

UNIVERSITÀ TOR VERGATA

CORSO DI LAUREA DI INGEGNERIA INFORMATICA

A.A. 2015-2016

Mobile Sniffing Tool

Progetto di Sicurezza Informatica e Internet

Autori:

Paolo SALOMÉ

Stefano AGOSTINI

Supervisori:

Prof. Giuseppe ITALIANO

Dr. Marco QUERINI

January 18, 2017

Contents

1	Introduzione	2
2	Descrizione dell'Applicazione	2
2.1	Tcpdump	2
2.2	Struttura APK	4
2.2.1	Activity di cattura	4
2.2.2	Activity di filtraggio	6
2.2.3	Service di cattura	7
2.2.4	Service di filtraggio	8
3	Test	9
3.1	Test sulle applicazioni di messaggistica	9
3.2	Test integrativi sulle modalità di utilizzo della scheda di rete	10
4	Conclusioni e sviluppi futuri	13

1 Introduzione

Nella società odierna vi é un larga diffusione di applicazioni mobile di messaggistica che espongono implicitamente gli utenti a problematiche riguardanti la privacy. Infatti fino a qualche mese fa i dati in transito di Whatsapp (e.g.) erano in chiaro e quindi accedibili facilmente da chiunque utilizzasse una qualsiasi applicazione per lo sniffing (e.g. Whireshark). Tuttavia attualmente molte delle applicazioni di messaggistica hanno rimediato interamente o parzialmente utilizzando tecniche di crittografia end-to-end. L'obiettivo del nostro progetto é la realizzazione di una applicazione per sistemi mobile Android che all'interno di una certa rete wifi catturi i dati in transito e li filtri in base ad un flag. In tal modo ci preponiamo l'obiettivo di isolare i pacchetti dati per ogni applicazione di messaggistica presa in considerazione e analizzarli.

2 Descrizione dell'Applicazione

Per realizzare uno *sniffer Android* c'erano due strade alternative da poter seguire: sfruttare le *Android VPN* oppure utilizzare l'eseguibile *C tcpdump*. L'approccio *Android VPN* consiste nella creazione di un'interfaccia *VPNService*, gestita da una applicazione in *userspace*, che una volta attivata forza tutto il traffico del *device* ad attraversarla. Tuttavia questo approccio non permette la cattura di pacchetti appartenenti a dispositivi diversi da quello ospitante l'applicazione. Per utilizzare il secondo approccio é necessario che il dispositivo abbia i permessi di *root* in quanto verrà eseguito uno *script bash tcpdump*, il quale si occuperá di catturare i pacchetti. Inoltre questa libreria permette di sfruttare la scheda di rete in uso in varie modalità (e.g. *promiscuous mode*, *monitor mode*) qualora il dispositivo lo consenta. La possibilità di sfruttare a pieno le potenzialità della scheda di rete ci ha indotto a scegliere *tcpdump*.

2.1 Tcpdump

Tcpdump é un eseguibile *command-line* basato su *libpcap*, una libreria *C* portabile che offre *API* utili per la cattura del traffico di rete. Abbiamo deciso di utilizzare *tcpdump* anziché direttamente le *API* di *libpcap* poiché esso si presta maggiormente

all'esecuzione su piattaforma *Android* (si esegue direttamente su *bash*).

Il comando *bash* dell'eseguibile *tcpdump* permette l'inserimento di alcuni parametri che consentono la visualizzazione dei pacchetti in vari formati. Nel caso della nostra applicazione utilizziamo i seguenti parametri:

- **-i** : seguito dal nome dell'interfaccia, per specificare dove porsi in ascolto (e.g. wlan0)
- **-XX** in alternativa a **-A**: il primo stampa l'header di ogni pacchetto e i dati in esadecimale e *ASCII* mentre il secondo non stampa l'esadecimale
- **-tttt**: stampa la data corrente davanti l'header di ogni pacchetto in formato *YYYY-MM-DD hh:mm:ss:dddddd*

Di seguito inseriamo un pacchetto di esempio stampato da *bash* come risultato del comando *sudo tcpdump -i wlan0 -XX -tttt* (figure 1) e del comando *sudo tcpdump -i wlan0 -A -tttt* (figure 2).

```
2017-01-11 17:22:59.118063 IP 192.168.1.107.42883 > 74.125.143.127.19305: UDP, length 122
0x0000: 001e e594 eb56 18f4 6ac1 f394 0800 4500 .....V..j.....E.
0x0010: 0096 1ec5 4000 4011 7f82 c0a8 016b 4a7d ....@.@.....kJ}
0x0020: 8f7f a783 4b69 0082 e2b6 906f 7a48 0392 ....Ki.....ozH..
0x0030: d930 2a8d 9dfd bede 0002 1094 323e ccd9 .0*.....2>..
0x0040: 0000 493a 5b44 98d7 4469 b75a 9dcd e0c7 ..I:[D..Di.Z....
0x0050: 0c5f 5630 d811 794b e8c5 886d 671a 75ed ._V0..yK...mg.u.
0x0060: 1cf0 426a cccb 62fd ac76 67da bfe1 8ce6 ..Bj..b..vg.....
0x0070: 136e bb7f 34a3 8380 a83a 2c52 f910 9029 .n..4.....:R...)
0x0080: 1513 f195 4a84 e9e8 a270 2162 7bad b473 ....J....p!b{..s
0x0090: c645 65ea dba6 930a 2149 7654 fe71 b4d3 .Ee.....!IvT.q..
0x00a0: 7b02 efe4 {...
```

Figure 1: Con opzione -XX

```
2017-01-11 17:35:03.273751 IP 192.168.1.107.42883 > 74.125.143.127.19305: UDP, length 1203
E....v@.@.....kJ}....Ki...@.d.....{IV.H....2.lM...q.9.... ..;-i_...|.UU5P.....=....
...a;<=.....$.-e.....f[V]K.<...-;VW-LiXd.@`_.....0....^N!?W...qb'...lJ.....f0
.Mt..+/n[...g....Y.F....q.o.....l.../. ...^..2....F7....N<.7d.>.....}|... ..c0.
....V.$5..m..U..~.6_..i,0
..D...T...;`.y.3.0.p[.....[*...+.....N5....V....l2.z.G..er0.LtE..S..=.....R.7....
..C...L~K9./.....#.e.`.%8s...8a.oNc.\...70....8~....".^L...b.%W..~.'.'0U3X.-...v`. "-o
.Ch.f..lu.....jx.l.`..qP.&...3.....q.<....w...e...Z..q.....%n..U.R.Wl.#&y../
..*.0...v!B.....'<.->.....`.8....ls.o1Ha..|..".f.F....T...[.....f5.....;.....
..].u.....}.K_b..6y.....4..%..).7V.....SF./^.....{"...GZ)..h.[B....g+.G.p..
```

Figure 2: Con opzione -A

2.2 Struttura APK

La nostra *APK* si compone di due *Activity* e due *Service*:

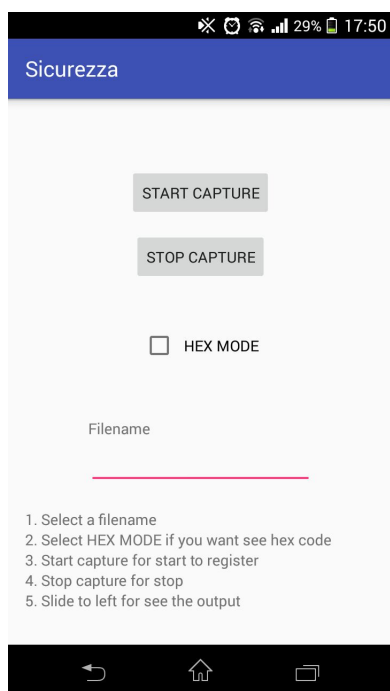
- L'*Activity* di cattura espone un'interfaccia semplice per impostare la modalità di cattura (specificare il *flag hex mode*). Successivamente l'utente può specificare il nome del file sul quale vuole che vengano salvati i pacchetti. Infine possibile attivare il servizio di cattura.
- L'*Activity* di filtraggio si occupa di fornire all'utente un elenco dei file contenenti i pacchetti, salvati dall'applicazione stessa nelle precedenti catture. Selezionato il file da elenco possibile inserire la parola chiave di ricerca. Mediante il servizio di filtraggio é vengono scanditi i pacchetti e visualizzati in una lista soltanto quelli contenenti la parola desiderata.
- Il *Service* di cattura si occupa dell'invocazione del comando *tcpdump* con le opzioni e il nome del file passati dall'*Activity* di cattura, redirezionando l'output su quest'ultimo. Quando questo servizio viene interrotto si esegue il comando bash *pskill tcpdump* per terminare il comando dello *sniffer*.
- Il *Service* di filtraggio agisce sul file selezionato esaminando ogni pacchetto e inserendolo in una lista visualizzata a schermo solo se contiene la *keyword* fornita, all'interno dell'*header* o del *body*.

Di seguito inseriamo degli *screen* dell'applicazione appena descritta (figure 3).

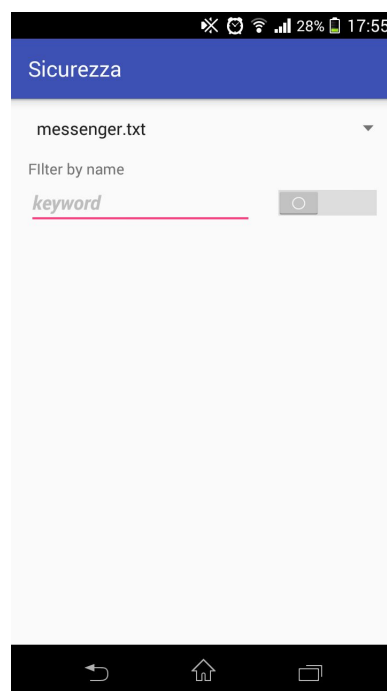
2.2.1 Activity di cattura

L'*Activity* di cattura implementata dalla classe *MainActivity* e realizza la schermata iniziale per l'utente. Essa estende la classe *AppCompatActivity* ed effettua l'*override* dei metodi *onCreate* e *onTouchEvent*. Quest'ultimo implementa un *listener* che, a seguito del primo tocco dell'utente, esegue le seguenti operazioni:

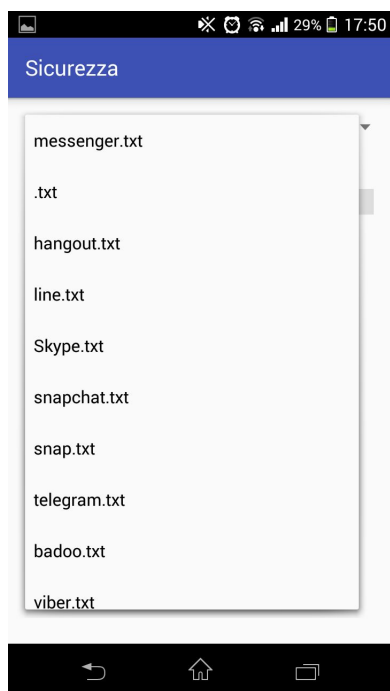
- Salva la posizione iniziale al tocco.
- Al rilascio calcola il *delta* rispetto alla posizione iniziale.
- Se il *delta* é maggiore di 400 *pixel* verso sinistra invoca l'*Activity* di filtraggio.



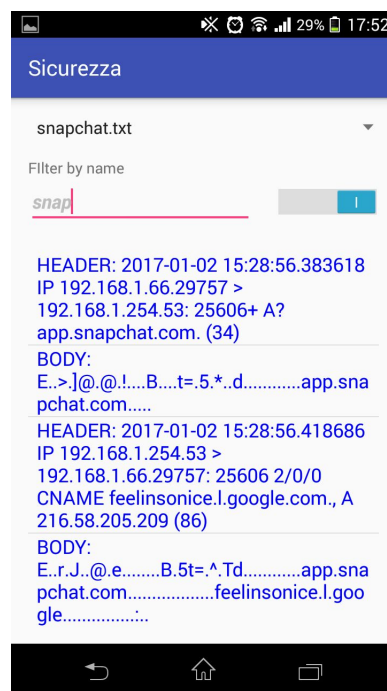
(a) Activity di cattura



(b) Activity di filtraggio



(c) Lista di file



(d) Pacchetti filtrati

Figure 3: Pagine dell'applicazione

Infine ci sono i due metodi legati ai bottoni di *start* e *stop* del servizio di cattura. Il metodo *startclick* esegue i seguenti passi all'invocazione:

- Cerca il riferimento all'*editText* contenente il nome del file in cui salvare i pacchetti catturati.
- Verifica lo stato della *checkbox* relativa alla modalità *hex mode*.
- Crea l'*Intent* da passare al servizio di cattura.
- Utilizza il metodo *putExtra* dell'*Intent* per passare i parametri di cattura al servizio (*path* del file e eventualmente il *flag hex*).

Il metodo *stopclick* crea un nuovo *Intent* per il servizio di cattura avviato precedentemente e invoca il metodo *stopService* per forzarne l'interruzione.

2.2.2 Activity di filtraggio

Questa *Activity* é implementata dalla classe *SecondActivity* la quale estende la classe *AppCompatActivity*. Il metodo *OnCreate* effettua le seguenti operazioni:

- Recupera lo *switch* relativo all'attivazione del servizio di filtraggio.
- Inizializza lo stato dello *switch* a *false*.
- Fa l'*override* del metodo *OnCheckedChanged*, il quale rappresenta il *listener* dello *switch*. Quest'ultimo:
 - Se lo *switch* é *checked* inizializza la *ListView* che apparirá a schermo e l'*Adapter* ad essa associato. Successivamente invoca il metodo *viewFile* di cui parleremo in seguito.
 - Se lo *switch* é *unchecked* invoca il metodo *stopServ*.
- Preleva tutti i file relativi alle precedenti registrazioni di pacchetti fatti dall'applicazione. I file sono raggruppati in una cartella sul dispositivo.
- Prelevati i file questi vengono inseriti in una lista di stringhe. Quest'ultima verrá associata con un *adapter* ad uno *spinner* per consentire all'utente la selezione di un file.

Il metodo *viewFile* effettua le seguenti operazioni:

- Preleva il nome del file dallo *Spinner* ed il filtro.
- Crea l'*Intent* relativo al servizio di filtraggio ed aggiunge come parametri il *filepath* ed il *filter*.
- Invoca il metodo *startService* con parametro l'*Intent* appena creato.

Il metodo *stopServ* crea un nuovo *Intent* per il servizio di filtraggio avviato precedentemente e invoca il metodo *stopService* per forzarne l'interruzione.

Questa *Activity* utilizza un *BroadcastReceiver* per ricevere i dati elaborati dal servizio invocato. Di questo *Receiver* viene fatto l'*override* del metodo *onReceive* il quale, in ricezione dei dati inviati dal servizio, aggiorna la *ListView* presente nell'*Activity* stessa. Per utilizzare il *BroadcastReceiver* col metodo *onReceive* esso deve essere registrato mediante il metodo *onResume*, il quale accetta un *flag* per identificare gli *Intent* da gestire (nel nostro sistema *filterMessage*). D'altro canto il metodo *onPause* serve a deregistrare il *BroadcastReceiver* quando l'*Activity* é in pausa.

Infine é implementato il metodo *onTouchEvent* che si comporta esattamente come quello dell'*Activity* precedentemente illustrata.

2.2.3 Service di cattura

Questo servizio é implementato dalla classe *RegisterService*. Nel metodo *onStartCommand* viene avviato lo *sniffer* in seguito ai seguenti passi:

- Vengono estratti dall'*Intent* passato dalla *MainActivity* il *path* del file ed eventualmente il *flag* contenente l'*hex mode*.
- Vengono preparati i comandi da eseguire su *bash* ovvero **su** per garantire i dovuti permessi di *superuser* e il comando **tcpdump**.
- Viene creato ed eseguito un processo *bash*.
- Viene effettuato un collegamento con l'*OutputStream* del processo e vengono scritti i comandi da eseguire preparati in precedenza.

- Al termine della scrittura viene opportunamente chiuso il collegamento con l'*OutputStream* del processo.

Il metodo *onDestroy* ha una struttura simile al metodo *onStartCommand*, con la differenza nei comandi da eseguire su *bash*. Esegue la terminazione dei processi precedentemente avviati tramite il comando **kill tcpdump**. Infine termina il servizio tramite la chiamata al metodo *this.stopSelf*.

2.2.4 Service di filtraggio

La classe *FilterService*, a differenza del *Service* di cattura, è un *IntentService*. Un *Service* di questo tipo riesce ad eseguire richieste asincrone fornite dal *Client* tramite un suo *working thread*, diverso dal *main thread* dell'applicazione. In tal modo si alleggerisce del lavoro il *main thread* il quale può nel frattempo occuparsi di altro. Inoltre un *IntentService* è in grado di eseguire più richieste, accodandole una dopo l'altra.

Abbiamo deciso di utilizzare questa soluzione poiché filtrare file di grandi dimensioni (come il caso di file contenenti pacchetti dati) avrebbe potuto causare attese molto lunghe e nei casi peggiori ledere la stabilità dell'applicazione, causando persino il *crash*.

Il metodo *onHandleIntent* si preoccupa di innescare il *working thread* all'arrivo dell'*Intent*, il quale esegue i seguenti passi:

- Recupera dall'*Intent* inviato dall'*Activity* di filtraggio il *path* del file e il *filter*.
- Apre il file per collegarlo ad un oggetto *BufferedReader*.
- Vengono definite due stringhe utili ad identificare l'*header* di ogni pacchetto: esse sono "*IP*" e "*ARP*" poiché qualsiasi pacchetto nell'*header* ha per identificativo anche il relativo protocollo di rete. Inoltre vengono inizializzate come stringhe vuote *header* e *body* utili a contenere l'*header* e il *body* del pacchetto corrente.
- Viene letto il file riga per riga ed ognuna di queste viene concatenata alla stringa *header* se contiene "*IP*" o "*ARP*" e nel caso contrario viene concatenata alla stringa *body*.

- Ogni volta che viene identificato un nuovo *header* vengono controllate le stringhe *header* e *body* relative al pacchetto precedente e, nel caso contengano la stringa *filter*, allora verrà invocata la *sendBroadcast* con queste come parametri. Dopodiché viene azzerata la stringa *body* e assegnato l'*header* appena identificato alla stringa *header*.
- Al termine del file viene chiuso il *BufferedReader*.

Il metodo *sendBroadcast* crea un *Intent* identificato dalla stringa "*filterMessage*" (in modo da essere gestito dall'*Activity* di filtraggio) e mediante il suo metodo *putExtra* vi allega i parametri *header* e *packet*. A questo punto invia l'*Intent* tramite il *LocalBroadcastManager* il quale, a differenza della versione globale, permette di non disperdere i messaggi al di fuori dell'applicazione stessa. Tutto ciò offre vantaggi in termini di *privacy* (poiché questi dati non sono visibili alle altre *app*), in termini di prestazioni per lo stesso motivo e in termini di sicurezza poiché le altre applicazioni non possono inviare dati malevoli nella nostra *app*.

Il metodo *onDestroy* quando invocato termina il servizio.

3 Test

3.1 Test sulle applicazioni di messaggistica

È stato selezionato un campione di dieci applicazioni di messaggistica per valutarne il grado di sicurezza percepito in seguito alla lettura dei pacchetti catturati mediante la nostra applicazione (identificazione di possibili *security flaws*). Di seguito elenchiamo le dieci applicazioni da noi testate: *Whatsapp*, *Telegram*, *Snapchat*, *Badoo*, *Instagram*, *Viber*, *Messenger(Facebook)*, *Hangout*, *Line*, *Skype*. Per ognuna di queste *app* abbiamo scambiato dei messaggi ed in parallelo eseguito lo *sniffing* per mezzo della nostra applicazione. Al termine di ogni cattura, per esaminare il contenuto dei pacchetti, abbiamo usufruito del filtro precedentemente descritto, messo a disposizione dalla nostra applicazione. In questo modo abbiamo provato a cercare frammenti di conversazione tra i pacchetti, senza avere alcun risultato positivo da nessuna delle applicazioni in quanto tutti i pacchetti sono risultati cifrati. Tuttavia la metà delle *app* presenta dei pacchetti riconoscibili tramite il nome stesso dell'applicazione (presente a volte nell'*header* e altre nel

body). Di seguito verranno mostrati degli *screen* relativi alle applicazioni i cui pacchetti sono stati riconosciuti (figure 4 e 5).

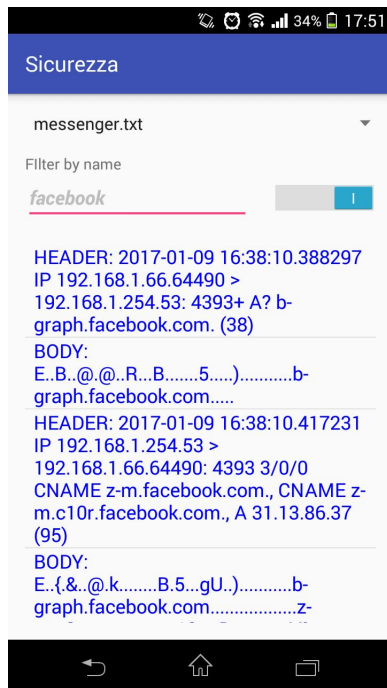


Figure 4: Applicazioni Analizzate (1)

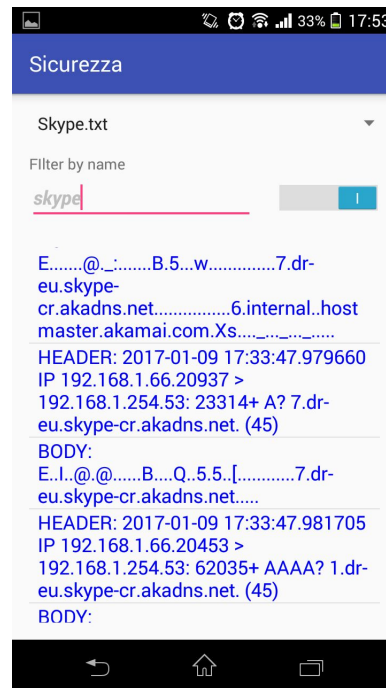
Inoltre nella tabella seguente vengono riassunti per ogni *app* i *security flaws* relativi all'*header* ed al *body* dei pacchetti (figure 6).

3.2 Test integrativi sulle modalità di utilizzo della scheda di rete

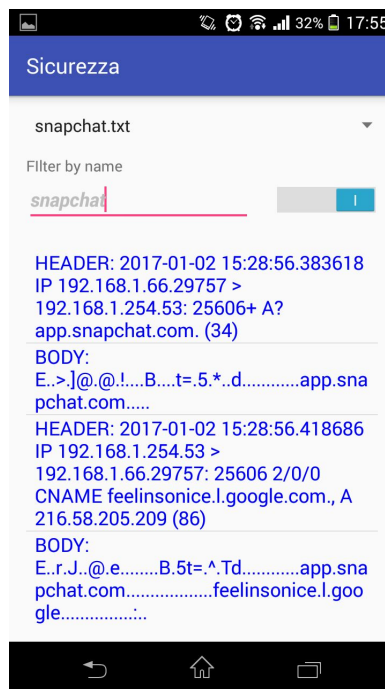
Per illustrare le differenze riguardanti la cattura dei pacchetti nell'utilizzo delle diverse modalità di funzionamento della scheda di rete abbiamo effettuato test ulteriori su una piattaforma *Linux* (*Linux Mint 17.1 Cinnamon-64 bit*). Abbiamo condotto i test utilizzando la modalità *promiscua* e la modalità *monitor*. La



(a) Pacchetto Messenger



(b) Pacchetto Skype



(c) Pacchetto Snapchat

Figure 5: Applicazioni Analizzate (2)

App	Header	Payload
Whatsapp	No security flaws	No security flaws
Telegram	No security flaws	No security flaws
Snapchat	Riconoscibile	Riconoscibile
Badoo	Riconoscibile	Riconoscibile
Instagram	Riconoscibile	Riconoscibile
Viber	No security flaws	No security flaws
Messenger / Facebook	Riconoscibile	Riconoscibile
Hangout	No security flaws	No security flaws
Line	No security flaws	No security flaws
Skype	Riconoscibile	Riconoscibile

Figure 6: Tabella delle applicazioni

modalità promiscua è una modalità di controllo in cui l'interfaccia di rete cablata o wireless filtra tutto il traffico di rete che l'unità centrale (CPU) riesce ad analizzare; la sostanziale differenza è che la modalità non promiscua analizza soltanto il frame che riceve o che destina. Adottando tale modalità siamo stati in grado di catturare un sottoinsieme dei pacchetti *multicast* identificati come appartenenti al protocollo *IGMP*, destinati ad *IP* della *LAN* differenti dall'*host tcpdump*. D'altro canto la modalità *monitor* permette la cattura di pacchetti della rete senza il bisogno di essere associati ad un *access point* ma si applica esclusivamente a reti *wireless*. Nel nostro test abbiamo notato una formattazione dei pacchetti totalmente differente da quella adottata dalla modalità *promiscua* che non ci ha permesso di isolare pacchetti appartenenti ad altri *IP*. Un'altra motivazione che ci ha indotto a preferire la prima modalità è legata al fatto che, nella modalità *monitor*, il *wireless adapter* non controlla il *CRC* dei pacchetti catturati e pertanto alcuni di questi può facilmente risultare corrotto, impedendo una corretta lettura.

4 Conclusioni e sviluppi futuri

Sebbene la maggior parte delle applicazioni di messaggistica abbia fatto ricorso ad una forma di crittazione, alcune di esse presentano ancora alcune lacune in ambito di sicurezza. Infatti, come da noi dimostrato, *app* come ad esempio *Snapchat*, *Badoo* e *Instagram* espongono i propri pacchetti dati ad un elevato grado di riconoscibilità. Un ipotetico attaccante, sebbene abbia un compito non banale quale la decrittazione di pacchetti, potrebbe pertanto distinguere da una grande mole di dati il suo target in maniera molto più immediata ed intuitiva. Tutto ciò dimostra che potrebbe essere necessario estendere la crittazione anche a porzioni del pacchetto che non rappresentano il *payload*.

Per quanto riguarda lo *sniffer*, un utilizzo più esteso delle opzioni della libreria *tcpdump* potrebbe portare a delle migliorie in termini di personalizzazione della cattura. *Tcpdump* infatti permette di combinare vari *flag* in modo da ottenere una visualizzazione pacchetti il più vicina possibile al formato desiderato dall'utente. Invece dal punto di vista dell'applicazione *Android* bisognerebbe apportare modifiche all'interfaccia grafica dell'*Activity* di filtraggio ed aggiungervi opzioni utili ad evidenziare porzioni desiderate della lista dei pacchetti.

References

- [1] <http://www.androidtcpdump.com>
- [2] <http://www.tcpdump.org>
- [3] <https://developer.android.com>
- [4] <https://en.wikipedia.org>