UNIVERSITÀ 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 2021/2022

Professore

Prof. Antonio Pescapè

Gruppo:

Francesco Pio Manna
N46005124

Francesca Miccio
N46004840

Martina Sannino
N46006760

Indice

Sommario

Lo scopo dell'elaborato in questione è lo studio del traffico dati proveniente da alcune applicazioni di uso comune tra cui Skype, Meet, GoMeeting, Discord, Zoom e WebEx. L'analisi prevede il confronto di alcuni parametri riguardanti videocall/chat/webinar effettuate con tali app tra cui visualizzazione delle risoluzioni DNS e dei protocolli HTTP/TLS ed estrazione dei biflussi TCP e UDP.

Capitolo 1: Classificazione del traffico

Dalla nascita di ArpaNET prima e Internet poi, abbiamo potuto constatare la rapidità di evoluzione di tale piattaforma e del suo utilizzo. In particolare, ripensando agli ultimi 2 anni, durante i quali il mondo è stato devastato dalla pandemia, non si può non evidenziare la presenza sempre più massiccia del WEB nella vita delle persone, sia da un punto di vista didattico/lavorativo che da un punto di vista sociale. Infatti, questa situazione di emergenza ha normalizzato l'utilizzo del WEB rendendolo un must-have per chiunque. In particolare, tra gli utilizzi più diffusi del WEB, spiccano webinar, chat e videocall, che sono proprio l'oggetto delle nostre analisi.

Il traffico può essere identificato a partire dai pacchetti IP che costituiscono i flussi dati, detti anche biflussi. Un biflusso è definito come una sequenza di pacchetti IP di sorgente e di destinazione (Source Port e Destination Port) e lo stesso protocollo (il campo Protocol dell'header IP) di trasporto. Questi cinque campi sono usualmente denominati *quintupla* e identificano efficacemente il flusso al quale un pacchetto IP appartiene.

È possibile classificare il flusso di traffico dati secondo diverse tecniche:

- ***Port-Based:*** con questo metodo si tenta di classificare il traffico grazie all'associazione porta/protocollo;
- ***Packet-Based:*** ogni pacchetto è analizzato indipendentemente dagli altri;
- ***Flow-Based:*** in cui si mostrano informazioni riguardante i biflussi introdotti precedentemente;
- ***Payload-Base:*** che analizza la parte dati del pacchetto.

La classificazione del traffico ci ha permesso di lavorare su alcuni livelli dello stack protocollare TCP/IP. Infatti abbiamo lavorato sul livello applicazione per quanto riguarda le informazioni su HTTP e DNS, sul livello trasporto per quanto riguarda le informazioni su TCP e UDP e sul livello rete in merito a IP.

Una delle limitazioni di questo progetto consisteva nell'effettuare manualmente ognuna delle procedure necessarie al recupero dei dati.

Per ovviare a tale limitazione, abbiamo realizzato un piccolo script in linguaggio BASH che ci dava la possibilità di automatizzare il processo e di memorizzare i dati in formato tabellare (.csv).

Capitolo 2: Strumenti utilizzati

Gli strumenti utilizzati per la realizzazione del progetto sono stati i seguenti:

- *Smartphone Android rooted:* gli smartphone messi a disposizione, oltre ad avere abilitati i permessi di root, davano la possibilità di inibire il traffico a tutte le applicazioni che non fossero quelle interessate dalla cattura;
- *Sistema di cattura MIRAGE;*
- *Linux:* abbiamo utilizzato tale sistema operativo su macchina virtuale (VirtualBox);
- *Wireshark:* un packet sniffer per effettuare un'analisi preliminare generale con un' interfaccia grafica che permette facilmente di identificare i pacchetti e i vari protocolli ad essi associati;
- *TShark:* un altro packet sniffer, la versione terminal-oriented di Wireshark, utilizzato poi effettivamente per realizzare il progetto, basato su riga di comando;
- *Whois:* è un protocollo di rete che consente, mediante l'interrogazione di appositi database server da parte di un client, di stabilire a quale internet provider appartenga un determinato indirizzo IP o uno specifico DNS.

Capitolo 3: Cattura e analisi del traffico generato da app ISSW

3.1 Cattura del traffico

Le catture sono state effettuate nel laboratorio ARCLAB in tre fasi, distanziate l'una dall'altra di almeno due settimane. Ogni cattura ha avuto una durata di 15 minuti, per ogni applicazione e per ogni servizio utilizzato su queste ultime. La cattura ha avuto inizio con l'avvio della call/videocall e si è conclusa alla chiusura della stessa. È stato necessario porre particolare attenzione al fatto che, durante la cattura, non dovevano essere aperte contemporaneamente altre applicazioni. In ogni caso, le funzionalità di rete di altre applicazioni, che non fossero quelle specifiche per la cattura, erano disattivate.

Le applicazioni oggetto di studio sono state:

- Skype;
- WebEx;
- Discord;
- Google Meet;
- GoMeeting;
- Zoom.

Il sistema di cattura ha prodotto numerosi file, tra cui il file .pcap, costituito da un insieme di dati creati utilizzando un analizzatore di protocollo, e *netstat*, un'utilità a riga di comando che fornisce statistiche essenziali su tutte le attività di rete e dà informazioni su quali porte e indirizzi funzionino le rispettive connessioni (TCP, UDP), oltre a indicare quale siano i porti aperti per accogliere richieste.

3.2 Analisi del traffico

Un'operazione preliminare è stata quella di installare le componenti tshark e Whois, tramite i seguenti comandi:

```
so@so-vbox:~$ sudo apt install -y tshark
so@so-vbox:~$ sudo apt install -y whois
```

Una volta fatto ciò, abbiamo iniziato ad utilizzare i primi tools.

3.2.1 Capinfos

Capinfos è un comando che legge uno o più file .pcap e mostra varie statistiche sul file.

```
so@so-vbox:~$ capinfos -A -B "cattura.pcap"
```

3.2.2 Estrazione biflussi TCP/UDP

Questo comando crea una tabella che mostra tutte le conversazioni (biflussi) presenti all'interno della cattura.

```
so@so-vbox:~$ tshark -r "cattura.pcap" -q -z conv,tcp
so@so-vbox:~$ tshark -r "cattura.pcap" -q -z conv,udp
```

3.2.3 DNS

Il seguente comando stampa a video le chiamate al DNS effettuate. In particolare, notiamo che il protocollo di livello trasporto utilizzato è UDP.

```
so@so-vbox:~$ tshark -r "cattura.pcap" -T fields -e frame.time_epoch -e frame.protocols -e ip.src -e
ip.dst -e ip.proto -e udp.srcport -e udp.dstport -e dns.a -e dns.qry.name -Y "(dns.flags.response ==
1 )"
```

3.2.4 SNI TLS

Questo comando permette la visualizzazione dei nomi dei server "virtuali" ospiti del server principale contattato.

```
so@so-vbox:~$ tshark -r "cattura.pcap" -T fields -e frame.time_epoch -e frame.proto
cols -e ip.src -e ip.dst -e ip.proto -e tcp.srcport -e tcp.dstport -e tls.handshake
.extensions_server_name tls.handshake.type == 1
```

3.2.5 Host

http.host contiene il nome dei siti web richiesti durante la cattura.

```
so@so-vbox:~$ tshark -r "cattura.pcap" -T fields -e frame.time_epoch -e frame.protocols -e ip.src -e
ip.dst -e ip.proto -e tcp.srcport -e tcp.dstport -e http.host -Y "http.host"
```

3.3 Script per l'automazione delle analisi

```
for d in */ ; do
    for f in $d*.pcap ; do

        echo -e  "Sto analizzando il file -> $f"
        echo -e " "

        #capinfos
        capinfos -A -B $f >> ./"$f"_capinfos.csv
        echo "Panoramica generale informazioni catture con Capinfos effettuata"

        #estrazione dei biflussi TCP e UDP
        tshark -r $f -q -z conv,udp >> "$f"_udp_biflussi.csv
        echo "Estrazione Biflussi UDP effettuata"
        tshark -r $f -q -z conv,tcp >> "$f"_tcp_biflussi.csv
        echo "Estrazione Biflussi TCP effettuata"

        #DNS
        tshark -r $f -T fields -e frame.time_epoch -e frame.protocols -e ip.src -e ip.dst -e ip.proto -e udp.srcport -e udp.dstport -e dns.a -e dns.qry.name
        -Y "(dns.flags.response == 1 )" >> ./"$f"_dns.csv
        echo "Stampa delle risoluzioni DNS effettuata"


        #SNI TLS
        tshark -r $f -T fields -e frame.time_epoch -e frame.protocols -e ip.src -e ip.dst -e ip.proto -e tcp.srcport -e tcp.dstport -e tls.handshake.extensions_server_name
        tls.handshake.type == 1  >> "$f"_tls.csv
        echo "Analisi TLS effettuata"


        #HTTP
        tshark -r $f -T fields -e frame.time_epoch -e frame.protocols -e ip.src -e ip.dst -e ip.proto -e tcp.srcport -e tcp.dstport -e http.host -Y "http.host"
        >> ./"$f"_host_HTTP.csv
        echo -e "HTTP fatto"

    done
done
```

Questo script è stato realizzato in linguaggio BASH. Prevede due for in range innestati, uno che scorre tutte le directories della cartella in cui è presente lo script, e uno che scorre tutti i .pcap individuati all'interno di esse. Le variabili *d* ed *f* conterranno, ad ogni iterazione del ciclo, i nomi delle rispettive directories e dei rispettivi .pcap (che contengono le informazioni riguardanti la cattura). Il comando *echo* permette di stampare a video informazioni di debug. Ognuno dei risultati dei comandi *tshark* sopra descritti è stato immagazzinato in un file.csv.

Capitolo 4: Risultati sperimentali

A seguito delle operazioni descritte in precedenza, abbiamo ottenuto diversi risultati interessanti e li abbiamo classificati per tipologia di attività svolta.

4.1 Chat

4.1.1 Skype

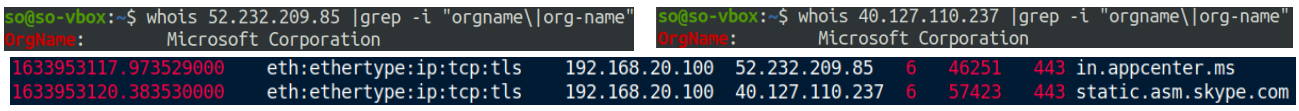
Dai risultati emersi dal comando capinfos, possiamo evidenziare il basso packet rate dei pacchetti scambiati (concorde con la natura discontinua/asincrona delle chat).

In quanto chat, necessita di una comunicazione sicura e affidabile, pertanto l'intero traffico viaggia su TCP (file biflussi UDP vuoto).

Tra i biflussi TCP, possiamo notare che la connessione tra l'host (nella nostra analisi sarà sempre il telefono utilizzato per la cattura) e l'indirizzo 8.8.4.4. su porto 853 (DNS over TLS) persiste per tutta la durata della cattura.

SKYPE	
Number of packets	3605
File size	1967 kB
Data byte rate	1971 bytes/s
Average packet size	529,87 bytes
Average packet rate	3 packets/s

Inoltre, la totalità dei biflussi fa riferimento a server virtuali di proprietà Microsoft.



4.2 Audiocall

4.2.1 Skype

Da questa cattura si può evidenziare il fatto che quasi tutto il traffico viaggia su UDP (tipico di comunicazioni real-time). Tra i porti destinazione compare 3480, porto utilizzato dal servizio Playstation Network per effettuare connessioni di test, il che ci ha incuriosito poiché tutti i server appartengono a Microsoft (che è un diretto concorrente di Sony).

SKYPE	
Number of packets	97 k
File size	16 MB
Data byte rate	15 kBps
Average packet size	152,96 bytes
Average packet rate	102 packets/s

Inoltre, i file riguardanti DNS e HTTP sono vuoti. Tuttavia, risulta una connessione sul porto 80 di durata 0, segno che c'è stato un problema.

4.2.2 WebEx

Il traffico è per la maggior parte di natura UDP, in particolare viene fatto uso del protocollo UDP Cast sul porto 9000, utilizzato per trasferire dati simultaneamente a gruppi multicast.

WEBEX	
Number of packets	63 k
File size	13 MB
Data byte rate	12 kBps
Average packet size	197,05 bytes
Average packet rate	62 packets/s

Effettuando test con whois, abbiamo constatato che i web server contattati appartengono per lo più a Cisco.

I comandi DNS e HTTP non hanno prodotto risultati.

4.2.3 Discord

DISCORD	
Number of packets	56 k
File size	12 MB
Data byte rate	12 kBps
Average packet size	211,92 bytes
Average packet rate	59 packets/s

All'interno del file DNS troviamo due coppie di query, una verso server Google e una verso server *connectivitycheckgstatic* utilizzati per funzioni di controllo. Inoltre, questi ultimi compaiono anche nei risultati prodotti da *http.host*.

Analizzando i biflussi UDP, possiamo evidenziare che ci sono una serie di richieste di durata 0 sul porto 123. Il porto 123 è utilizzato dal protocollo *NTP* (Network Time Protocol), che fornisce funzioni di sincronizzazione tra host sulla rete.

28	192.168.20.101:40641	<->	72.30.35.88:123	0	0	1	90	1	90	8,211006000	0,0000
29	192.168.20.101:34928	<->	45.15.168.198:123	0	0	1	90	1	90	74,412033000	0,0000
30	192.168.20.101:59594	<->	162.159.200.1:123	0	0	1	90	1	90	80,448038000	0,0000
31	192.168.20.101:60668	<->	72.30.35.88:123	0	0	1	90	1	90	86,458040000	0,0000
32	192.168.20.101:52474	<->	45.33.31.34:123	0	0	1	90	1	90	152,732070000	0,0000
33	192.168.20.101:51985	<->	162.159.200.1:123	0	0	1	90	1	90	158,746072000	0,0000
34	192.168.20.101:39806	<->	69.89.207.99:123	0	0	1	90	1	90	164,798084000	0,0000
35	192.168.20.101:52509	<->	45.33.31.34:123	0	0	1	90	1	90	230,817113000	0,0000
36	192.168.20.101:55054	<->	162.248.241.94:123	0	0	1	90	1	90	236,975112000	0,0000

Dal TLS, notiamo che ci sono server appartenenti ad aziende diverse.

1	1635500690.745293000	eth:ethertype:ip:tcp:tls	192.168.20.101	162.159.133.234	6	52301	443	gateway.discord.gg
2	1635500694.843294000	eth:ethertype:ip:tcp:tls	192.168.20.101	162.159.128.235	6	37248	443	latency.discord.media
3	1635500702.323299000	eth:ethertype:ip:tcp:tls	192.168.20.101	162.159.128.235	6	37253	443	eu-central9900.discord.media
4	1635500742.286311000	eth:ethertype:ip:tcp:tls	192.168.20.101	216.58.205.68	6	54124	443	www.google.com

4.2.4 Google Meet

Dalla lettura dei biflussi TCP emerge una connessione sul porto 5228, che corrisponde a servizi proprietari (Android Cloud to Device Messagging Service, Google Cloud Messaging)

```
32 192.168.20.101:58392        <-> 108.177.119.188:5228
```

TLS ci mostra un dato interessante: nonostante Google abbia chiuso il proprio servizio VoIP Google Hangouts, risultano ancora server con hostnames originari.

GOOGLE MEET	
Number of packets	134 k
File size	21 MB
Data byte rate	18 kBps
Average packet size	140,58 bytes
Average packet rate	131 packets/s

```
1 1635495354.759798000    eth:ethertype:ip:tcp:tls    192.168.20.101  142.250.184.106 6   54194   443 hangouts.googleapis.com
2 1635495355.047801000    eth:ethertype:ip:tcp:tls    192.168.20.101  216.58.208.138  6   35406   443 meetings.googleapis.com
```

Per quanto riguarda l'analisi UDP, ci sono due protocolli di Google utilizzati per le call, sui porti 19302 e 19305, ovvero *Google Talk Voice and Video connections* per il primo, e il secondo appartiene a un gruppo di porti (19303-19308) utilizzati da Google per app come Google Talk, DUO e Hangouts.

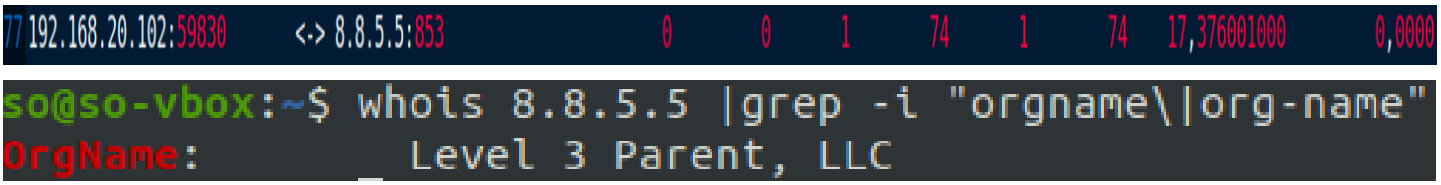
```
6 192.168.20.101:55846      <-> 142.250.82.99:19305      56056  7691865  67328  9603734  123384  17295599  0,620999000  1018,5915
7 192.168.20.101:55846      <-> 108.177.119.127:19302      102     7548    103     6386     205     13934  0,000000000  1014,2092
8 192.168.20.101:49205      <-> 142.250.82.99:19305         41     5494     41     6158      82     11652  1,237999000  1007,8888
9 192.168.20.101:49205      <-> 108.177.119.127:19302       13      962     13      806      26      1768  0,000997000   120,5054
```

4.2.5 GoMeeting

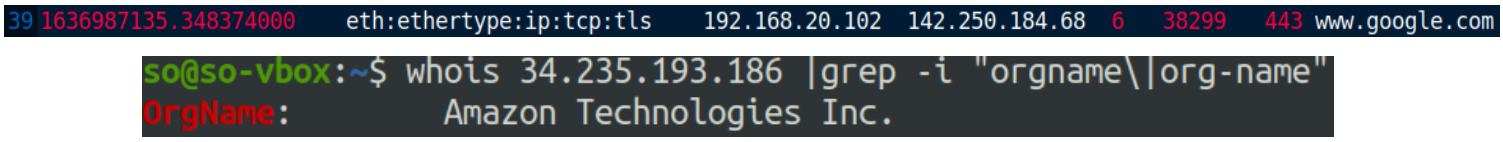
GOMEETING	
Number of packets	101 K
File size	16 MB
Data byte rate	15 kBps
Average packet size	145,69 bytes
Average packet rate	102 packets/s

Dai biflussi TCP compare una connessione sul porto 8200, corrispondente al servizio *MiniDLNA media server Web Interface* che utilizza DLNA/UPnP.

In aggiunta c'è una connessione DNS over TLS di durata nulla. Il server contattato appartiene alla *Level 3 Parent*.



La maggioranza dei server appartenenti ai biflussi sono di Amazon e Google.



4.2.6 Zoom

ZOOM	
Number of packets	92 K
File size	24 MB
Data byte rate	23 kBps
Average packet size	252,22 bytes
Average packet rate	93 packets/s

Le considerazioni riguardanti questa audiocall sono simili alle altre, con l'unica differenza che i server sono proprietari di Zoom. Inoltre, viene utilizzato un protocollo con porto 8801 appartenente a Zoom.

4.3 Webinar

I dati raccolti durante i webinar presentano analogie con quelli registrati con le corrispettive audiocall, se non per quanto riguarda le dimensioni e i rates dei pacchetti. Di seguito riporteremo queste discrepanze.

4.3.1 WebEx

Durante questo webinar, è stata simulata una vera e propria lezione online, in cui uno di noi ha condiviso un pdf (statico) sul quale ha annotato. È chiaro dunque che, essendo l'immagine per lo più statica, sono piuttosto ridotte per essere un webinar.

Nota: per maggiori informazioni riguardo i .csv di analisi, fare riferimento ai dati raccolti dalla corrispettiva audiocall.

WEBEX	
Number of packets	106 K
File size	45 MB
Data byte rate	44 kBps
Average packet size	408,72 bytes
Average packet rate	109 packets/s

4.3.2 Discord

A differenza del caso precedente, in questo webinar è stata condivisa una lezione registrata di un corso universitario, con uso intensivo della lavagna e numerosi movimenti. Questo spiega la grande dimensione del file.

Nota: per maggiori informazioni riguardo i .csv di analisi, fare riferimento ai dati raccolti dalla corrispettiva audiocall.

DISCORD	
Number of packets	283 K
File size	224 MB
Data byte rate	220 kBps
Average packet size	774 bytes
Average packet rate	285 packets/s

4.3.3 Google Meet

Le modalità di simulazione del webinar sono state analoghe al caso di Webex, con i medesimi contenuti.

Nota: per maggiori informazioni riguardo i .csv di analisi, fare riferimento ai dati raccolti dalla corrispettiva audiocall.

GOOGLE MEET	
Number of packets	127 K
File size	53 MB
Data byte rate	53 kBps
Average packet size	406,22 bytes
Average packet rate	132 packets/s

4.3.4 GoMeeting

In questo webinar abbiamo proiettato un video dalla piattaforma YouTube, con numerosi movimenti e ad alta qualità. Questo spiega il numero di pacchetti e la dimensione molto elevata.

Nota: per maggiori informazioni riguardo i .csv di analisi, fare riferimento ai dati raccolti dalla corrispettiva audiocall.

GOMEETING	
Number of packets	346 K
File size	314 MB
Data byte rate	308 kBps
Average packet size	890,63 bytes
Average packet rate	346 packets/s

4.3.5 Zoom

Le attività svolte in questo webinar sono state analoghe a quelle di WebEx.

Nota: per maggiori informazioni riguardo i .csv di analisi, fare riferimento ai dati raccolti dalla corrispettiva audiocall.

ZOOM	
Number of packets	108 K
File size	45 MB
Data byte rate	43 kBps
Average packet size	407,83 bytes
Average packet rate	107 packets/s

Conclusioni

In questa parte finale dell'elaborato si vuole porre attenzione alle differenze e alle analogie. Tra le analogie, troviamo la presenza nei file HTTP di quasi tutte le catture di un server di tipo *connectivity check*, utilizzato dai dispositivi Android (in questo caso quello utilizzato per la cattura) per controllare se un utente ha l'accesso a Internet sulla rete a cui è connesso. Le chiamate a questi server rappresentano anche le uniche connessioni HTTP presenti (*http.host*) nelle nostre catture poiché le restanti sono tutte HTTPS. HTTPS si differenzia da HTTP per l'utilizzo del certificato SSL/TLS.
Per quanto riguarda il DNS, in alcuni file non ci sono state query, segno che le risoluzioni DNS erano già presente nel local DNS server.
Invece, erano praticamente onnipresenti chiamate a DNS over TLS su porto 853, un particolare tipo di DNS che veicola le richieste su un canale cifrato.
In alcuni casi, queste connessioni duravano per tutta la cattura (Discord).
Il comando whois non è riuscito in alcuni casi a fornirci il nome dell'organizzazione, probabilmente perché gli indirizzi che abbiamo richiesto erano protetti.
Come ci si aspettava, la chat è stata la cattura con la minor dimensione e il minor rate, in quanto ci si scambia solo caratteri ASCII in modo asincrono mentre le audiocall rappresentavano una via di mezzo tra le chat e i webinar, che avevano dimensioni e rates nettamente superiori rispetto alle altre attività.

Bibliografia

[1] Ionos.it, Cos'è netstat e come funziona? - IONOS , 24/11/21

[2] rfc-edito.org, RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions (rfc-editor.org) , 24/11/21

[3] Gorlani.com, https://www.gorlani.com/articles/dns.php, 29/11/21

[4] datatracker.ietf.org, https://www.gorlani.com/articles/dns.php, 29/11/21

[5] yoodley.com, What Is Gstatic? Everything You Need To Know | Yoodley, 04/12/21

[6] it-swarm.it, https://www.it-swarm.it/it/http/cose-lintestazione-host-http/830947383/, 10/12/21