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 2021/2022

Professore

Prof. Antonio Pescapè

Gruppo – **Petracca-Pepe-Motta**

Luca Petracca
Matr. N46004251

Gianluca Pepe
Matr. N46004302

Riccardo Motta
Matr. N46004668

Indice

Sommario

L’elaborato qui illustrato è un progetto di Cattura e Analisi di traffico Internet generato da dispositivi mobili. L’elaborazione segue diverse fasi: in una prima fase preparatoria, è stato generato del traffico utilizzando applicazioni mobile su terminali Android.

Questo è stato catturato tramite appositi sistemi di cattura forniti dal laboratorio ARCLAB della facoltà.

La successiva fase riguarda la classificazione del traffico e l’analisi dei risultati che è stata operata tramite lo sviluppo di un apposito script che automatizza le procedure.

Capitolo 1: Classificazione del traffico

1.1 Contestualizzazione del problema

Il recente sviluppo delle tecnologie web e la nascita degli smartphone hanno cambiato considerevolmente il traffico di rete. Mentre pochi anni fa il traffico Internet era prevalentemente incentrato sulla visita di pagine web, scambio di posta elettronica o trasferimento di file, oggi siamo perennemente connessi. Acquistiamo beni online, utilizzando servizi bancari digitali, messaggiamo i nostri amici e guardiamo i nostri show preferiti su televisori connessi ad Internet, le cosiddette smart tv.

A causa di questa crescita esponenziale di host connessi alle reti, la sfida è quella di essere in grado di classificare il traffico di rete, cioè associare flussi di dati alle app che li generano per capire quali protocolli viaggiano su una rete.

1.2 Classificazione del traffico mobile

L'analisi dei flussi e l'identificazione del traffico Internet avvengono tramite l'osservazione dei pacchetti IP che costituiscono il flusso stesso. Questa classificazione è resa più ardua a causa del crescente utilizzo di protocolli cifrati (TLS), che rendono impossibile utilizzare metodi di Deep Packet Inspection, e dall'enorme numero di app che generano traffico. Sorgono inoltre una serie di problematiche derivanti dal tentativo di analizzare e classificare il traffico quali ad esempio questioni legate alla privacy o ad eventuali terze parti malevole.

A fronte dei numerosi tentativi, realizzati da più parti, tesi ad una capillare analisi e classificazione del traffico mobile, assistiamo ad una presa di posizione opposta, da parte di chi, invece, custodisce gelosamente la propria applicazione e il proprio anonimato. Cio' rende particolarmente complessa tale classificazione, soprattutto a causa dell' utilizzo di protocolli cifrati, come i TLS, che ostacolano la Deep Packet Inspection nel traffico prodotto da un numero crescente di applicazioni, per non parlare dei limiti imposti dalla normativa sulla Privacy, sempre più stringente e dell' eventuale intervento di hacker.

1.2.1 Principali tecniche di classificazione

Port based:

La tecnica del Port based si basa sull'assegnazione di una porta di default per ogni protocollo (DNS su porta 53, SMTP su porta 25 ecc...) per poter poi analizzare il traffico basandosi su tale associazione. Tale tecnica è, però, ormai obsoleta, poiché presenta numerosi difetti: molte applicazioni, infatti, girano anche su porte diverse da quella associata al protocollo prescelto inizialmente o addirittura non associate ad alcun protocollo, il che inficia la precisione dell' analisi.

Packet based:

La tecnica Packet based, invece, analizza in maniera esclusiva il singolo pacchetto, separandolo dagli altri, utilizzando principi sperimentati con successo in software come Wireshark e simili e riuscendo a raccogliere numerose informazioni attraverso una capillare analisi. Ma, purtroppo anche questa metodologia ha un limite: essa, infatti, non consente la visione complessiva delle applicazioni operative al momento dell'analisi.

Flow based:

Nella tecnica Flow based, invece, si considera un flusso definito da una quintupla: Source IP address, Destination IP address, Source Port, Destination Port, Transport Protocol ID. La classificazione assomma le informazioni relative al singolo pacchetto e quelle del flusso. Nonostante sia necessario un lavoro suppletivo per salvare tutte le informazioni del flusso, tale tecnica risulta particolarmente accurata, consentendo il riconoscimento di molti protocolli, non solo quelli legati al singolo pacchetto. Anche il metodo Deep Packet Inspection (DPI), utilizzando il concetto di flusso, ha acquisito maggiore efficacia, potendo coniugare le informazioni estrapolate dal singolo pacchetto con quelle del flusso.

Payload based:

Il Deep Packed Inspection si basa sulla tecnica Payload based. Essa classifica il protocollo in base ai dati del pacchetto, tenendo conto delle caratteristiche applicative peculiari del protocollo, estrapolate attraverso l'uso di dissectors, correlando i pacchetti dello stesso flusso. Volendo esemplificare il processo, possiamo dire che i dissector decodificano parte del payload, basandosi sul proprio protocollo; se la parte analizzata coincide con il codice del dissector, il pacchetto e il relativo flusso vengono associati al protocollo. In caso contrario, l' attività viene demandata al successivo dissector. Una simile analisi del livello applicativo consente di ottenere informazioni anche riguardo Query DNS e certificati SSL, oltre al fatto di rendere possibile il riconoscimento di una applicazione tramite il suo host name contenuto, evitando in tal modo le problematiche legate alla cifratura e rendendo, nel contempo, possibile anche la classificazione del flusso delle app basandosi sul nome del provider estratto. Va ricordato, però, che tale analisi del payload non esclude ma valorizza le tecniche precedenti, con l' unico obiettivo di rendere la classificazione il più precisa possibile.

1.3 Motivazioni e Limiti

I motivi che giustificano tanto impegno profuso nella classificazione del traffico sono legati alla necessità di gestire e monitorare la rete, garantendo la qualità del servizio offerto. Ma, inoltre, essa fornisce informazioni preziose anche per numerosi operatori come advertiser, compagnie assicurative o della sicurezza, sempre nel rispetto della normativa sulla privacy, che limita l' analisi del traffico attraverso protocolli cifrati e altri accorgimenti.

Capitolo 2: Strumenti utilizzati

2.1 Capinfos

Il programma Capinfos legge un file di cattura e restituisce statistiche su ciascun file in uno dei due formati di output, ovvero *lungo* o *tabella*.
Una volta specificato quale dei due formati si preferisce utilizzare e quale tipo di statistica si preferisce visualizzare, è possibile ottenere le informazioni desiderate specificando dei flag che corrispondono al tipo di report preferito. Nel caso in cui non si specificasse alcuna opzione, il programma fornirà la statistica disponibile in formato *lungo*.

2.2 TShark

Il programma TShark è un analizzatore di protocollo di rete. Esso rende possibile l'acquisizione dei dati dei pacchetti da una rete live, come pure la lettura dei pacchetti da file precedentemente salvati, attraverso la stampa di tali pacchetti decodificati, oppure trascrivendoli su un file. Nel caso in cui non si impostasse nessuna preferenza, il programma utilizzerà la libreria pcap, acquisendo il traffico dalla prima interfaccia di rete disponibile e restituendo una riga di riepilogo per ogni pacchetto ricevuto.

2.2.1 Le principali opzioni utilizzate

-r consente di specificare il file di acquisizione dal quale TShark potrà leggere i pacchetti, restituendo una riga di riepilogo per ogni pacchetto letto.

-e consente di aggiungere un campo all'elenco dei campi da visualizzare qualora - T fields fosse selezionato.

-qz conv,type[,filter] rende possibile la creazione di una tabella che raggruppi tutte le conversazioni visualizzate durante l' acquisizione, specificando i tipi di endpoint di conversazione sui quali basare le statistiche(ad esempio TCP o UDP).

-Y consente l'abilitazione di un filtro che permetta di visualizzare solo i pacchetti che superano il filtro.

2.3 Whois

Il protocollo di rete Whois consente al client, attraverso il query di appositi database server (che raccolgono tutte le informazioni relative ai titolari dei nomi a dominio) di risalire a quale internet provider faccia capo un determinato indirizzo IP o uno specifico DNS. Esso permette, inoltre, di ottenere ulteriori informazioni riguardanti l'intestatario, la data di registrazione o di scadenza di un dominio. Tradizionalmente viene utilizzato da riga di comando.

Capitolo 3: Cattura e analisi del traffico mobile

3.1 Cattura del traffico

Per la cattura, a turni, ci siamo recati presso il laboratorio ***ARCLAB***. Con l'ausilio dello strumento di cattura ***MIRAGE***, fornito dal laboratorio, e con l'aiuto dei ricercatori abbiamo effettuato la cattura del traffico di rete sulle seguenti app: ***Skype, Meet, Messenger, Slack, GoToMeeting e Discord***.

Per ogni applicazione abbiamo effettuato un totale di 4 catture attraverso l'applicazione Traffic Research.

Per ogni cattura il sistema di cattura ha generato un file .pcap, che conteneva tutte le informazioni di rete da sottoporre ad analisi e anche altri file utili per velocizzare tale processo.

3.2 Fase di analisi

Nell'ambito della fase di analisi, per ogni traccia .pcap abbiamo:

. Estratto le informazioni preliminari
. Segmentato le catture in biflussi TCP e UDP
. Analizzato il traffico
. Ricercato informazioni relative alla compagnia intestataria dell'indirizzo IP

In particolare, l' analisi si è sviluppata nelle seguenti fasi:

3.2.1 Estrazione delle informazioni preliminari della traccia

Per effettuare questa operazione abbiamo utilizzato il tool *capinfos,* che appartiene alla famiglia di tool tshark e ci ha consentito di estrarre informazioni preliminari per ogni traccia, quali: la grandezza del file .pcap, la durata della cattura, l'ora e la data del primo e dell'ultimo pacchetto inviato.

Skype (audiocall e videocall):

```
SKYPE (LUCA)
File name:
/home/so/Scaricati/Luca/RDC_G009/RDC_G009/1634286724/traffic.pcap
File type:        Wireshark/tcpdump/... - pcap
File encapsulation:  Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 262144 bytes
Number of packets:   98 k
File size:        16 MB
Data size:        14 MB
Capture duration:    925,334438 seconds
First packet time:   2021-10-15 10:32:10,974414
Last packet time:    2021-10-15 10:47:36,308852
Data byte rate:      16 kBps
Data bit rate:       129 kbps
Average packet size: 152,31 bytes
Average packet rate: 105 packets/s
SHA256:
47ab45adda1386184b1303d084074976114e78083c544237ec12f36ebc4c6177
RIPEMD160:        1092f936b1bc3d9073af2ac43b2d94bcfcc8a206
SHA1:             d5e02f2cfb68c0d139d11e25a2f3858695c28cfa
Strict time order:   False
Number of interfaces in file: 1
Interface #0 info:
                Encapsulation = Ethernet (1 - ether)
                Capture length = 262144
                Time precision = microseconds (6)
                Time ticks per second = 1000000
                Number of stat entries = 0
                Number of packets = 98000
```

```
SKYPE (LUCA)
File name:
/home/so/Scaricati/Luca/RDC_G009/RDC_G009/1634287801/traffic.pcap
File type:        Wireshark/tcpdump/... - pcap
File encapsulation:  Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 262144 bytes
Number of packets:   247 k
File size:        177 MB
Data size:        173 MB
Capture duration:    943,248444 seconds
First packet time:   2021-10-15 10:50:05,423966
Last packet time:    2021-10-15 11:05:48,672410
Data byte rate:      183 kBps
Data bit rate:       1.471 kbps
Average packet size: 700,68 bytes
Average packet rate: 262 packets/s
SHA256:
daf248a7a5b6dff6358735c8fee87a248f12a378072142efa613bfe8ddcf404c
RIPEMD160:        a726c0d50375cd917177b8ea8f4b2953cf343aac
SHA1:             d3f64602bbc081816fc1fb90539cd96b620f48a8
Strict time order:   False
Number of interfaces in file: 1
Interface #0 info:
                Encapsulation = Ethernet (1 - ether)
                Capture length = 262144
                Time precision = microseconds (6)
                Time ticks per second = 1000000
                Number of stat entries = 0
                Number of packets = 247620
```

Google Meet (Audiocall e videocall):

MEET (LUCA)
File name:
/home/so/Scaricati/Luca/RDC_G009/RDC_G009/1634291442/traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 103 k
File size: 16 MB
Data size: 14 MB
Capture duration: 921,660423 seconds
First packet time: 2021-10-15 11:50:47,454615
Last packet time: 2021-10-15 12:06:09,115038
Data byte rate: 15 kBps
Data bit rate: 126 kbps
Average packet size: 140,46 bytes
Average packet rate: 112 packets/s
SHA256:
07916cb6c3d81c1fa4a93f81923e65761fb8b6c2414c21cbb97f753ed75ef03e
RIPEMD160: 142a94aef6699c9f12fae498428fd18f2516dde9
SHA1: 2ac843d84205163e5be0bc0f155850fbb28eefa4
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 103535

MEET (LUCA)
File name:
/home/so/Scaricati/Luca/RDC_G009/RDC_G009/1634292499/traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 186 k
File size: 85 MB
Data size: 82 MB
Capture duration: 919,885436 seconds
First packet time: 2021-10-15 12:08:32,867107
Last packet time: 2021-10-15 12:23:52,752543
Data byte rate: 89 kBps
Data bit rate: 716 kbps
Average packet size: 443,03 bytes
Average packet rate: 202 packets/s
SHA256:
c026942ab545701dbc274fea2d828344a599f8ae8bdf8913a48eaeb96202145e
RIPEMD160: d6ab13fc35d657b2c098c7f3ac266f90ab5420ab
SHA1: 51e9670112e3f7a8bead00bb8baa37bd00779014
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 186054

GoToMeeting (Audiocall e Videocall):

GOTOMEETING (GIANLUCA)
File name: traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 101 k
File size: 17 MB
Data size: 15 MB
Capture duration: 1046,316537 seconds
First packet time: 2021-11-26 15:14:08,188096
Last packet time: 2021-11-26 15:31:34,504633
Data byte rate: 14 kBps
Data bit rate: 118 kbps
Average packet size: 151,80 bytes
Average packet rate: 97 packets/s
SHA256:
43090908f3685c03b91d937462a284e4769f96d952c9fce4d02fbe35a7c3dbbe
RIPEMD160: 02eecd18a8304a464d03f82c35871d601357f3c1
SHA1: b679466b4de47f255a29cf9cc135b01d32f00398
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 101796

GOTOMEETING (GIANLUCA)
File name: traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 265 k
File size: 138 MB
Data size: 134 MB
Capture duration: 1149,101625 seconds
First packet time: 2021-11-26 15:51:46,396113
Last packet time: 2021-11-26 16:10:55,497738
Data byte rate: 116 kBps
Data bit rate: 934 kbps
Average packet size: 506,07 bytes
Average packet rate: 230 packets/s
SHA256:
779280020e1dde739c0f2a1e8295b111bae3cd6ef885cb4e87a186b383c5780f
RIPEMD160: 25ca673b62b38f44afa3bdb6773c65f97179b7af
SHA1: 34537b80510eb92dc3538996bc29056c69b60310
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 265238

Discord (Audiocall e Videocall):

DISCORD (GIANLUCA)
File name:
/home/so/Scaricati/Gianluca/RDC_G009/1637941346/traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 49 k
File size: 11 MB
Data size: 10 MB
Capture duration: 911,655955 seconds
First packet time: 2021-11-26 16:42:33,244097
Last packet time: 2021-11-26 16:57:44,900052
Data byte rate: 11 kBps
Data bit rate: 94 kbps
Average packet size: 217,55 bytes
Average packet rate: 54 packets/s
SHA256:
676e653e80c3b1cc7acd54a51b473cac1da8dcab038e461eaf79d2a18413e991
RIPEMD160: 75fa4d8ab9b1cfc4616a87a587072ed250ca40e7
SHA1: 5c0aa421a91e5506fac19a8aab2aba3c2245f434
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 49599

DISCORD (GIANLUCA)
File name:
/home/so/Scaricati/Gianluca/RDC_G009/1637943450/traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 575 k
File size: 604 MB
Data size: 595 MB
Capture duration: 944,044435 seconds
First packet time: 2021-11-26 17:17:35,131605
Last packet time: 2021-11-26 17:33:19,176040
Data byte rate: 630 kBps
Data bit rate: 5.044 kbps
Average packet size: 1034,15 bytes
Average packet rate: 609 packets/s
SHA256:
29e1ec5dd2e59495fa1b783bbf8b605fc68fa0016c05b23fbe760769a50720a1
RIPEMD160: 0a5c42b23be080aa8b6f0774b71bca88912431b4
SHA1: b740880d71eaf5e5ccf0f0affc180c48cb6b173b
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 575606

Slack (Videocall):

SLACK (RICCARDO)
File name:
/home/so/Scaricati/Riccardo/RDC_G009/RDC_G009/1637165893/traffic.pcap
File type: Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
File timestamp precision: microseconds (6)
Packet size limit: file hdr: 262144 bytes
Number of packets: 131 k
File size: 21 MB
Data size: 19 MB
Capture duration: 917,758420 seconds
First packet time: 2021-11-17 17:18:23,092712
Last packet time: 2021-11-17 17:33:40,851132
Data byte rate: 20 kBps
Data bit rate: 165 kbps
Average packet size: 144,85 bytes
Average packet rate: 142 packets/s
SHA256:
727497ab0efcbd495cc507ba28e9a4162c2dec207376fc9da65ef7b5081173c6
RIPEMD160: 619ad14d9eee33f9a876a0e02b6f9be733d207a9
SHA1: f284a85e683330e92e49511035c64afbc9e9aac8
Strict time order: False
Number of interfaces in file: 1
Interface #0 info:
Encapsulation = Ethernet (1 - ether)
Capture length = 262144
Time precision = microseconds (6)
Time ticks per second = 1000000
Number of stat entries = 0
Number of packets = 131209

3.2.2 Segmentazione delle catture in biflussi TCP e UDP

In questa fase, utilizzando il comando *tshark,* abbiamo estratto tutti i flussi bidirezionali TCP e UDP, per ogni traccia, esportando il risultato in una tabella in formato CVS, tramite il comando -qz *conv*, type[,*filter]*.

Nella tabella, ogni riga rappresenta un biflusso e per ogni biflusso abbiamo le seguenti informazioni: la quintupla (indirizzi IP, porti di livello trasporto e protocollo di livello trasporto) e il numero di pacchetti/byte in ciascuna direzione ed il numero totale di pacchetti/byte del biflusso.

TCP Conversations during a Skype video-call

	<-		->		Total		Relative	Duration
	Frames	Bytes	Frames	Bytes	Frames	Bytes	Start	
192.168.20.103:35665	<-> 52.114.88.200:443	108	40095	68	31976	176	72071	19,728959000	918,5664
192.168.20.103:48401	<-> 13.83.65.43:443	62	25222	86	72410	148	97632	9,560954000	933,6875
192.168.20.103:42533	<-> 142.250.180.106:443	78	16265	58	31263	136	47528	763,276306000	3,1666
192.168.20.103:48418	<-> 13.83.65.43:443	37	13835	48	34677	85	48512	112,142003000	827,7524
192.168.20.103:60732	<-> 51.140.202.63:443	33	11913	50	6549	83	18462	16,094955000	926,4708
192.168.20.103:41319	<-> 20.189.173.6:443	37	9466	46	49772	83	59238	30,598960000	17,1220
192.168.20.103:48403	<-> 13.83.65.43:443	32	13539	46	31603	78	45142	9,802952000	187,7081
192.168.20.103:50699	<-> 8.8.8.8:853	36	12687	38	5702	74	18389	9,296954000	41,3410
192.168.20.103:54396	<-> 40.79.197.34:443	24	6910	35	31475	59	38385	938,342387000	4,1670
192.168.20.103:44609	<-> 142.250.184.110:443	24	15248	22	6874	46	22122	511,255186000	245,4151
192.168.20.103:34427	<-> 68.232.34.200:443	20	9465	20	2422	40	11887	261,046071000	180,6158
192.168.20.103:34430	<-> 68.232.34.200:443	23	9663	17	2226	40	11889	494,645179000	180,1997
192.168.20.103:34434	<-> 68.232.34.200:443	22	9619	18	2292	40	11911	734,500291000	179,8932
192.168.20.103:40137	<-> 204.79.197.200:443	19	10382	20	1909	39	12291	14,181958000	1,7640
192.168.20.103:39591	<-> 13.107.254.128:443	19	8491	19	2422	38	10913	938,920387000	3,4060
192.168.20.103:59737	<-> 40.70.161.102:443	12	6454	25	3522	37	9976	118,236006000	708,2183
192.168.20.103:34411	<-> 68.232.34.200:443	18	9267	16	1828	34	11095	14,350962000	27,6990
192.168.20.103:44691	<-> 20.189.173.1:443	14	7882	20	7433	34	15315	56,560976000	56,6380
192.168.20.103:46396	<-> 20.189.173.14:443	15	8326	19	7041	34	15367	131,395013000	191,3880
192.168.20.103:50736	<-> 8.8.8.8:853	15	3101	18	2383	33	5484	733,991289000	49,8975
192.168.20.103:50742	<-> 8.8.8.8:853	15	8959	18	2322	33	11281	938,203384000	1,2490
192.168.20.103:48404	<-> 13.83.65.43:443	13	8735	18	6832	31	15567	9,823955000	4,8430
192.168.20.103:50725	<-> 8.8.8.8:853	13	7354	18	1892	31	9246	112,834005000	38,8136
192.168.20.103:50732	<-> 8.8.8.8:853	15	7007	16	1609	31	8616	493,983183000	37,5257
192.168.20.103:55596	<-> 51.105.197.41:443	14	7522	16	2457	30	9979	15,663955000	97,3328
192.168.20.103:50059	<-> 40.114.211.99:443	14	7413	15	2301	29	9714	14,321962000	1,6760
192.168.20.103:55597	<-> 51.105.197.41:443	14	7506	15	2375	29	9881	15,673958000	97,3408
192.168.20.103:54128	<-> 13.94.251.244:443	13	7109	14	2005	27	9114	13,316956000	2,1750
192.168.20.103:53973	<-> 52.232.209.85:443	13	6517	14	2126	27	8643	19,358957000	93,8907
192.168.20.103:50089	<-> 40.114.211.99:443	13	7359	14	2215	27	9574	939,027384000	3,2980
192.168.20.103:48405	<-> 13.83.65.43:443	11	7469	15	3957	26	11426	11,660955000	3,3560
192.168.20.103:48406	<-> 13.83.65.43:443	11	7686	15	3940	26	11626	11,673957000	3,2050
192.168.20.103:55625	<-> 51.105.197.41:443	12	7349	13	2206	25	9555	939,441384000	3,0890
192.168.20.103:55626	<-> 51.105.197.41:443	12	7333	13	2190	25	9523	939,454388000	2,8730
192.168.20.103:58878	<-> 52.178.103.218:443	10	6598	14	4188	24	10786	9,378953000	0,3389
192.168.20.103:50086	<-> 40.127.110.237:443	12	7055	12	1892	24	8947	734,119288000	126,5813
192.168.20.103:54146	<-> 13.94.251.244:443	13	8569	10	1740	23	10309	254,154072000	126,0399
192.168.20.103:54149	<-> 13.94.251.244:443	12	7055	11	1832	23	8887	494,332181000	124,9557
192.168.20.103:48417	<-> 13.83.65.43:443	10	7021	12	3589	22	10610	112,123000000	124,7000
192.168.20.103:48419	<-> 13.83.65.43:443	10	7523	12	3665	22	11188	112,168003000	123,1728
192.168.20.103:50729	<-> 8.8.8.8:853	10	1789	12	1685	22	3474	253,979068000	27,1010
192.168.20.103:50720	<-> 8.8.8.8:853	9	1232	11	1468	20	2700	56,429980000	20,1640
192.168.20.103:42532	<-> 142.250.180.106:443	9	2258	9	3148	18	5406	750,720299000	0,2000
192.168.20.103:58901	<-> 216.58.209.42:443	9	2798	8	2312	17	5110	763,857302000	0,1470
192.168.20.103:51495	<-> 13.107.4.52:80	4	748	4	402	8	1150	9,337956000	65,5228
192.168.20.103:35936	<-> 216.58.206.42:443	4	466	3	210	7	676	236,829994000	5,0451
192.168.20.103:56864	<-> 52.177.138.113:443	3	170	3	369	6	539	9,771955000	0,2177
192.168.20.103:38475	<-> 40.70.161.7:443	3	170	3	705	6	875	112,952017000	0,2107
192.168.20.103:40083	<-> 108.177.119.188:5228	2	739	2	170	4	909	42,113865000	0,2279
192.168.20.103:49872	<-> 142.250.184.106:443	2	120	2	155	4	275	235,267813000	60,0226
192.168.20.103:49873	<-> 142.250.184.106:443	2	120	2	156	4	276	235,893486000	60,0506
192.168.20.103:52252	<-> 172.217.21.74:443	2	120	2	156	4	276	236,452197000	60,2559
192.168.20.103:53924	<-> 52.232.209.85:443	0	0	3	291	3	291	9,364952000	240,6471

UDP Conversations during a Skype video-call

		<-		->		Total		Relative		Duration	
		Frames	Bytes	Frames	Bytes	Frames	Bytes	Start			
93.34.32.30:31993	<-> 192.168.20.103:29883	111681	80369046	131217	91904746	242898	172273792	20,511960000	918,1469		
192.168.20.103:51840	<-> 93.34.32.30:32013	1179	158046	1105	146314	2284	304360	32,241971000	906,6228		
192.168.20.103:38638	<-> 52.114.104.37:3478	103	21562	61	19874	164	41436	19,539964000	918,7291		
192.168.20.103:29883	<-> 20.202.4.57:3478	43	10226	49	13032	92	23258	19,464958000	917,2629		
192.168.20.103:51840	<-> 20.202.4.76:3478	43	10226	45	11836	88	22062	32,089974000	905,7003		
192.168.20.103:29883	<-> 20.202.4.102:3478	7	1461	1	110	8	1571	19,404963000	26,3120		
192.168.1.152:41614	<-> 192.168.20.103:29883	5	770	0	0	5	770	20,458956000	1,2550		
192.168.20.103:51840	<-> 20.202.4.102:3478	4	873	1	110	5	983	32,017965000	22,0207		
192.168.20.103:40084	<-> 142.250.184.110:443	0	0	5	6960	5	6960	511,254188000	4,0350		
192.168.20.103:39293	<-> 142.250.180.106:443	0	0	5	6960	5	6960	764,242313000	4,0400		
192.168.20.103:51840	<-> 192.168.1.152:38224	0	0	4	616	4	616	32,189967000	1,3220		
192.168.20.103:40804	<-> 142.250.180.106:443	0	0	4	5568	4	5568	764,397304000	2,2010		
192.168.20.103:22014	<-> 20.202.4.102:3478	2	467	1	110	3	577	19,419958000	0,0807		
192.168.20.103:22014	<-> 20.202.4.76:3478	0	0	2	484	2	484	19,466958000	916,8314		

3.2.3 Analisi del traffico

In questa fase, da ogni biflusso estratto abbiamo estrapolato informazioni (che poi abbiamo aggiunto alla tabella) riguardanti la risoluzione dns, campo SNI di TLS e host di HTTP.
Tale procedimento è stato realizzato utilizzando il comando tshark seguito da tutta una serie di opzioni e filtri.
Ad esempio, per ottenere la risoluzione dns abbiamo utilizzato il comando:

tshark -r traffic.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 dns.a -e dns.qry.name -Y "(dns.flags.response == 1)" > dns_conversations.csv

I comandi che seguono l'opzione *-T fields* costituiscono tutte le informazioni che vogliamo ricavare, come ad esempio lo stack dei protocolli, l'indirizzo dns e il query dns. Il comando che segue l'opzione *-Y* indica il filtro di visualizzazione utilizzato per filtrare i pacchetti da visualizzare, in particolare per visualizzare solamente i pacchetti con risposta dns.

DNS Conversations during a Discord audio-call:

1637941385.414888000	eth:ethertype:ip:udp:dns	8.8.8.8	192.168.20.103	172	16	58.205.68	www.google.com		
1637941385.429874000	eth:ethertype:ip:udp:dns	8.8.8.8	192.168.20.103	172	16	58.208.163	connectivitycheck.gstatic.com		
1637942027.633486000	eth:ethertype:ip:udp:dns	8.8.8.8	192.168.20.103	172	16	58.208.132	www.google.com		
1637942027.637481000	eth:ethertype:ip:udp:dns	8.8.8.8	192.168.20.103	142.250.180.67			connectivitycheck.gstatic.com		

Per l'analisi del campo SNI di TLS abbiamo utilizzato il comando:

tshark -r traffic.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 tls.handshake.extensions_server_name -Y "(tls.handshake.extensions_server_name)" > tls_conversations.csv

TLS Conversations during a Discord audio-call:

1637941355.693620000eth:ethertype:ip:tcp:tls192.168.20.103162.159.136.232644084443discord.com
1637941355.704622000eth:ethertype:ip:tcp:tls192.168.20.103162.159.133.232655764443dl.discordapp.net
1637941357.382621000eth:ethertype:ip:tcp:tls192.168.20.103162.159.130.234651788443gateway.discord.gg
1637941366.040626000eth:ethertype:ip:tcp:tls192.168.20.103162.159.129.235658331443rotterdam2968.discord.media
1637941384.534634000eth:ethertype:ip:tcp:tls192.168.20.103162.159.138.234648723443latency.discord.media
1637941385.460635000eth:ethertype:ip:tcp:tls192.168.20.103216.58.205.68651422443www.google.com
1637942027.674937000eth:ethertype:ip:tcp:tls192.168.20.103216.58.208.132659243443www.google.com
1637942255.576039000eth:ethertype:ip:tcp:tls192.168.20.103162.159.128.233640938443discord.com

Per l'analisi delle richieste HTTP con host valido abbiamo utilizzato il comando:

tshark -r traffic.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)" > http_conversations.csv

HTTP Conversations during a Slack video-call:

1637165937.890721000eth:ethertype:ip:tcp:http192.168.20.102216.58.205.6765487480connectivitycheck.gstatic.com
1637166204.005844000eth:ethertype:ip:tcp:http192.168.20.102142.250.184.3564804180connectivitycheck.gstatic.com
1637166734.846096000eth:ethertype:ip:tcp:http192.168.20.102142.250.184.3564851880connectivitycheck.gstatic.com

3.2.4 Ricerca di informazioni sulla compagnia intestataria dell'indirizzo IP

In questa ultima fase, abbiamo individuato l'indirizzo IP di destinazione di ogni biflusso e lo abbiamo passato al comando *whois*. In tal modo, siamo riusciti ad ottenere ulteriori informazioni relativamente alla compagnia intestataria dell' indirizzo IP, che abbiamo inserito nella tabella.

ES: GoToMeeting

```
Bulk mode; whois.cymru.com [2021-11-26 15:11:03 +0000]
13335   | 104.17.209.240   | CLOUDFLARENET, US
14618   | 23.22.189.111    | AMAZON-AES, US
14618   | 34.206.99.180    | AMAZON-AES, US
14618   | 34.235.193.186   | AMAZON-AES, US
14618   | 54.146.88.2      | AMAZON-AES, US
14618   | 54.175.121.155   | AMAZON-AES, US
15169   | 142.250.180.100  | GOOGLE, US
15169   | 142.250.180.131  | GOOGLE, US
15169   | 142.250.184.35   | GOOGLE, US
15169   | 216.58.205.68    | GOOGLE, US
15169   | 35.186.241.51    | GOOGLE, US
15169   | 35.190.25.25     | GOOGLE, US
15169   | 8.8.8.8          | GOOGLE, US
16509   | 13.226.175.112   | AMAZON-02, US
16509   | 13.226.175.62    | AMAZON-02, US
16509   | 143.204.11.15    | AMAZON-02, US
16509   | 143.204.11.7     | AMAZON-02, US
16509   | 35.165.244.87    | AMAZON-02, US
16509   | 52.10.63.86      | AMAZON-02, US
16509   | 68.64.5.152      | AMAZON-02, US
16509   | 68.64.5.203      | AMAZON-02, US
16509   | 78.108.124.37    | AMAZON-02, US
16509   | 78.108.124.39    | AMAZON-02, US
16509   | 99.86.162.45     | AMAZON-02, US
16815   | 68.64.22.20      | GOTO-PRIMARY-AS, US
16815   | 68.64.29.231     | GOTO-PRIMARY-AS, US
```

3.3 Script per l'automazione delle analisi

```
#!/bin/bash
echo "Benvenuto!"
read junk
sync
echo "Premi invio per generare il file dei biflussi TCP"
read junk
sync
tshark -r traffic.pcap -qz conv,tcp > tcp_conversations.csv
echo "OK.. Generato"
echo "Premi invio per generare il file dei biflussi UDP"
read junk
sync
tshark -r traffic.pcap -qz conv,udp > udp_conversations.csv
echo "OK.. Generato"
echo "Premi invio per generare il file delle richieste DNS"
read junk
sync
tshark -r traffic.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 dns.a -e dns.qry.name -Y "(dns.flags.response == 1)" > dns_conversations.csv
echo "OK.. Generato"
echo "Premi invio per generare il file delle richieste SSL con campo SNI valido"
read junk
sync
tshark -r traffic.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 tls.handshake.extensions_server_name -Y "(tls.handshake.extensions_server_name)" > tls_conversations.csv
echo "OK.. Generato"
echo "Premi invio per generare il file delle richieste HTTP con campo host valido"
read junk
sync
tshark -r traffic.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)" > http_conversations.csv
echo "OK.. Generato"
```

3.3.1 Funzionamento dello script

Per poter utilizzare lo script, è necessario posizionarlo nella cartella contenente i file da analizzare ed eseguirlo nella shell con il comando ./script_tshark

Capitolo 4: Risultati sperimentali

I dati estratti dallo strumento capinfos mostrano risultati evidenti: l'applicazione Discord genera pacchetti con una dimensione media considerevolmente più elevata rispetto alle altre applicazioni trattate. Mentre le altre applicazioni sono pensate per un utilizzo "da ufficio", Discord si rivolge prevalentemente ad un pubblico di videogiocatori. Risulta quindi necessario ottimizzare la qualità trasmissiva dell' audio e del video in un ambiente in cui parametri come la latenza giocano un ruolo fondamentale.

Si noti, inoltre, che vi è continuità tra tutte le applicazioni nella differenza tra le dimensioni di file scambiati durante le video-call rispetto alle audio-call, con le prime considerevolmente più pesanti delle seconde. Ciò è dovuto, banalmente, all'aggiunta del video oltre all'audio.

Nella fase di analisi delle connessioni, risulta evidente come ogni applicazione utilizzata fa uso in ugual modo dei protocolli TCP e UDP.

Un ulteriore risultato significativo riguarda la risoluzione DNS. Si noti come, in seguito alla creazione della tabella CSV, in alcuni casi il risultato sia nullo, in quanto le risposte DNS hanno fatto riferimento alla cache DNS del dispositivo mobile, di conseguenza non hanno effettuato traffico con il server DNS.

Esempio risoluzione DNS con risultato nullo (Meet video-call):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														

Esempio risoluzione DNS con valori (Discord video-call):

	A	B	C	D	E	F	G	H	I	J	K	L
1	1637943475.328604000eth:ethertype:ip:udp:dns8.8.8.8192.168.20.10317216.58.206.35connectivitycheck.gstatic.com											
2	1637943638.663652000eth:ethertype:ip:udp:dns8.8.8.8192.168.20.10317216.58.205.68www.google.com											
3	1637943961.278687000eth:ethertype:ip:udp:dns8.8.8.8192.168.20.10317142.250.180.164www.google.com											
4	1637943961.290715000eth:ethertype:ip:udp:dns8.8.8.8192.168.20.10317142.250.180.67connectivitycheck.gstatic.com											
5												