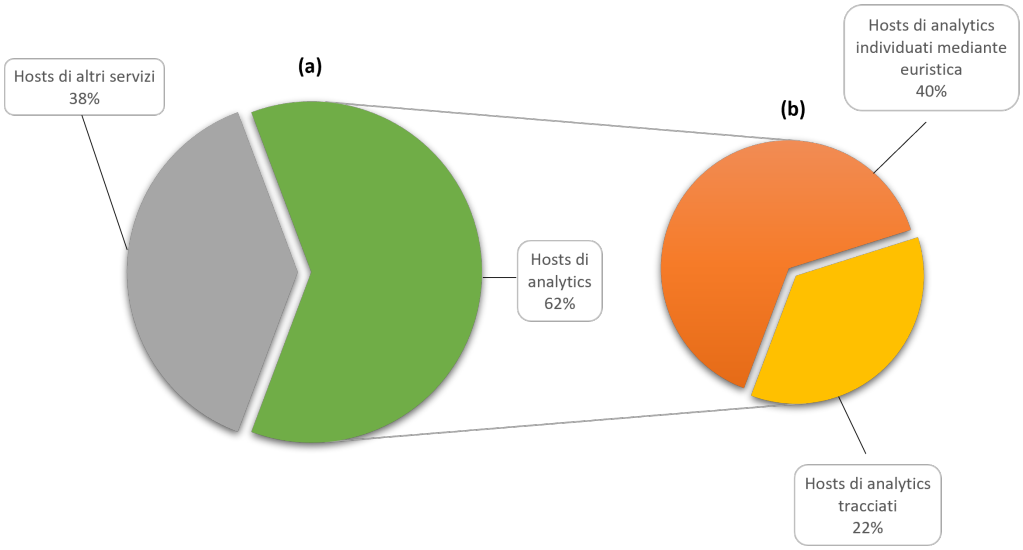


Figura 14: Diagramma degli Host individuati

Partendo da questi ultimi dati, si è quindi approfondito l'impatto di ciascuno dei singoli host di analytics, analizzando il numero di richieste inviate da ciascuno di essi. In Figura 15, sono stati riportati i dati relativi al numero di pacchetti inviati dai primi venti host di analytics per numero di richieste, ordinati in maniera decrescente. Dal diagramma ottenuto, si evince che gli host più largamente presenti, all'interno delle applicazioni testate, siano *firebaseinstallations.googleapis.com* di Google e *graph.facebook.com* di Facebook, con oltre 39000 e 27000 pacchetti inviati, rispettivamente. Queste, assieme, costituiscono, infatti, quasi l'80% delle richieste complessive.

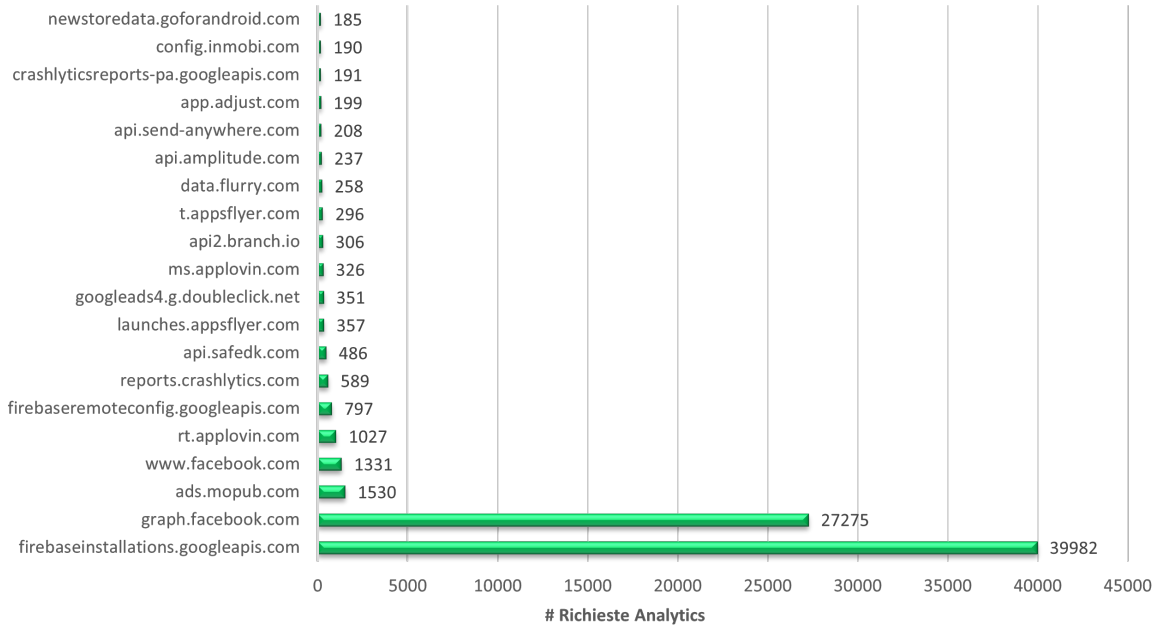


Figura 15: Diagramma delle richieste per singoli Host di Analytics

Dalla somma di questi valori si è potuto ricavare anche il rapporto tra il numero di richieste di analytics, rispetto al numero complessivo delle richieste intercettate da HideDroid: 94195. In Figura 16 (a), 85044 richieste (il 90%) appartengono a servizi di API di analytics; le restanti 9151 (il 10%) appartengono, invece, ad altri servizi di API, necessari al corretto funzionamento delle app stesse. In Figura 16 (b), infine, viene mostrato come il rapporto tra quelle individuate mediante euristica, 46814 richieste (il 50%) e quelle mediante file `trackers.json` (Sezione 5.3.2), 38230 richieste (il 40%) sia piuttosto equilibrato. Questo dato mette in risalto che, ad oggi, le applicazioni tendano ad appoggiarsi a servizi di analytics, che raccolgono un grande quantitativo di informazioni, senza che l'utente ne abbia dato il consenso o che ne sia a conoscenza.

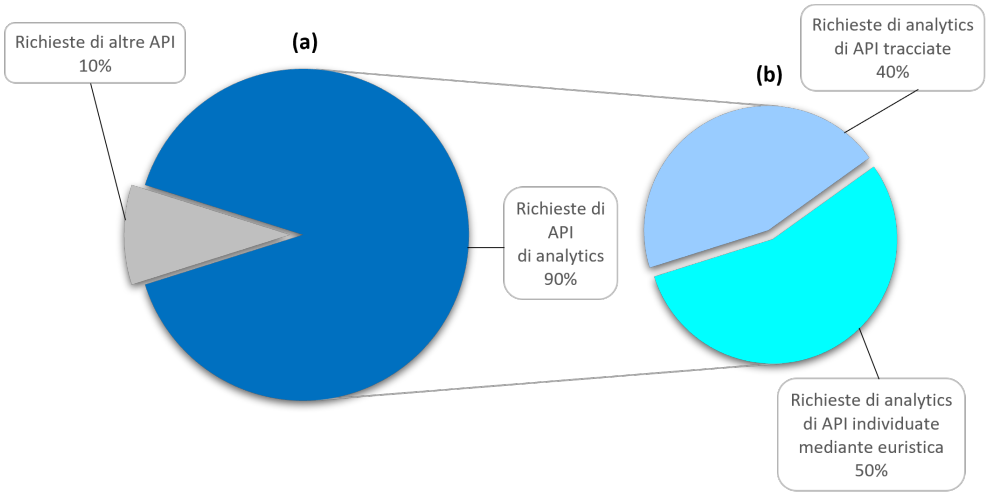


Figura 16: Diagramma dell'insieme complessivo di richieste intercettate

Come descritto in Sezione 2, una richiesta di analytics può contenere dati di natura diversa al suo interno, come informazioni appartenenti agli eventi, che l'utente genera durante l'interazione con l'applicazione, oppure informazioni sul dispositivo e/o sull'utente stesso. I test sperimentali hanno permesso di raccogliere gli attributi e le informazioni memorizzate con più frequenza dalle librerie di analytics.

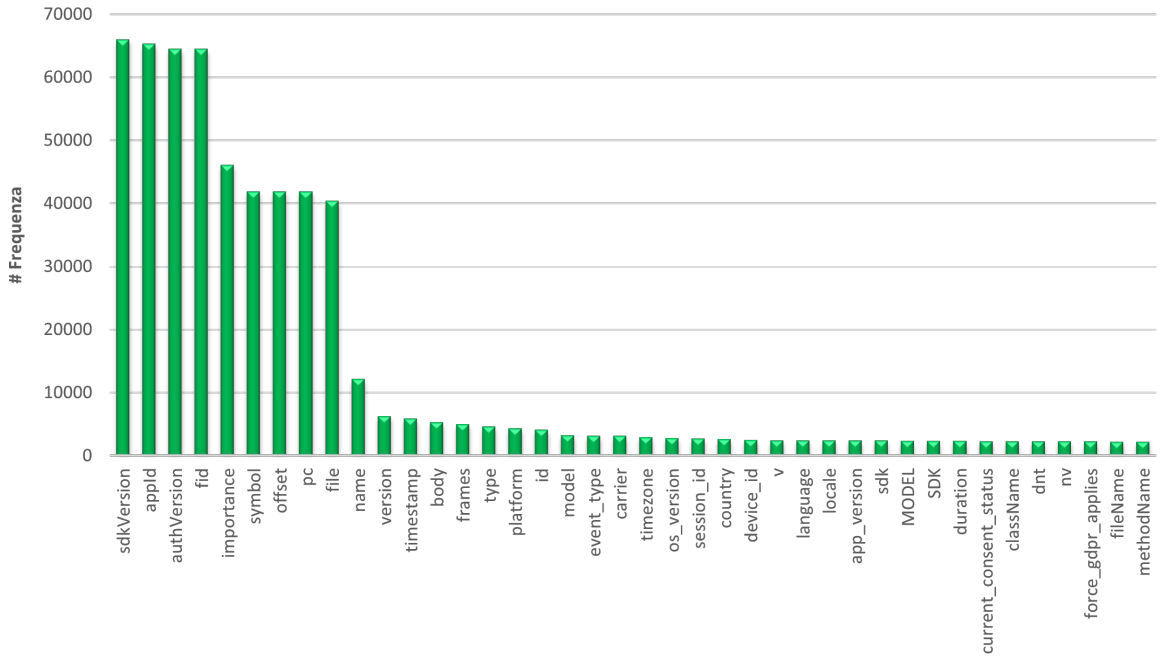


Figura 17: Diagramma degli attributi più presenti nelle richieste di Analytics

In Figura 17 troviamo alcune delle informazioni più frequenti, come: *"carrier"*, che identifica il modello di dispositivo utilizzato, *"timezone"*, il fuso orario relativo alla zone geografica impostata sul dispositivo, *"language"*, la lingua selezionata dall'utente all'interno dell'applicazione, *"total_memory"*, che identifica la quantità di memoria del dispositivo e *"battery"*, che indica il livello di batteria attuale del dispositivo.
Per quanto riguarda gli eventi memorizzati da queste librerie, in Figura 18, viene riportato, in ordine decrescente di frequenza, l'insieme degli eventi registrati durante l'utilizzo delle applicazioni testate. L'evento più frequente è *"CUSTOM_APP_EVENTS"*, che è riconducibile al servizio di analytics di *graph.facebook.com* di Facebook. Attraverso questo attributo, gli sviluppatori sono in grado di definire, e quindi monitorare, eventi personalizzati creati ad-hoc.

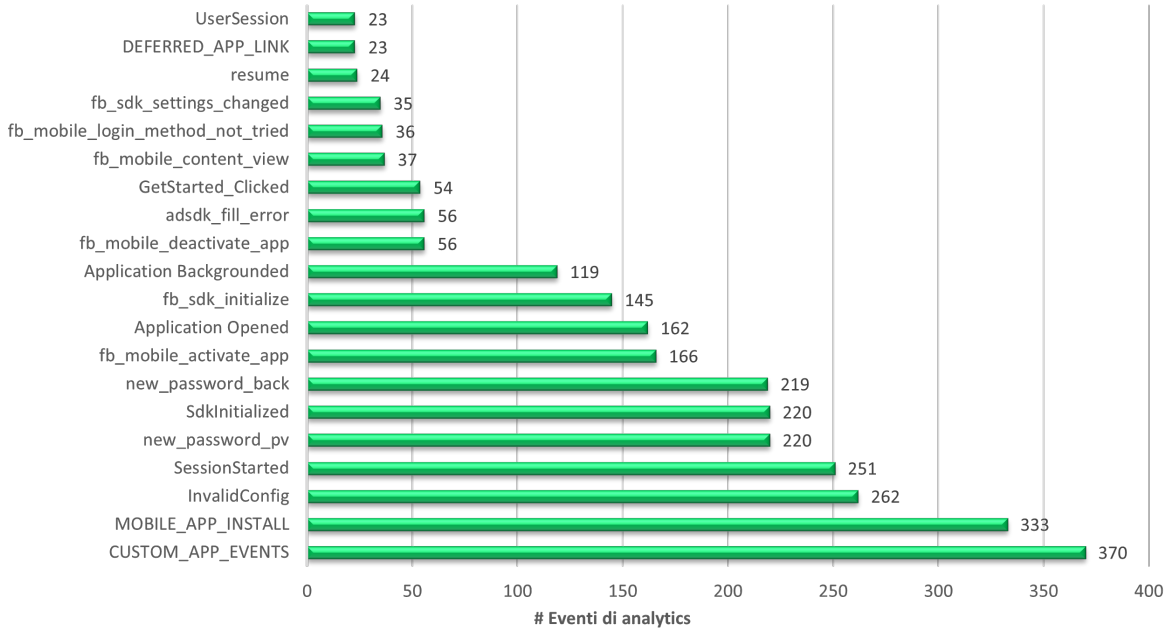


Figura 18: Diagramma degli eventi registrati durante l'utilizzo delle applicazioni

6.2 Risultati sperimentali sull'utilizzo di HideDroid

Durante il processo di anonimizzazione gli attributi presenti all'interno di una richiesta subiscono modifiche tali da poterne compromettere l'accettazione al momento dell'inoltro al server di destinazione. In Tabella 6 viene riportata la percentuale delle richieste accettate per i primi diciotto host di analytics, rispetto al numero di richieste inviate verso i server di destinazione. Uno degli host di analytics più restrittivi è risultato

essere *firebaseinstallations.googleapis.com* di Google, che ha rifiutato la totalità delle richieste inviate da HideDroid. Questo fenomeno è riconducibile all'utilizzo, da parte di Google, di particolari formati come *protobuf* [28], che per essere analizzati richiedono la conoscienza della struttura dati orginaria, ovvero quella con cui sono state serializzate le informazioni della richiesta. In secondo luogo, come si può vedere sempre dalla Tabella 6, la percentuale di richieste accettate dall'host *graph.facebook.com* di Facebook, è superiore al 95%. Questo è dato dal fatto che il processo di anonimizzazione delle richieste relative a questo host di analytics, è stato gestito, in HideDroid, mediante l'utilizzo di un apposito DGH, contenente regole specifiche per quell'host.

Host	**Percentuale di richieste accettate**
firebaseinstallations.googleapis.com	0%
graph.facebook.com	95.8%
googleads.g.doubleclick.net	99.6%
pagead2.googlesyndication.com	100%
ads.mopub.com	53.8%
tpc.googlesyndication.com	100%
www.googleadservices.com	99.7%
rt.applovin.com	100%
firebaseremoteconfig.googleapis.com	16.4%
adservice.google.com	100%
reports.crashlytics.com	98.8%
settings.crashlytics.com	97%
googleads4.g.doubleclick.net	97.7%
api2.branch.io	0%
www.google-analytics.com	69%
launches.appsflyer.com	92.1%
api.amplitude.com	9.3%
ms.applovin.com	100%

Tabella 6: Percentuali di risposte accettate

L'utilizzo di un apposito DGH si è reso necessario sia per la sua diffusione sia per la presenza di controlli, da parte del server, sui valori di alcuni degli attributi presenti

all'interno delle richieste invate. In Tabella 7, vengono riportate alcune delle regole utilizzate da Facebook, che si è stati in grado di individuare.

Attributo	Possibili Valori
event	[MOBILE_APP_INSTALL, CUSTOM_APP_EVENTS, DEFERRED_APP_LINK]
custom_events	lista contenente almeno l'attributo *_eventName*
_eventName	una qualunque stringa, lunga al più 39 caratteri alfanumerici, contenente anche caratteri "_", "-"
_logTime	deve essere un intero
_eventName_md5	calcolabile dal corrispondente valore dell'attributo *_eventName*
boolean	[true, false]
SDK_CAPABILITY	array di interi
PLACEMENT_ID	non modificabile
TEMPLATE_ID	non modificabile

Tabella 7: Tabella di regole individuate per *graph.facebook.com*

6.3 Esempi rilevanti

Per concludere, di seguito, riportiamo alcune delle richieste più significative, individuate dall'analisi effettuata, al termine degli esperimenti.
Il primo esempio è una richiesta individuata durante l'utilizzo dell'app *Akinator*[14]. Come è possibile vedere in Script 5, l'applicazione ottiene una lista delle app installate dal dispositivo e le invia al proprio servizio di analytics.

```
1 POST /ship HTTP/2.0
2 Authority: collector.appconsent.io
3 User-Agent: sfbx-xchange/2.3.17 (Android 6.0; Android Android SDK
   built for x86)
4 Content-Type: application/json; charset=UTF-8
5 Accept-Encoding: gzip
6
```

[14] https://play.google.com/store/apps/details?id=com.digidust.elokence.akinator.freemium&hl=it&gl=US

```
{
  "appconsentID": "36e931a0-20e2-45a0-8e80-0ab6444f06fd",
  "buildID": 336,
  "data": {
      "appNameBundle": "com.digidust.elokence.akinator.freemium",
      "deviceCarrier": "Android",
      "deviceCountryCode": "US",
      "deviceManufacturer": "Unknown",
      "deviceModel": "Android SDK built for x86",
      "deviceOS": "ANDROID",
      "deviceOSVersion": "23",
      "networkType": "LTE"
  },
  "packagesInstalled": [
      {
          "collectTimestamp": 1596188510607,
          "installTimestamp": 1596116759181,
          "packageName": "com.example.datacollectapp",
          "updateTimestamp": 1596120525881
      },
      {
          "collectTimestamp": 1596188510607,
          "installTimestamp": 1596117888238,
          "packageName": "com.guoshi.httpcanary",
          "updateTimestamp": 1596117888238
      },
      ...
  ]
}
```

Script 5: Pacchetto catturato durante l'utilizzo di *Akinator*

Infine, per concludere, riportiamo nello Script 6, un frammento del contenuto di una richiesta, appartenente all'app di *Duolingo*[15] e indirizzata ad un servizio di analytics. Al suo interno è possibile ritrovare numerose informazioni, sia sul dispositivo utilizzato, sia sugli eventi generati dall'utente. Ad esempio: *"event_type": "app_open"*, per indicare l'evento di avvio dell'applicazione da parte dell'utente; *"memory_maximum"*, ovvero la memoria massima complessiva del dispositivo, *"learning_language"*, che indica la lingua che l'utente vuole imparare nell'utilizzo dell'app; *"total_time_spent"*,

[15] https://play.google.com/store/apps/details?id=com.duolingo&hl=it&gl=US

per indicare il tempo di utilizzo dell'applicazione; *"geoip_country"*, che indica il paese di provenienza della richiesta e quindi dell'utente; *"device_year"*, indicante l'anno del dispositivo; *"sim_network_country"*, che contiene la provenienza della sim.


```
[{
      "event_type": "app_open",
      "event_timestamp": 1600184824000,
      "client": {
          "client_id": "android-excess"
      },
      "attributes": {
          "memory_maximum": 536870912,
          "memory_system_available": 630865920,
          "memory_total": 28699224,
          "memory_free": 1259384,
          "memory_used": 27439840,
          "learning_language": "en",
          "learning_reason": "WORK",
          "geoip_country": "US",
          "has_google_play_services": true,
          "device_year": 2013,
          "$screen_height": 2792,
          "$screen_width": 1440,
          "sim_provider_country": "US",
          "volume": 0,
          "total_time_spent": 135,
          "sim_network_country": "US",
          "$os": "Android",
          "$model": "Android SDK built for x86",
          "$os_version": "8.0.0",
          "$brand": "google",
          ...
      }
  },
  ....
  ]
```


Script 6: Pacchetto catturato durante l'utilizzo di *Duolingo*

6.4 Discussione

In questa sezione, si andranno a discutere gli approcci metodologici e implementativi della soluzione proposta in questa tesi.

Tecniche di anonimizzazione adottate. La metodologia proposta si offre di anonimizzare le richieste, contenenti potenziali informazioni sensibili, appoggiandosi a tecniche di generalizzazione e di DP. Il contenuto informativo dei dati, presenti all'interno di una richiesta di analytics, è molto eterogeneo e questo ha comportato difficoltà nel scegliere quali tecniche di anonimizzazione adottare. La generalizzazione proposta, pur appoggiandosi ad un DGH, non è in grado di applicare una regola universale per ogni tipo di informazione, dovendo, così, ricorrere ad una generalizzazione più "grezza", tramite la sostituzione di caratteri con "*" e l'arrotondamento di numeri. Questa scelta si è ripercossa negativamente sull'accettazione dei pacchetti anonimizzati, da parte di alcuni dei servizi di analytics (ad esempio: *Google Firebase Analytics* e *Facebook Analytics*), che, possedendo regole ben precise, eseguono controlli sulla validità del valore di alcuni attributi, presenti nella richiesta.

Criticità presenti nel Repackaging. Una delle fasi più critiche dell'implementazione proposta, è l'operazione di repackaging delle applicazioni, selezionate dall'utente. Questa operazione è basata sulla modifica dell'applicazione (il file `.apk`), violando, quindi, la sicurezza del dispositivo o dell'applicazione stessa. Negli anni, quindi, sono nate nuove tecniche di anti-repackaging, che ostacolano questa operazione. Alcune applicazioni, generalmente quelle più famose e più scaricate, come, ad esempio, Facebook, Instagram, TikTok, presentano tecniche di anti-repackaging. Una delle tecniche di anti-repackaging più utilizzate, e riscontrata durante gli esperimenti, è stata SafetyNet [29]: un'insieme di API, sviluppate da Google, che consentono di proteggere, a run-time, l'applicazione, e quindi il dispositivo, da una sua possibile manomissione. I due parametri con cui viene verificata l'integrità di un dispositivo sono:

- *ctsProfileMatch*, un parametro piu rigoroso sull'integrità del dispositivo. Se il suo valore è *true*, il profilo del dispositivo che esegue l'app, corrisponde al profilo di un dispositivo che ha superato il test di compatibilità Android.
- *basicIntegrity*, un parametro più permissivo sull'integrità del dispositivo. Se solo il suo valore è *true*, è probabile che il dispositivo, che esegue l'app, non sia stato

manomesso. Tuttavia, il dispositivo non ha necessariamente superato i test di compatibilità di Android.

Di seguito, in Tabella 8, riportiamo alcuni casi di come lo stato di un device possa influenzare i valori di questi due parametri.

Stato del Device	**ctsProfileMatch**	**basicIntegrity**
Dispositivo certificato e autentico	true	true
Dispositivo certificato con bootloader sbloccato	false	true
Dispositivo con ROM personalizzata (non rootato)	false	true
Dispositivo emulato	false	false
Dispositivo rootato	false	false

Tabella 8: Tabella di valutazione di SafetyNet

Pinning dei certificati. Un'app può proteggersi da certificati emessi in modo fraudolento, mediante una tecnica nota come pinning. L'applicazione ha la possibilità di restringere l'insieme dei certificati da lei riconosciuti come validi limitandoli a quelli necessari solo per il suo corretto funzionamento. Questa restrizione permette all'app di proteggersi da certificati non noti al dispositivo stesso e da certificati self-signed, come quello di HideDroid, che non sono garantiti da nessuna CA. Questa operazione può essere eseguita sia a livello di file di configurazione (*Network Security Config*, Sezione 2.3), sia a livello di codice. Nel primo caso, HideDroid è in grado di raggirare questa limitazione tramite l'operazione di repackaging; nel secondo caso, invece, non è possibile intervenire, poichè richiederebbe una modifica del codice sorgente dell'applicazione e quindi una possibile alterazione al funzionamento dell'app.

7 Conclusioni e Sviluppi Futuri

In questo lavoro di tesi è stata presentata una metodologia, chiamata MobHide, che ha l'obbiettivo di affrontare il problema della privacy durante l'utilizzo di applicazioni mobile. La fattibilità, l'affidabilità e l'efficacia dell'approccio sono state misurate tramite una sua prototipazione in un'app Android, chiamata HideDroid. Testando con HideDroid le prime 2000 applicazioni più scaricate dal Google Play Store, è stato possibile valutare le performance dell'app e la sua applicabilità in uno scenario realistico. I risultati ottenuti, hanno consentito di eseguire un'analisi più approfondita dell'uso che gli sviluppatori fanno dei servizi di API di analytics, all'interno delle proprie applicazioni, e di quali informazioni vengono estratte più frequentemente. Dai risultati ottenuti, si è constatato, che il 90% delle richieste sono indirizzate a servizi terzi di analytics, che permettono agli sviluppatori di raccogliere ingenti quantità di informazioni personali e sul dispositivo. Esaminando le librerie di analytics appartenenti ad aziende, come Facebook, si è potuto, inoltre, quantificare e identificare il numero di eventi che possono essere collezionati durante l'esecuzione delle applicazioni, e di come questi dati possano rappresentare un pericolo per la re-identificazione degli stessi utenti. Questi ultimi, risultando, così, completamente ignari di quali informazioni vengano estratte durante l'utilizzo delle loro applicazioni, non hanno la possibilità di scegliere quali dati vengano prelevati. La novità di MobHide, e della sua implementazione HideDroid, risiede nella realizzazione di un servizio di anonimizzazione completamente integrato all'interno del dispositivo, che non necessita di alcuna interazione esterna con servizi terzi, mantenendo, così, i dati sempre nelle mani del legittimo proprietario. A differenza degli approcci attualmente esistenti la privacy sui dati dell'utente viene riposta nuovamente al centro dell'attenzione permettendo al vero proprietario dei dati di poterli gestire, garantendo, a sua discrezione, una certa utilità dei dati stessi. Inoltre, il suo utilizzo non comporta modifiche del dispositivo o l'obbligo di acquisire i permessi di root, risultando, così, di semplice accesso per gli utenti finali. Hidedroid rappresenta, quindi, una soluzione innovativa, non ancora presa in considerazione dai paper scientifici, che, attualmente, risiedono allo stato dell'arte. Si ritiene che il suo sviluppo rappresenti quindi un importante passo avanti all'interno della data privacy nel mondo mobile.
Gli sviluppi futuri che è possibile prospettare dal seguente elaborato, sono:

1. Un miglioramento del servizio di anonimizzazione, con l'adozione di tecniche più

sofisticate, che si adattino alla natura dei dati individuati, al fine di ridurre, così, il numero di richieste rifiutate da parte dei server di analytics.

2. L'implementazione di un servizio di intercettazione delle richieste che consenta una loro bufferizzazione, utile alla realizzazione del punto 1.

3. La possibilità di esportare l'applicazione HideDroid in ambiente *iOS*.

4. Utilizzare un ambiente virtuale, all'interno del dispositivo, come DroidPlugin [30] e VirtualApp [31], che consentono di evitare gli step di repackaging o di installazione dei certificati superando alcune limitazioni della soluzione proposta in questo lavoro di tesi.

5. L'implementazione di algoritmi in grado di poter decodificare e, quindi, anonimizzare richieste in formato *protobuf* [28] di Google Firebase Analytics.

Riferimenti bibliografici

[1] Exodus, "Most frequent trackers - google play," https://reports.exodus-privacy.eu.org/en/trackers/stats/.

[2] Y. He, X. Yang, B. Hu, and W. Wang, *Dynamic privacy leakage analysis of android third-party libraries.* Journal of Information Security and Applications, 2019.

[3] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, *Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem.* IEEE Transactions on Mobile Computing, 2020.

[4] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, *Information leakage through mobile analytics services.* Proceedings of the 15th Workshop on Mobile Computing Systems and Applications, 2014.

[5] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, *Investigating user privacy in android ad libraries.*

[6] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, *On the (Un)Reliability of Privacy Policies in Android Apps.* Proc.of the IEEE International Joint Conference on Neural Networks, 2020.

[7] P. Samarati and L. Sweeney, *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression.* SRI International, 1998.

[8] C. Dwork and A. Roth, *The algorithmic foundations of dierential privacy.* Foundations and Trends® in Theoretical Computer Science, 2014.

[9] C. Ferrara, "Quanto è diffuso https?" https://www.html.it/19/12/2019/quanto-e-diffuso-https/, 2019.

[10] A. Possemato and Y. Fratantonio, *Towards HTTPS Everywhere on Android: We Are Not There Yet.* Proceedings of the 29th USENIX Security Symposium, 2020.

[11] Wikipedia, "Certificate authority," https://en.wikipedia.org/wiki/Certificate_authority, Last updated 18-11-2020.

[12] N. Vallina-Rodriguez, S. Sundaresan, A. Razaghpanah, R. Nithyanand, M. Allman, C. Kreibich, and P. Gill, *Tracking the trackers: Towards understanding the mobile advertising and tracking ecosystem.* arXiv preprint arXiv:1609.07190, 2016.

[13] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, *MockDroid: trading privacy for application functionality on smartphones.* Proceedings of the 12th Workshop on Mobile Computing Systems and Applications. HotMobile '11, Association for Computing Machinery, New York, NY, USA, 2011.

[14] H. Zhang, S. Latif, R. Bassily, and A. Rountev, *Privaid: Differentially-private event frequency analysis for google analytics in android apps.* Ohio State University, Columbus, Ohio, USA, 2018.

[15] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill, *Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem.* Network and Distributed Systems Security (NDSS) Symposium, 2018.

[16] L. Sweeney, *Achieving k-anonymity privacy protection using generalization and suppression.* School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2002.

[17] A. Developers, "Android 6.0 marshmallow," https://www.android.com/intl/it_it/versions/marshmallow-6-0/, 2015.

[18] A. 10, "Android 10," https://www.android.com/intl/it_it/android-10/, 2019.

[19] Wikipedia, "Rooting (android)," https://en.wikipedia.org/wiki/Rooting_(Android), Last updated 7-11-2020.

[20] S. Overflow, "What is smali code android," https://stackoverflow.com/questions/30837450/what-is-smali-code-android, 2017.

[21] A. Developers, "Vpnservice," https://developer.android.com/reference/android/net/VpnService, Last updated 2020-09-30.

[22] Wikipedia, "Lz77 and lz78," https://en.wikipedia.org/wiki/LZ77_and_LZ78#LZ77.

[23] A. Enterprises and J.-L. Gailly, *RFC 1950: ZLIB Compressed Data Format Specification version 3.3.* IETF, May 1996.

[24] A. Enterprises, *RFC 1951: DEFLATE Compressed Data Format Specification version 1.3.* IETF, May 1996.

[25] Y. Li, Z. Yang, Y. Guo, and X. Chen, *DroidBot: a lightweight UI-Guided test input generator for android.* IEEE, 2017.

[26] A. Developer, "Android debug bridge (adb)," https://developer.android.com/studio/command-line/adb, Last updated 2020-09-10.

[27] S. Rastogi, K. Bhushan, and B. B. Gupta, *Android applications repackaging detection techniques for smartphone devices.* Dept. of Computer Engg., National Institute of Technology Kurukshetra, Kurukshetra, 2015.

[28] G. Developers, "Protocol buffers," https://developers.google.com/protocol-buffers.

[29] A. Developers, "Protect against security threats with safetynet," https://developer.android.com/training/safetynet, Last updated 27-12-2019.

[30] cmzy, "Droidplugin," https://github.com/DroidPluginTeam/DroidPlugin, 2015.

[31] asLody, "Virtualapp," https://github.com/asLody/VirtualApp, 2016.