UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato d'esame

Reti di Calcolatori I

Raccolta e analisi di tracce di traffico di applicazioni mobili

Anno Accademico 2018/2019

Professore

Prof. Antonio Pescapè

Gruppo – Dotani/Ferrara - 30

Claudio Dotani
Matr. N46002878

Giuseppe Ferrara
Matr. N46002766

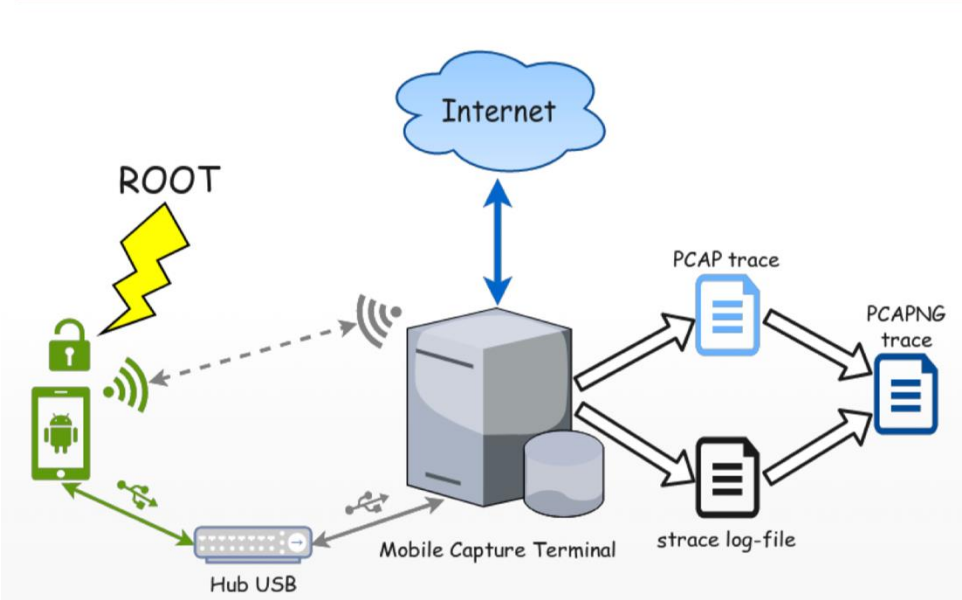
Indice

Reti di Calcolatori I

Sommario

*Il traffico generato dalle applicazioni Onefootball, Foursquare, Waze e Wish è stato classificato con l'aiuto del software **TIE**, che sfrutta algoritmi di machine learning.*
In particolare è stato utilizzato un plugin per effettuare la classificazione sulla base del contenuto del payload dei pacchetti catturati.
Il risultato di questa classificazione è stato poi arricchito con l'aiuto di uno script e ne sono stati analizzati i dati generati.

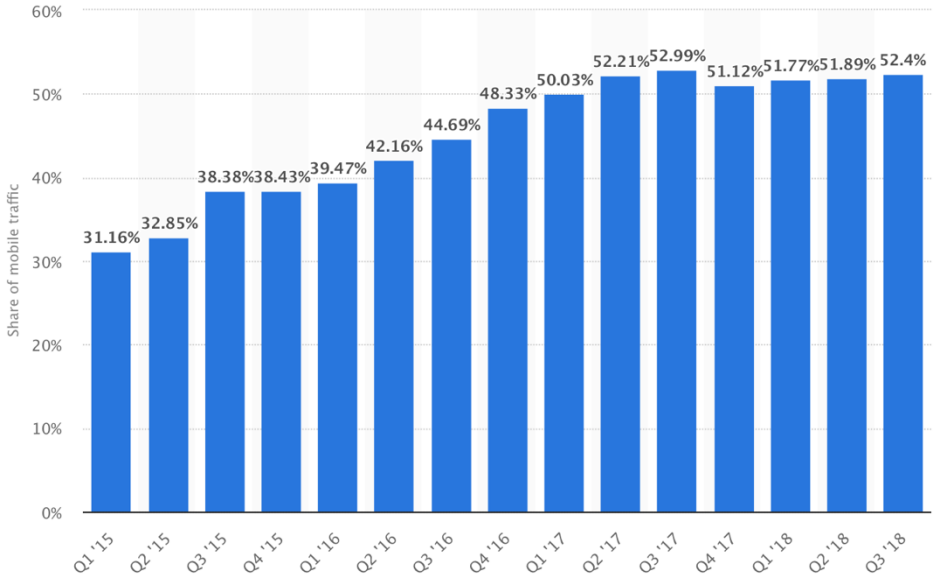
Capitolo 1: Cattura e classificazione del traffico



Durante la fase di cattura del traffico, che verrà descritta in dettaglio in seguito, è stato simulato un utilizzo normale delle applicazioni al fine di generare i file necessari per la successiva classificazione.
È stato utilizzato il laboratorio di cattura del traffico mobile (ARCLAB) messo a disposizione dall'Università che è dotato di una postazione descritta schematicamente in figura.

Il traffico mobile è in costante crescita negli ultimi anni e continuerà sicuramente a crescere.

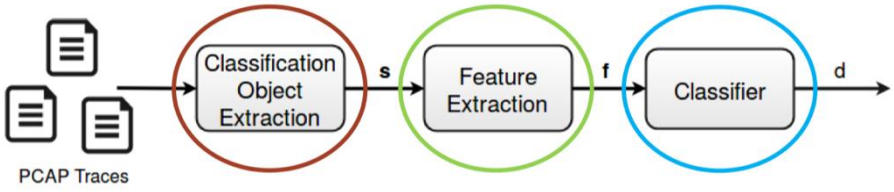
L'analisi e la classificazione del traffico è una tecnica utilizzata per studiare il flusso di dati scambiati da una certa applicazione mobile e verificarne la natura. Quest'analisi viene spesse volte effettuata da compagnie che si occupano di analisi di dati o di pubblicità, ma anche da assicurazioni e agenzie di sicurezza. In particolare la classificazione del traffico mobile può avvenire con diverse tecniche, ad esempio mediante algoritmi di machine e deep learning o mediante analisi del contenuto del pacchetto e può essere effettuata contestualmente alla cattura (on the fly) o successivamente.



Le prime non sono però esaustive perché è necessario limitare la quantità di elaborazioni da effettuare e sono limitate dall'hardware utilizzato.
Le classificazioni effettuate successivamente alla cattura invece sono di diverse tipologie e si basano sull'analisi di biflussi relative alle quintuple. Vengono analizzati questi dati perché data la mole di dati scambiati tipicamente sarebbe molto oneroso classificare ogni singolo pacchetto ma ci si limita a classificare un gruppo di pacchetti caratterizzati dalla stessa quintupla e associarli per comprendere la natura di questo traffico e l'applicazione che l'ha generato.

La classificazione basata sul DPI (Deep Packet Inspection) consiste nella lettura del contenuto del pacchetto, ma a causa della crescita del volume del traffico cifrato è una tecnica ormai poco utilizzabile.

Le tecniche basate su machine learning invece si basano sulla costruzione di una Ground Truth (essa può essere costruita in base all'analisi del payload, sulla base di euristiche o manualmente da una o gruppi di persone) e sull'analisi del traffico confrontandolo con essa. Anche questa tecnica comporta però problemi sia in caso di traffico cifrato che in caso di scarso traffico di Ground Truth in relazione alla differenza di applicazioni e del tipo di traffico.

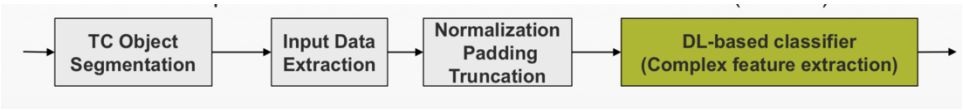


Le tecniche basate sul Deep Learning consentono di ridurre questo problema utilizzando una Feature Extractor gerarchica.
Tali tecniche sono inoltre computazionalmente meno onerose delle tecniche precedenti.

Traditional Machine Learning Flow

Deep Learning Flow

Di seguito è illustrato come avviene l'estrazione, il DL-based-classifier svolge la stessa funzione del Feature Extractor utilizzato nelle tecniche di machine learning.



Capitolo 2: Strumenti utilizzati

Sono stati utilizzati dei dispositivi Android dotati di permessi di root collegati mediante un Hub USB ad una macchina che, utilizzando un software di cattura, ha catturato il traffico dati generato dalle applicazioni salvandolo nel file ***traffic.pcap*** *e ha catturato anche tutte le chiamate a sistema effettuate dalle applicazioni, tra le cui sono state di interesse le richieste di apertura delle socket, salvandole nel file* ***strace.log.***

Per effettuare analisi e classificazione del traffico catturato sono stati utilizzati vari software.

In un primo momento, il software Wireshark, che consente di visualizzare le tracce di traffico ed eventualmente di applicare dei filtri per la visualizzazione dei parametri.

Il software TIE, sviluppato da W. de Donato, A. Pescapè, A. Dainotti, effettua una classificazione del traffico mobile utilizzando algoritmi di machine e deep learning.

In particolare è stato utilizzato con il plug-in ***ndping_1.0*** *che effettua la classificazione sulla base dell'analisi del payload dei pacchetti (Deep Packet Inspection).*

Il problema dell'utilizzo di questo software è che buona parte del traffico è cifrato, in questo caso TIE riesce a fornire una prima classificazione sulla sola base della firma dell'applicazione (se nota) o del protocollo di trasporto utilizzato.

In questo caso è necessario dunque o utilizzare altre tecniche o integrare l'output della classificazione del traffico fornita da TIE con altre informazioni tra cui il package che ha richiesto l'apertura della socket a livello di trasporto, gli indirizzi IP forniti in risposta alle query DNS effettuati dalle applicazioni e, mediante risoluzione DNS inversa e interrogazione whois, analizzare le informazioni per classificare il traffico e confrontarlo con la classificazione effettuata da TIE

Per la realizzazione dello script è stato necessario utilizzare alcuni comandi del terminale linux come tshark, versione da terminale di wireshark, il comando dig che consente di eseguire la risoluzione dns inversa e il comando whois, che è stato sfruttato per poter identificare l'entità che gestiva un dato indirizzo IP.

Capitolo 3: Cattura e analisi del traffico mobile

3.1 Cattura del traffico

È stata effettuata la cattura del traffico generato dalle seguenti applicazioni:

- Fo*urs*quare
- OneFootball
- Waze
- Wish

Il traffico generato è stato catturato in due giorni differenti e ogni giorno per una durata complessiva di 60 minuti sono state effettuate 6 catture da 5 minuti per ciascuna applicazione.
Durante questo breve periodo è improbabile che uno stesso porto sorgente sia stato utilizzato da due applicazioni differenti, ciò consente di affermare l'unicità della quintupla, con buona approssimazione durante tutta la cattura.
Durante i primi 5 minuti per ciascuna applicazione è stata effettuato il download, il login, l'installazione e, per quanto possibile, un utilizzo dell'applicazione.
Per il tempo restante è stato simulato un uso normale dell'applicazione.

3.2 Script per l'automazione delle analisi

Per integrare la classificazione effettuata da TIE è stato sviluppato uno script in linguaggio python per effettuare le operazioni automatiche di correzione.
Le operazioni in questione sono l'aggiunta delle colonne con le informazioni relative ad ogni quintupla del package, del dominio del quale è stata chiesta una risoluzione DNS, della risoluzione DNS inversa degli indirizzi IP forniti dal DNS, dell'interrogazione whois, di eventuali handshake SSL e di pacchetti HTTP.
Lo script è stato realizzato in python sfruttando la libreria os per alcune chiamate di sistema e la libreria csv per la gestione dei file formattati in tal modo, come il file generato da TIE e la libreria sys per prendere come argomento il nome del file da classificare per evitare di andare a modificare lo script e renderlo più generale possibile.
*Lo script deve essere eseguito con il comando **python3 script.py NOMEFILE** specificando al posto di NOMEFILE il nome del file generato da TIE senza l'estensione.*
*Lo script produrrà automaticamente un file con lo stesso nome e con estensione **.gt.tie.***
*Inizialmente lo script chiama dalla shell tre comandi di **tshark** e genera dei file di testo, a partire dal file **traffic.pcap** che saranno utilizzati poi per la ricerca dei campi DNS, SSL handshake e HTTP.*

```
#genero il file per le query DNS
#considero solo i campi ip e nome host
os.system("tshark -r traffic.pcap -T fields -e dns.a -e dns.qry.name -Y\"(dns.flags.response == 1)\" > trafficpcap.txt")
#genero il file per il campo SSL
os.system("tshark -r traffic.pcap -T fields -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport -e ssl.handshake.extensions_server_name -Y
ssl.handshake.extensions_server_name > ssl.txt")
#genero il comando per il campo http
os.system("tshark -r traffic.pcap -T fields -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport -e http.host -Y http.host > http.txt")
```

In particolare il primo comando genera un file dove sono presenti due colonne, nella prima ci sono gli indirizzi IP di risposta da parte del server DNS e nella seconda gli indirizzi IP della relativa richiesta. In caso di più di un indirizzo questi sono separati da virgola.
Con il secondo comando invece viene generato un file di testo che ha come prime colonne i campi relativi alla quintupla, alla sesta colonna il nome host con il quale è stata aperta una connessione SSL. Con l'opzione -Y di tshark vengono filtrati i pacchetti che non hanno aperto una connessione SSL e non vengono inseriti nel file.
I pacchetti vengono filtrati perché per ogni cattura di 5 minuti tutti i pacchetti scambiati tra host che condividono la stessa quintupla, fatta eccezione per rari casi[1] questi vengono classificati come un unico biflusso, in caso di apertura di più connessioni SSL queste avrebbero comunque lo stesso nome di dominio.
Con il terzo comando invece viene generato un file contenente come prime colonne i campi relativi alla quintupla e alla sesta colonna il nome di un host se questo ha inviato/ricevuto pacchetti HTTP.

```
#leggo dal file gt.tie
with open(nomefile,newline = "") as filecsv:
    lettore = csv.reader(filecsv,delimiter="\t")
    for count in dodici: header = next(lettore) #elimino le righe di intestazione
    n_col = 0
    for elem in header:
        col_names_temp.append(elem)
        n_col = n_col + 1 #conto il numero di colonne
    index = 0
    while index < n_col:
        #correggo la sbagliata indentazione del file .tie che genera colonne vuote
        if(col_names_temp[index]!=""): col_names.append(col_names_temp[index])
        index = index + 1
    print("i nomi delle colonne sono: ",col_names)
    #aggiungo i campi alla prima riga, le nuove colonne
    col_names.append("package")
    col_names.append("DNS")
    col_names.append("dig")
    col_names.append("whois")
    col_names.append("SNI_TLS")
    col_names.append("HTTP")
```

[1] TIE non effettua la classificazione per ogni pacchetto, ma accorpa gruppi di pacchetto in biflussi. Può accadere che ci sia una connessione aperta e che gli host non si scambino dati per un po. TIE utilizza un intervallo di tempo per determinare se pacchetti aventi la stessa quintupla appartengono o meno al biflusso, in caso contrario ne crea uno nuovo.
Nel nostro caso abbiamo lasciato il valore di default per il timeout, al massimo ci potrebbe essere qualche biflusso ripetuto.

*Il file **class.tie** generato da TIE viene aperto come un file di tipo csv con elementi separati da un tab. Vengono ignorate le prime 12 righe di intestazione in quanto non contengono dati necessari all'elaborazione. Viene poi generata la prima riga contenente tutti i nomi delle colonne e vengono poi aggiunti gli altri campi di interesse dopo aver corretto il fatto che alcune colonne fossero separate da più tab, il che migliora la visualizzazione ma rende difficile l'elaborazione.*
Lo script leggerà tutte le righe dal file di TIE, ogni riga è relativa ad una quintupla e per ogni quintupla eseguirà le operazioni necessarie per estrarre i campi da aggiungere. Viene creato quindi una rappresentazione virtuale del file memorizzata un una lista di liste. In particolare ogni riga del file è una lista di stringhe di cui ogni campo è un elemento di una colonna, una lista di liste contiene quindi tutte le righe del file.
Il file di output viene poi generato solo alla fine dello script e l'intera struttura dati viene scritta su file.

```
with open(nomefileout, 'w') as filecsvw:
    writer = csv.writer(filecsvw, delimiter="\t")
    i = 0
    while i < len(rows):
        writer.writerow(rows[i])
        i = i + 1

filecsvw.close()
```

*Per trovare il package che ha richiesto l'apertura della socket bisogna dividere i due casi **TCP** e **UDP**.*

```
if(protoC == "TCP"):

    #creo il primo output
    cmd = "grep -i "+ ipsrc + ":" + psrc + " strace.log > src.txt"
    print("eseguo il comando ",cmd)
    os.system(cmd)
    print("comando eseguito")

    #secondo
    cmd = "grep -i "+ ipdst + ":"+ pdst + " src.txt > dst.txt"
    print("eseguo il comando ",cmd)
    os.system(cmd)
    print("comando eseguito")

    #terzo
    cmd = "grep -i "+ protoC + " dst.txt > proto.txt"
    print("eseguo il comando ",cmd)
    os.system(cmd)
    print("comando eseguito")
```

*Nel primo caso nel file **strace.log** le socket TCP aperte contengono le informazioni relative alla quintupla essendo TCP un protocollo connection oriented.*
*Lo script quindi per ogni quintupla letta dal file utilizza **grep** per la ricerca, filtra i dati relativi alla quintupla e manda il risultato in un file di output.*

```
elif(protoC == "UDP"):
    ip = ""
    if(ipdst.startswith("192.168")):
        ip = ipsrc
    elif(ipsrc.startswith("192.168")):
        ip = ipdst

    #creo un comando per leggere il package delle socket UDP
    cmdUDP = ""
    cmdUDP = "grep -i UDP strace.log | grep -i " + ip + " > proto.txt"
    print("eseguo comando ",cmdUDP)
    os.system(cmdUDP)
    print("comando eseguito")
```

Essendo invece UDP un protocollo connectionless le socket vengono trattate diversamente. In particolare viene memorizzato solo un identificativo, il protocollo e l'indirizzo di destinazione.
Dato che TIE potrebbe invertire l'indirizzo sorgente e destinazione ed è di interesse solo quest'ultimo, se viene rilevato che esso è un indirizzo privato, allora viene scelto l'altro per effettuare la ricerca delle socket UDP.

Viene quindi generato un file che contiene come primo elemento il package che richiede al sistema l'apertura della socket, il package viene letto carattere per carattere fino allo spazio che per questo file è il separatore.
Se il file generato è vuoto non viene trovato nessun valore per il package, un esempio è il caso delle richieste al server DNS 8.8.8.8.

```
#ricerca in traffic.pcap del DNS
#apro il file trafficpcap.txt
with open("trafficpcap.txt",newline = "") as filedns:
    readerpcap = csv.reader(filedns,delimiter = "\t")
    rowDns = []
    for rowdns in readerpcap:
        rowDns = rowdns
        print("linea letta: ",rowDns)
        #leggo gli indirizzi ip della risposta DNS, considero solo il primo IP
        if(ipdst.startswith("192.168")):
            ip = ipsrc
        elif(ipsrc.startswith("192.168")):
            ip = ipdst
        if(rowDns[0].find(ip) != -1):
            i = 0
            ipdns = ""
            rowd = rowDns[0]
            while i < len(rowDns[0]):
                if(rowd[i] != ","):
                    ipdns = ipdns + rowd[i]
                    i = i + 1
                else: break
```

Per effettuare la ricerca delle risoluzioni DNS, a partire dal file generato da tshark all'inizio dello script, si ricerca in questo file l'indirizzo IP dell'host contattato, fornito in risposta dal server DNS e viene confrontato con uno dei due indirizzi IP della quintupla (quello che non è un indirizzo privato) e ricercato nella prima colonna del file ***traffic.txt****, nella colonna DNS viene inserito, se presente, il nome di dominio legato all'indirizzo IP della risposta DNS.*
Questo indirizzo IP viene poi utilizzato per effettuare le query per la risoluzione dns inversa (utilizzando il comando ***dig****) e per l'interrogazione* ***whois****.*
Queste interrogazioni saranno necessarie per comprendere meglio la presenza di traffico di terze parti generato dall'applicazione.

```
    #ssl_handshake
filessl = open("ssl.txt","r")
ssl = filessl.readlines()
added = False
for line in ssl:
    linessl = line.split("\t")
    if(linessl[0] == ipsrc and linessl[1] == psrc and linessl[2] == ipdst and linessl[3] == pdst):
        ssline = linessl[4][0:len(linessl[4])-1] #rimuovo l'ultimo carattere che danneggia la formattazione
        row.append(ssline)
        added = True
        nSSL = nSSL + 1
        print("trovato SSL handshake : ",ssline)

if(added == False):
    row.append("Non Trovato!")
```

Per ricercare il nome host associato all'indirizzo IP contattato dall'applicazione che apre una connessione SSL viene cercata la quintupla all'interno del file generato inizialmente con il filtro tshark descritto in precedenza.

Similmente viene effettuata anche la ricerca per eventuale traffico HTTP.

```
        #http

    filehttp = open("http.txt","r")
    httplines = filehttp.readlines()
    for line in httplines:
        http = line.split("\t")
        if(http[0] == ipsrc and http[1] == psrc and http[2] == ipdst and http[3] == pdst):
            nHTTP = nHTTP + 1
            #correzione output
            http[4] = http[4][0:len(http[4])-1]
            row.append(http[4])
            print("trovato HTTP: ",http[4])
    filehttp.close()
```

3.3 Output e Classificazione

Per ogni cattura, lo script genera un file formattato come CSV contenente tutte le colonne necessarie per effettuare la classificazione.
Dall'output dello script è stato quindi generato un file excel in cui ogni foglio rappresenta una cattura.
Quindi sono stati analizzati i dati manualmente confrontando i protocolli di trasporto utilizzati e il numero di socket UDP e TCP aperte.

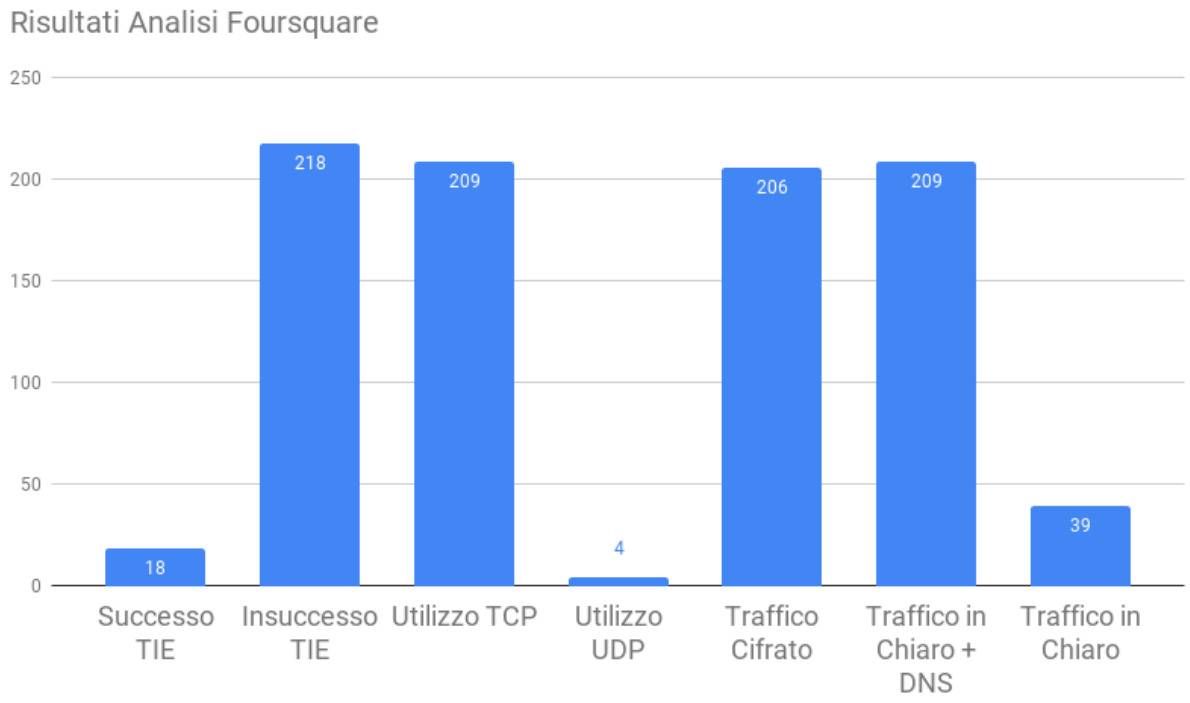
Inoltre sono stati analizzati i biflussi classificati, più o meno, correttamente da TIE confrontando il risultato con quello dello script ed effettuando ricerche in Internet sui servizi forniti dalle entità contattate.

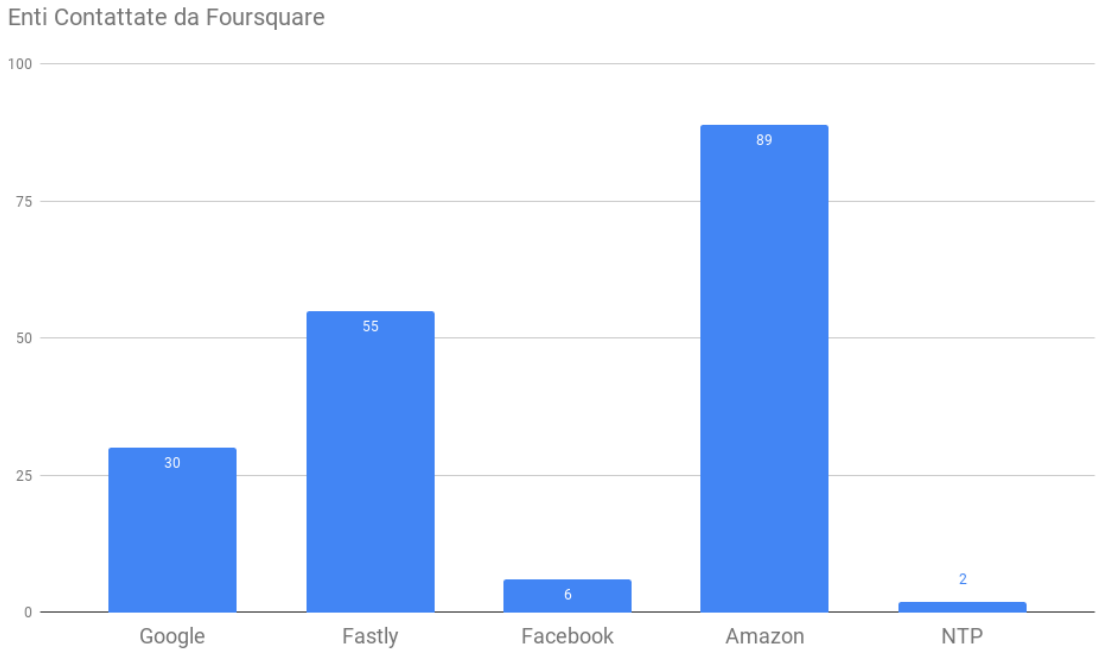
Capitolo 4: Risultati sperimentali

I dati sono stati analizzati singolarmente cattura per cattura, di seguito verrà fornita un'analisi generale del traffico catturato e classificato descrivendo e confrontando le quattro applicazioni mobili testate. Ove di interesse verrà effettuata una descrizione più approfondita.
Per ogni applicazione sono stati di particolare interesse le colonne whois e SNI_TLS, dai quali sono stati determinati rispettivamente le entità contattate e il numero di biflussi cifrati rispetto a quelli non cifrati.

FOURSQUARE

TIE *effettua una classificazione basata sul contenuto del payload dei pacchetti, ma nella maggioranza dei casi, non è riuscita a effettuare la classificazione correttamente a causa della maggioranza di traffico cifrato.*
La maggior parte del traffico inviato in chiaro è costituito infatti dai biflussi relativi alle richieste **DNS**.
Il protocollo più utilizzato a livello di trasporto è il TCP, UDP viene utilizzato quasi esclusivamente per le richieste DNS.





Foursquare *utilizza diversi servizi riconducibili ad* ***Amazon*** *come i servizi di* ***Content Delivery Network*** *(CloudFront) e* ***hosting*** *(Amazon Web Services).*
Anche la rom modificata del telefono (Cyanogenmod) utilizza il servizio di hosting AWS che genera traffico di controllo stats.cyanogenmod.org non generato dall'applicazione, in quanto non risulta alcuna socket aperta per quelle quintuple che nel file contenente il log delle chiamate a sistema ***strace.log.***

*Vengono utilizzati anche servizi di Content Delivery forniti da **Fastly** e servizi di terze parti come i servizi di analisi e report di eventuali crash (**crashlytics**) fornito da **Google**.*

***Foursquare** offre inoltre la possibilità di effettuare il login mediante **Facebook,** da noi non utilizzato, ma l'applicazione comunque contatta Facebook mediante l'API **graph.facebook.com** che consente di effettuare query su dati in possesso di Facebook e scambiare informazioni sull'utilizzo dell'applicazione con quest'ultimo al fine di fornire pubblicità mirata.*

In particolare abbiamo utilizzato Google per le funzionalità di login.

*L'applicazione contatta Google e utilizza le api offerte dal colosso di Mountain View per accedere ai dati di **Google Maps** per quanto riguarda la posizione delle attività **(play.googleapis.com).***

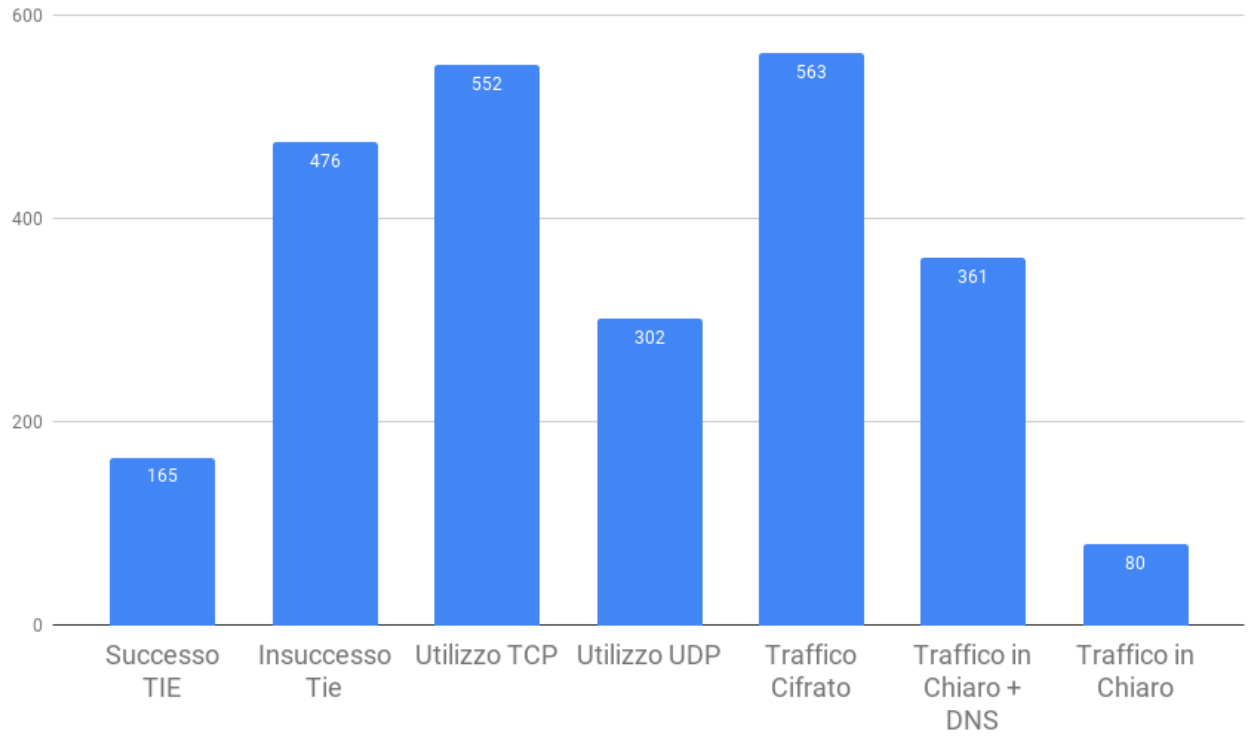
*Il traffico verso Google comprende anche il traffico dati relativo al download dell'applicazione generato dal package (**com.android.vending**).*

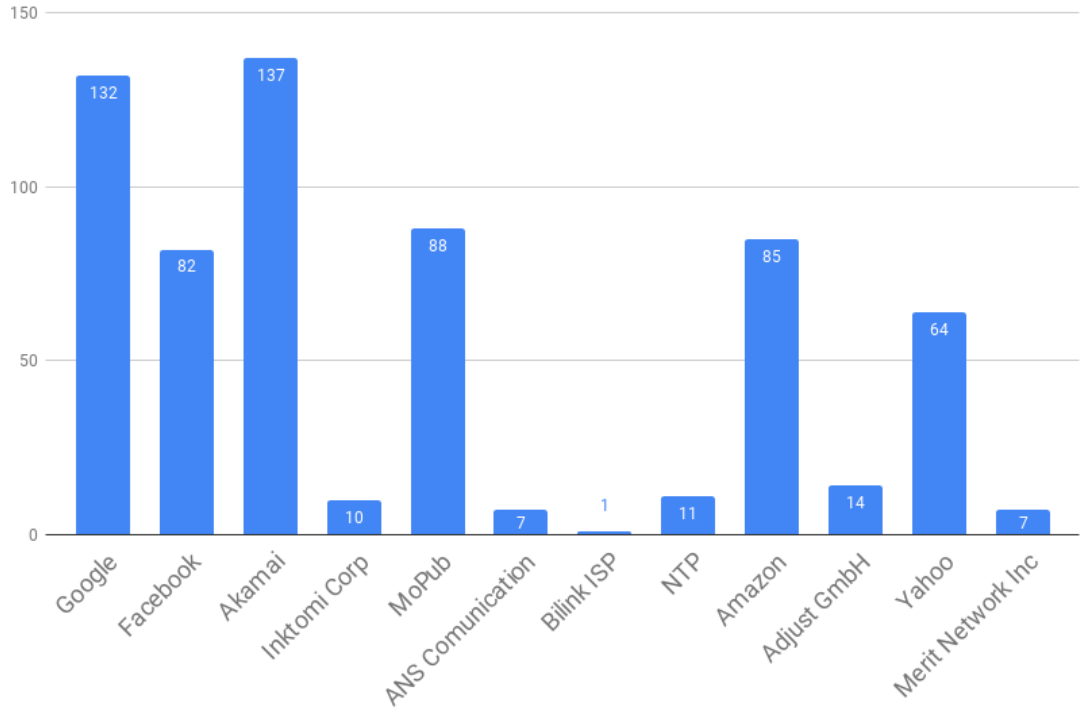
Il flusso di dati generati dal package del Play Store avrebbe dovuto essere presente solo nella prima cattura ma casualmente, in seguito a dei freeze dell'applicazione è stato involontariamente lanciato il Play Store.

*L'applicazione utilizza inoltre servizi di terze parti forniti da **Branch**, una compagnia che si occupa di analisi dei dati, in particolare utilizza i suoi servizi di generazione di link abbreviati e analisi di questi ultimi (**api.branch.io**) che non risultano nel grafico perché ospitati dai server AWS.*

Onefootball

In basso è possibile osservare il grafico che rappresenta l'efficienza dell'applicativo TIE e contestualmente anche il valore dei parametri analizzati.
Anche in questo caso, a causa della forte maggioranza del traffico cifrato la classificazione di TIE non è andata a buon fine.





Questo Grafico illustra il numero di biflussi indirizzati alle rispettive entità contattate dall'applicazione OneFootball.
Di seguito verranno descritti i servizi forniti dalle entità. Non considereremo come servizi di terze parti NTP (Network Time Protocol) e CDN (Content Delivery Network) in quanto necessari per il corretto funzionamento dell'applicazione.
Onefootball usufruisce di NTP e di alcuni servizi di terze parti, tra i quali spiccano i servizi di advertising, profilazione e analisi dei dati utente.
Questi ultimi vengono utilizzati per campagne pubblicitarie ad hoc basate sugli interessi degli utenti.

NTP è un protocollo di rete che consente di ricavare da alcuni server il tempo corrente, perché potrebbe essere errato sul dispositivo. Non è considerato un servizio di terze parti ma Onefootball contatta diversi NTP server appartenenti a diverse entità per ricevere informazioni corrette sul tempo per riuscire a fornire informazioni esatte sugli eventi sportivi in diretta.

Il servizio di Content Delivery è fornito, nel caso di Onefootball, da Akamai.

Akamai viene utilizzata in qualità di CDN (Content Delivery Network) per lo streaming video nella sezione notizie.

Le restanti entità forniscono a Onefootball servizi dall'advertising all'analisi dell'utilizzo dell'applicazione al fine di migliorare l'esperienza di utilizzo e fornire pubblicità mirata.

*In particolare vengono utilizzati i servizi forniti da Google, per le funzionalità di login, da noi utilizzata e advertising (**pubads.g.doubleclick.net**).*

Facebook invece viene utilizzata per servizi di log-in ed inoltre le funzionalità di facebook sono implementate all' interno dell' app per permettere all'utente di condividere i risultati delle partite o commentare eventi.
*L'API fornita da Facebook **graph.facebook.com** consente di effettuare numerose operazioni con il social network come il login o inviare e ricevere dati da Facebook come ad esempio pubblicità o informazioni per esse.*

Inktomi Corporation era una società che forniva software per provider di servizi Internet (ISP). È stato riorganizzato in Delaware e ha sede a Foster City, in California. I clienti includono Microsoft, HotBot, Amazon.com, eBay e WalMart.

MoPub è una società controllata da Twitter, fornisce soluzioni di monetizzazione per editori e sviluppatori di app per dispositivi mobili in tutto il mondo.

ANS Communication si occupa della progettazione, gestione e costruzione di Infrastrutture di rete.

Bilink è un servizio di Advertising.

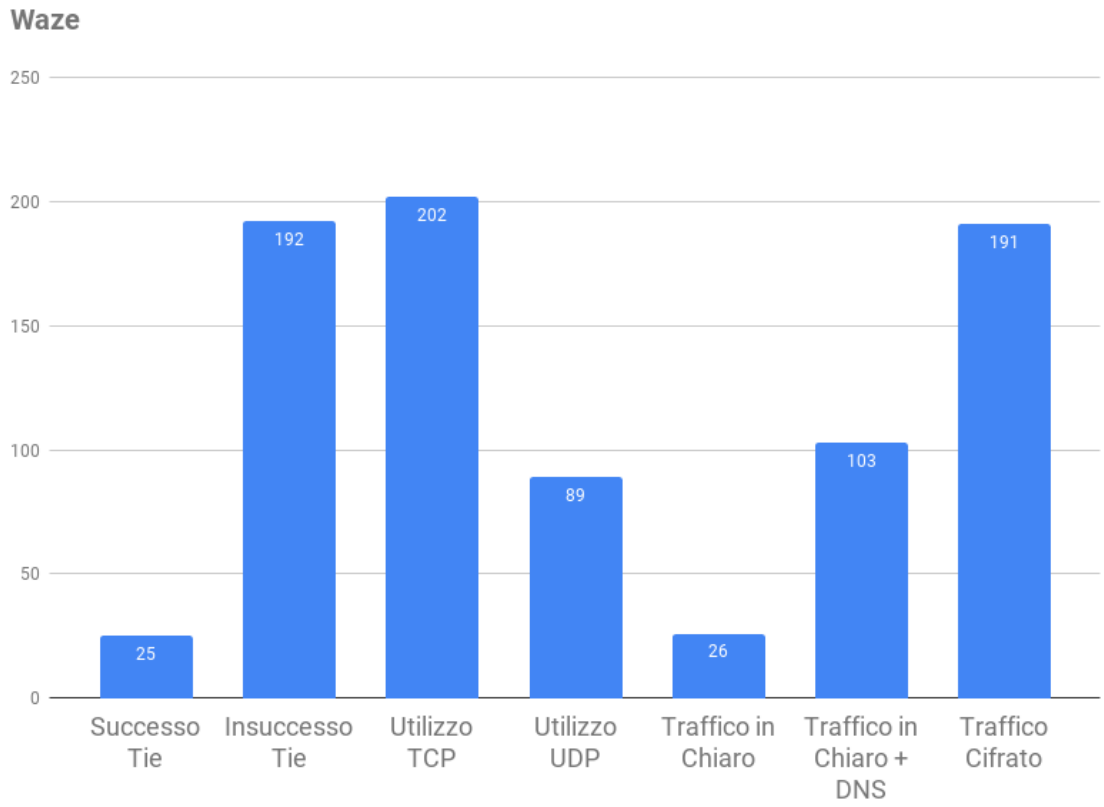
Amazon AWS è uno dei servizi più utilizzati perché offre servizi di cloud computing affidabili, scalabili ed economici. L'account è gratuito e si pagano solo i servizi utilizzati.

Adjust GmbH (Mobile Measurement Company) unisce tutte le tue attività di marketing in un'unica potente piattaforma, offrendo le informazioni necessarie per ridimensionare il business aziendale.

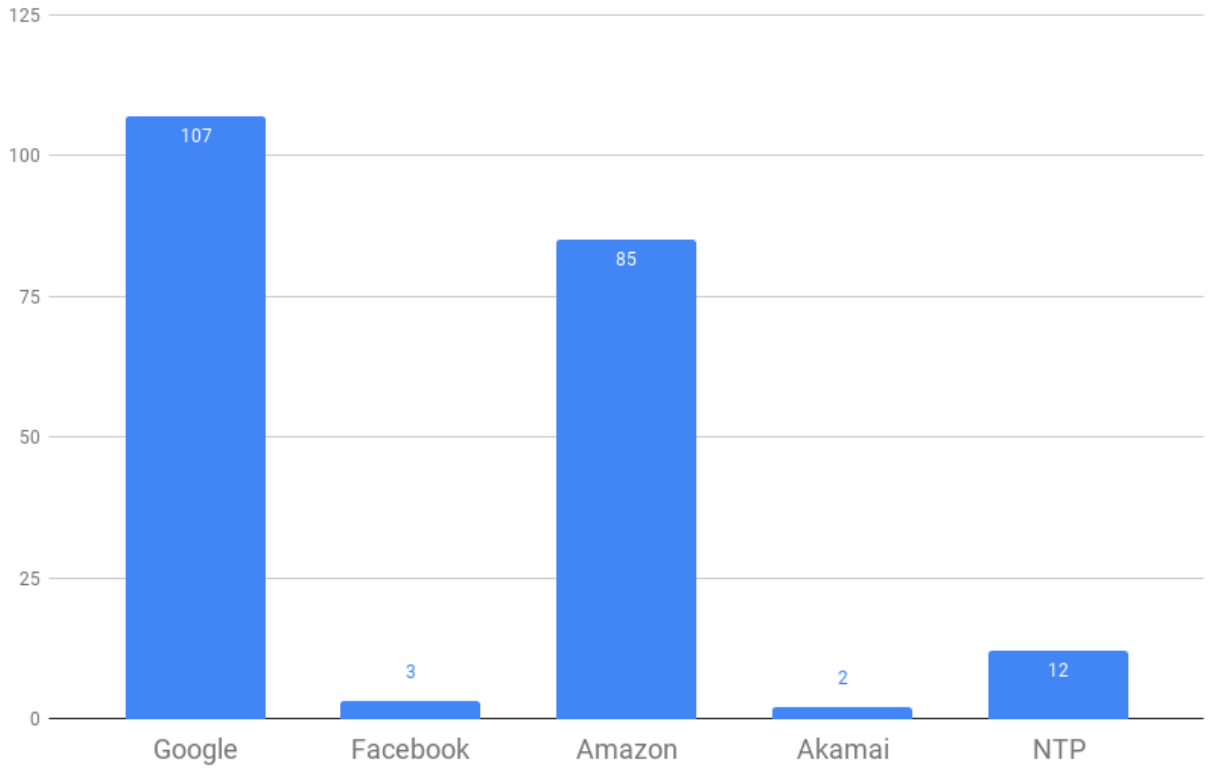
*Yahoo! viene contattata dall'applicazione Onefootball perché essa utilizza il suo servizio **flurry**, un servizio che consente di analizzare il comportamento dell'utente all'interno dell'applicazione per poter migliorare facilmente l'esperienza di utilizzo. Questo servizio effettua un'analisi dei dati di utilizzo e, combinato con informazioni relative alla posizione, viene effettuata una profilazione completa degli utenti. Dopo aver effettuato la raccolta dati e la profilazione, Yahoo! fornisce anche pubblicità ad hoc.*

Merit Network,Inc. offre assistenza in rete e soluzioni IT per università pubbliche, college, organizzazioni K-12, biblioteche, governo statale, assistenza sanitaria, biblioteche, istituti di ricerca e altre organizzazioni no-profit nel Michigan. Offre servizi Internet e di rete per supportare applicazioni ad ampia larghezza di banda, come l'apprendimento a distanza, trasferimenti di dati su larga scala, videoconferenze, laboratori virtuali, strumentazione remota e librerie digitali.

Di seguito verrà analizzato il traffico generato dall'applicazione waze.



Anche quest' applicazione genera flussi di traffico prevalentemente cifrati e TIE, per questo motivo, non riesce ad effettuare una corretta classificazione.



Google ha acquisito Waze nel 2013. Il traffico diretto a Google quindi non può essere considerato come un traffico verso un servizio di terze parti ma l'applicazione utilizza molti servizi offerti dal noto motore di ricerca, in particolare Waze invia le segnalazioni riguardanti il traffico in modo da poter essere visibili su Google Maps e utilizza la ricerca dei luoghi di quest'ultima. Viene utilizzato inoltre il servizio di ***Text To Speech*** *(****tts.waze.com****) fornito da Google, che viene contattata anche per le pubblicità presenti nell'applicazione (****advil.waze.com****).*

Facebook viene invece contattata perché offre la possibilità di ricercare amici nelle vicinanze che utilizzano l'applicazione e per i servizi di pubblicità sulla base delle informazioni di localizzazione delle pagine.

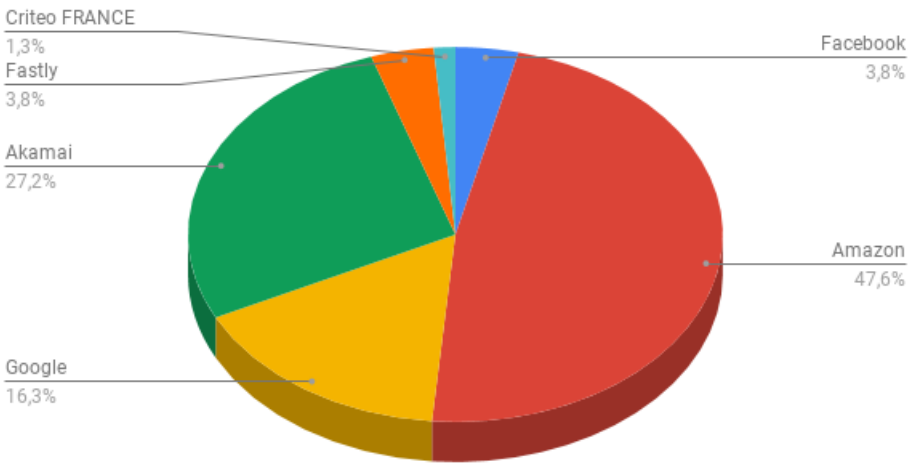
Amazon AWS è utilizzato da Waze simultaneamente con il cloud fornito dal Google per migliorare l'esperienza utente e l'affidabilità dell'applicazione.

NTP viene utilizzato anche da questa applicazione per ottenere il tempo corrente e mostrare le corrette informazioni sul traffico.

org.cyanogenmod.gello.browser	www.mcdonalds.it	a95-101-34-105.deploy.static.akamaitechnologies.com	Akamai	www.mcdonalds.it
org.cyanogenmod.gello.browser	www.mcdonalds.it	a95-101-34-105.deploy.static.akamaitechnologies.com	Akamai	www.mcdonalds.it

È stato rilevato inoltre che il browser del telefono durante la cattura ha contattato Mc Donald's e che questi utilizza il servizio di Content Delivery di Akamai.

L'applicazione Wish contatta diverse entità, ma non tutte sono riconducibili a servizi di terze parti.



Ad esempio Wish utilizza AWS per l'hosting delle immagini dei prodotti acquistabili, mediante l'applicazione, mentre è Akamai a fornire una rete di Content Delivery (CDN).
L'applicazione sfrutta però diversi servizi di terze parti per advertising, verifica dei pagamenti ed analisi e profilazione delle utenze.
Oltre il 16% del traffico è diretto verso Google, molti sono i servizi forniti da questa multinazionale.
Diversi sono i servizi utilizzati dall'applicazione, ci sono la funzionalità di login (**accounts.google.com**), da noi utilizzata per testare l'applicazione durante la cattura del traffico generato, e le **API** di google che consentono all'applicazione di avere accesso ad alcuni dati tra cui GoogleMaps e Google+ e alcuni servizi di pubblicità per In-App-Advertising (**www.googleadservices.com**).
Gstatic, stando ad alcune ricerche effettuate in rete,risulta essere una CDN di Google ad uso privato sulla quale sono presenti i contenuti statici che l'applicazione richiede, usata per un caricamento più veloce e per fornire tempi di attesa più brevi.
Wish utilizza due servizi di terze parti per la gestione dei pagamenti, *BrainTree*, un servizio relativo a Paypal e Stripe.com, un servizio di terze parti che si occupa della gestione e verifica delle transazioni.

com.contextlogic.wish	m.stripe.com	ec2-54-218-104-	Amazon.com, Inc	m.stripe.com
N/C	m.stripe.com	ec2-54-218-104-	Amazon.com, Inc	Non Trovato!

Inoltre, è stato possibile osservare che Wish, usufruisce anche dei servizi offerti da Riskified.com, che si occupa di prevenire il fenomeno di fraudulent chargebacks.
Questo fenomeno si verifica quando un consumatore che acquista su siti di e-commerce, chiede un chargeback alla propria banca dopo l'effettivo ricevimento della merce, il chargeback annulla la transazione finanziaria e il consumatore riceve un rimborso dei soldi spesi.

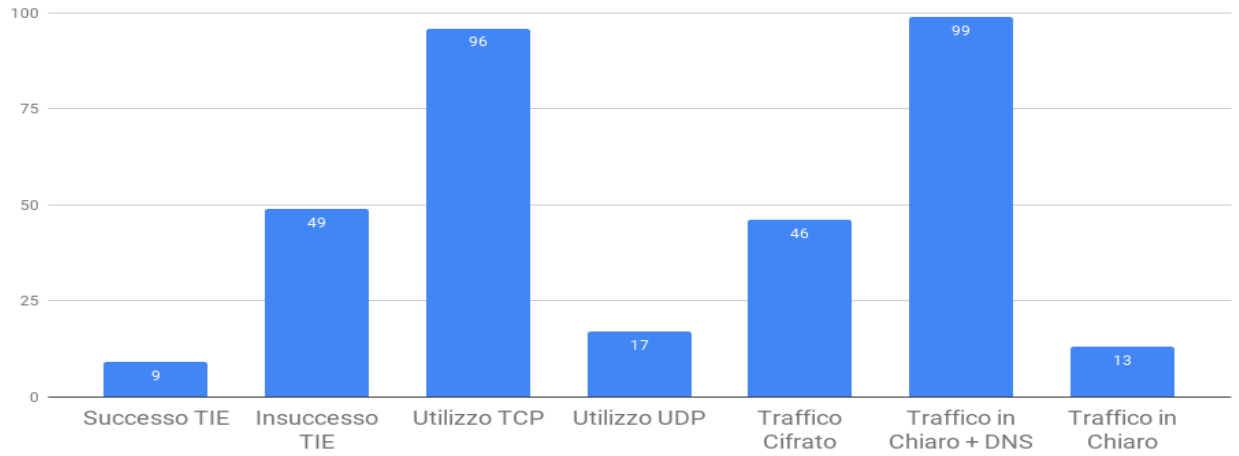
package	DNS	dig	whois	SNI_TLS	HTTP	FLAGS
com.contextlogic.wish	c.riskified.com	ec2-52-6-250-95	Amazon Technol	c.riskified.com		N

Facebook viene invece contattata sia per fornire il servizio di login, da noi non utilizzato, ma in particolare i server **edge-star-***** vengono utilizzati per inviare a Facebook dati relativi alle abitudini dell'utente e ricevere in cambio pubblicità mirata.

com.contextlogic.wish	connect.facebook.net	xx-fbcdn-shv-01-mxp1.fbcdn.net	connect.facebook.net
com.contextlogic.wish	www.facebook.com	edge-star-mini-shv-01-mxp1.facebook.com	www.facebook.com

Wish utilizza anche un altro servizio di targeting dei clienti offerto da **Criteo FRANCE**, in particolare questa offre servizio di advertising dinamico basato su tecniche di Machine Learning, è quindi in grado di proporre ai clienti prima i prodotti che probabilmente acquisterà o prodotti già visti in precedenza.

I prodotti sono determinati analizzando il comportamento dei clienti e contemporaneamente il catalogo dei prodotti di Wish e proponendo le giuste offerte.



Dai dati risulta che la classificazione effettuata da TIE non è sempre corretta, questo perché il traffico inviato, anche in questo caso, è per buona parte cifrato.
Del traffico inviato in chiaro dall'applicazione, oltre alle richieste DNS, troviamo le immagini inviate da **contestimg.wish.com** che vengono classificate da TIE solo come JPG, probabilmente perché TIE non riconosce la firma dell'applicazione.
Inoltre tra i dati più rilevanti estrapolati dal grafico è evidente che le connessioni sono sempre di più TCP Oriented.

Conclusioni

Dall'analisi dei dati si evince che, in prima istanza, **TIE** non riesce a classificare correttamente la maggior parte del traffico, a causa della cifratura del **Payload**.
Inoltre si nota da subito che tutte le applicazioni usufruiscono dei servizi di **Content Delivery**, siano esse CDN proprietarie, come nel caso di Waze, o meno, ad esempio OneFootball.
Tutte le applicazioni sfruttano inoltre servizi di terze parti (in maggioranza Google LLC) che in cambio di dati da parte degli utenti, forniscono servizi di profilazione al fine di fornire pubblicità mirata, ciò da un lato migliora significativamente l'esperienza di utilizzo delle applicazioni in quanto i contenuti propinati sono maggiormente attinenti agli interessi dell'utenza, dall'altro viene fornito un servizio molto più efficiente agli inserzioinisti, ma questo spesso vìola la privacy dell'utente in mancanza di una richiesta esplicita da parte dell'applicazione.
Analizzando poi i dati relativi all' utilizzo dei protocolli di trasporto, si nota immediatamente che l'utilizzo del **TCP** è significativamente superiore a quello dell' **UDP**, quest'ultimo utilizzato solo dal Google Play Store che, probabilmente, utilizzerà in realtà il suo protocollo **GQUIC**[2], che è un protocollo che incapsula diversi servizi forniti da **TCP** a livello applicazione.

[2] GQUIC: Google Quick UDP Internet Connection

Bibliografia

- W. de Donato, A. Pescapè, A. Dainotti, “Traffic identification engine: an open platform for traffic classification,” Network, IEEE , vol.28, no.2, pp.56-64, March-April 2014.
- Reti di calcolatori e internet. Un approccio top-down. Ediz. mylab
  di James F. Kurose (Autore), Keith W. Ross (Autore), A. Capone (a cura di), S. Gaito