Università degli Studi di Genova

Facoltà di Ingegneria

Laurea Triennale in Ingegneria Informatica

Identificazione e mascheramento del traffico Voip

Monica Curti

Relatore:

Chiar. Mo Prof. Ing. Mauro Migliardi

Corelatore:

Chiar. Mo Ing. Giorgio Ravera

17 Dicembre 2010

Indice

Ringraziamenti

Una laurea è un piccolo traguardo, o forse, come insegna un padre, è un punto di inizio.

Sono i miei genitori i primi a meritare un reale ringraziamento, ottimi educatori, perchè è risaputo, un figlio impara fin da piccolo assimilando ed imitando. Così, loro, predisposti all'idea del sacrificio, perchè su di esso han creato tutto ciò che mi circonda, mi han indirizzata a concepire lo studio quale forma di lusso, mi han parlato di futuro come se ne parla ad una persona adulta ancor prima che adulta divenissi, han reso la lora vita complice della mia.

Valentina, mia gemella, ha un significato incommensurabile nella mia realtà e nelle mie scelte. E' il mio alterego, la persona che cammina parallelamente a me con sfumature di vita e scelte differenti, che si incrociano nello stesso percorso formativo.

Elena, mia sorella maggiore, per cui nutro una gran stima per la risolutezza che le appartiene sia in campo professionale che non.

Un ringraziamento particolare è rivolto al mio Relatore, Prof. Ing. Mauro Migliardi, e al mio Corelatore, Ing. Giorgio Ravera, che mi hanno offerto la possibilità di crescere sotto l'aspetto professionale, si sono mostrati impeccabili sotto l'aspetto formativo, e mi han proposto un lavoro di tirocinio che ha coinvolto appieno i miei interessi.

Mattia Monti, Simone Schiappacasse, Giorgio Ravera, sono state le persone che mi hanno appoggiato e sostenuto senza mai smetter di credere in me, instradandomi verso una grande apertura mentale e partecipando alla mia formazione e crescita sotto ogni aspetto.

Insieme a loro, le mie migliori amiche, Luisa Guizzardi e Federica Porta, le persone che nei momenti di pausa, davanti ad un solo caffè così come ad una serata impegnativa, han reso i miei occhi capaci di sorridere.

Ringrazio i miei innumerevoli coetanei che han reso questi anni motivo di divertimento, regalandomi una vita sociale eccezionale, a cui non ho mai rinunciato, nonostante le piccole privazioni che tale carriera formativa implichi di norma.

Infine, voglio ringraziare i miei compagni di corso, semplici e mai competitivi, ma costruttivi, che han dato luce ad un ambiente magnifico sotto l'aspetto umano.

Obiettivi

Lo scopo di questa tesi è presentare metodologie atte ad identificare il traffico Voip in un flusso dati e delle metodologie atte a rendere inefficaci quanto precedentemente presentato.

Tramite Wireshark, un software che permette di osservare ed analizzare il traffico di rete, si potranno rilevare i flussi di dati e salvarli in un file di formato standard, pcap, che verrà descritto in seguito.

Quindi, dopo aver effettuato la cattura del traffico di rete, sarà possibile passare ad una fase di analisi dello stesso e identificare il traffico Voip tra gli altri protocolli di comunicazione.

Ciò sarà reso possibile tramite caratteristiche peculiari del traffico voce ricercabili nello scambio dati attraverso un' analisi statistica.

Una volta ottenuti i parametri e i modelli statistici che permettono di identificare il traffico voce, agiremo su tali parametri modificandoli opportunamente allo scopo di mascherare le comunicazioni VoIP, così da far apparire tale traffico di natura diversa.

Cenni Storici

Il termine VoIP é l'acronimo inglese di "Voice Over Internet Protocol", in Italiano "Voce su Protocollo Internet" [R6].

E' una tecnologia che consente di effettuare telefonate utilizzando una connessione internet e senza transitare sulla Rete telefonica tradizionale.

La sua nascita si fa risalire al 1995, quando alcuni appassionati informatici israeliani sperimentarono la prima comunicazione voce tra due computer.

Nello stesso anno la tecnologia fu inserita in un software noto come "Internet Phone Software": per parlare da PC a PC era sufficiente un modem, una scheda audio, altoparlanti e microfono.

Attraverso la compressione e digitalizzazione del segnale audio, la voce viaggiava lungo la rete internet sottoforma di "pacchetti dati". All'inizio la qualità audio era scarsa e certamente inferiore a quella delle tradizionali linee telefoniche, ma la tecnologia migliorò rapidamente.

Nel 1998, in particolare, furono introdotti i primi gateway di interconnessione per abilitare le chiamate da PC alle linee telefoniche tradizionali.

L'utilizzo del VoIP sta crescendo esponenzialmente, grazie alla rapida diffusione delle connessioni ADSL a banda larga e ad applicativi di semplice utilizzo (es. Skype) che oggi consentono di chiamare in tutto il mondo gratis o con una spesa irrisoria.

Nato in origine per consentire le chiamate da PC a PC, il VoIP si é rapidamente evoluto per permettere di ricevere ed effettuare chiamate sulla rete telefonica tradizionale.

Per poter utilizzare la tecnologia era necessario, fino a qualche anno fa, munirsi di cuffiette e di microfono da collegare al proprio computer.

Da alcuni anni, per favorire la diffusione del VoIP, aumentandone l'accessibilità da parte degli utenti, sono stati commercializzati apparecchi telefonici, simili ai tradizionali cordless, che si possono collegare direttamente al PC (generalmente attraverso la porta USB) e che consentono di fruire del servizio telefonico con una modalità molto simile a quella tradizionale.

Questi apparecchi richiedono comunque, oltre ad una connessione internet ADSL, la necessità di disporre di un PC e di un software per il VoIP (es. Skype).

Più recentemente il VoIP è diventata una tecnologia accessibile anche in assenza di un PC.

E' sufficiente disporre di una connessione internet a banda larga e di un apposito abbonamento fornito da molti operatori (es. Tiscali, Infostrada, etc.). A questo punto, si collega direttamente il telefono di casa a un apparato che integri un modem/router ADSL e un gateway SIP (o protocollo analogo), oppure ci si procura direttamente un semplice VoIP phone, (diverso da un telefono tradizionale, sia esso analogico o ISDN) e si sfrutta il VoIP per telefonare in tutto il mondo a prezzi estremamente vantaggiosi.

Capitolo 1 VoIP

Voice over IP (*Voice over Internet Protocol*), acronimo **VoIP**, è una tecnologia che rende possibile effettuare una conversazione telefonica sfruttando una connessione Internet, o un'altra rete dedicata che utilizza il protocollo IP. Più specificatamente, col termine VoIP s'intende l'insieme dei protocolli di comunicazione di strato applicativo e di sessione, che rendono possibile tale tipo di comunicazione. Grazie a numerosi provider VoIP, è possibile effettuare telefonate anche verso la rete telefonica tradizionale (PSTN). Il vantaggio principale di questa tecnologia sta nel fatto che essa elimina l'obbligo di riservare della banda per ogni telefonata (commutazione di circuito), sfruttando l'allocazione dinamica delle risorse, caratteristica del protocollo IP (commutazione di pacchetto). Vengono instradati sulla rete pacchetti di dati contenenti le informazioni vocali, codificati in forma digitale, e ciò solo nel momento in cui è necessario, cioè quando uno degli utenti collegati sta parlando.

La tecnologia VoIP consente di trasformare un segnale audio (la voce) in formato digitale e di trasmetterlo, compresso, sottoforma di "pacchetti dati" attraverso Internet (VoIP = Voce su Protocollo Internet).

I "pacchetti di dati" compressi contengono, oltre alle informazioni relative alla voce, anche altre informazioni relative all'origine, alla destinazione e all'ora di invio dei pacchetti. Tali informazioni consentono di "ricostruire" correttamente la voce quando i pacchetti di dati raggiungono il destinatario e sono riconvertiti in segnale audio analogico.

In questo modo il nostro interlocutore che sta dall'altro capo del telefono/computer comprende il messaggio trasmesso.

Per evitare l'effetto "ritardo" nella comunicazione, é indispensabile che i pacchetti di dati viaggino ad una velocità molto elevata.

Per apprezzare al meglio la tecnologia VoIP é necessario disporre di una connessione Internet a banda larga (es. ADSL) con una latenza verso il proprio gestore bassa (massimo 150ms) e stabile.

In questo modo é possibile effettuare chiamate in tutto il mondo gratis o a prezzi estremamente convenienti.

Oggi il VoIP é diventato un enorme business sia per le compagnie telefoniche tradizionali sia per nuove aziende specializzate che offrono servizi dedicati a privati e ad imprese.

Sono disponibili offerte, spesso gratuite o a prezzi estremamente convenienti, per telefonare dal PC e dal telefono di casa.

1.1. Perché il VoIP conviene ?

Per gli utenti la convenienza é legata alla possibilità di abbattere drasticamente i costi delle bollette telefoniche, o di azzerarle del tutto effettuando chiamate da computer a computer (o da telefono verso utenti con lo stesso abbonamento VoIP).

E' naturale chiedersi perché la tecnologia VoIP (basata sulla commutazione di pacchetto) consenta di ottenere simili risparmi rispetto alla tecnologia a commutazione di circuito (cioè la rete telefonica tradizionale).

Le ragioni sono principalmente le seguenti:

a) minore richiesta di banda a parità di qualità (il VoIP, infatti, non necessita di un circuito chiuso e dedicato);

b) ridotti costi di chiamata, specialmente per quelle a lunga distanza; la chiamata telefonica fatta con il VoIP, infatti, é pari al costo di una chiamata locale per collegarsi al proprio provider Internet;

c) riduzione dei costi di infrastruttura. Infatti, non serve più un cavo apposito per ogni telefono e si utilizza la porta di rete di un computer, mentre i terminali software si installano direttamente nei PC. Non occorre ri-cablare una linea telefonica in caso di trasloco di interni [R10].

Oggi moltissimi software, scaricabili da Internet, consentono a qualunque persona dotata di un PC collegato ad Internet di chiamare e videochiamare gratuitamente in tutto il mondo un'altra persona anch'essa munita di PC e connessione a Internet.

Ciò è possibile proprio grazie alla tecnologia VoIP, che consente la trasmissione della voce e del segnale video (qualora si scelga di effettuare una videochiamata) attraverso la connessione Internet del proprio PC.

Tra i software più diffusi si possono ricordare Skype , Google Talk e Msn Messenger.

In particolare, molti programmi, nati in origine solo per consentire la chat testuale (software di Instant Messaging, come per es. Msn Messenger), sono stati successivamente arricchiti per consentire di effettuare chiamate vocali e videochiamate.

Alcuni di questi programmi (es. Skype) consentono oggi di effettuare chiamate dal proprio PC verso i numeri telefonici tradizionali di rete fissa e mobile pagando soltanto il costo di interconnessione con l'operatore telefonico e consentendo significativi risparmi all'utente finale.

Le conversazioni VoIP non devono necessariamente viaggiare su Internet, ma possono anche essere veicolate su una qualsiasi rete privata basata sul protocollo IP, per esempio una LAN all'interno di un edificio o di un gruppo di edifici.

Uno dei vantaggi di questa tecnologia è che permette di fare leva su risorse di rete preesistenti, consentendo una notevole riduzione dei costi in ambito sia privato che aziendale, specialmente per quanto riguarda le spese di comunicazione interaziendali e tra sedi diverse. Una rete aziendale, infatti, può essere sfruttata anche per le comunicazioni vocali, permettendo di semplificare l'installazione e il supporto e di aumentare il grado di integrazione di uffici dislocati sul territorio, ma collegati tramite l'infrastruttura di rete. Il consumatore privato, utilizzando un collegamento ad Internet a banda larga (ad alta velocità e sempre attivo), può effettuare e ricevere chiamate telefoniche, potendo contare su tariffe molto economiche, soprattutto per le chiamate internazionali.

La tecnologia VoIP introduce, inoltre, nuove possibilità per l'offerta del servizio telefonico, quali: eliminare la distinzione tra chiamate locali ed a lunga distanza; mantenere diversi numeri telefonici su un solo collegamento; salvare messaggi vocali

sul proprio computer; permettere telefonate completamente gratuite tra utenti dello stesso fornitore.

1.2. Il VoIP è sicuro?

Il VoIP può offrire un significativo vantaggio per la tutela della privacy.

Utilizzando una connessione via Internet, è possibile adottare una comunicazione sicura, con crittografia a chiave asimmetrica e firma digitale, che impedisce a terzi di intercettare la conversazione e manipolarla (a meno che siano in possesso di una grande potenza di calcolo).

Recentemente, gli è stato accostato SX for Skype, una applicazione sviluppata proprio per

rispondere ai sempre piu' fastidiosi spammer presenti su Skype [R11].

SX for Skype permette di filtrare i contenuti dell'applicazione VoIP in diversi modi: filtro dei contenuti, filtro antispam e phishing. Il primo filtra automaticamente i messaggi ritenuti non idonei, ed e' possibile impostareil livello di controllo dei contenuti, da minimo a massimo. Il filtro antispam come suggerisce il nome serve per evitare di ricevere messaggi di tipo spam, mentre il filtro phising impedisce l'utente di mettersi in contatti con utenti ritenuti pericolosi.

Le reti IP non dispongono di per sé di alcun meccanismo in grado di garantire che i pacchetti di dati vengano ricevuti nello stesso ordine in cui vengono trasmessi, né alcuna garanzia relativa in generale alla qualità di servizio. Le attuali applicazioni nel mondo reale della telefonia VoIP si trovano a dover affrontare problematiche legate a problemi di lentezza (sostanzialmente si deve ridurre il tempo di transito e di elaborazione dei dati durante le conversazioni) e di integrità dei dati (prevenire perdite e danneggiamenti delle informazioni contenute nei pacchetti).

Il problema di fondo della tecnologia VoIP è la corretta ricostruzione dei pacchetti di dati ricevuti, tenuto conto del fatto che durante la trasmissione può cambiare la sequenza dei pacchetti e che alcuni pacchetti possono aver subito perdite o danneggiamenti delle informazioni contenute, e assicurare così che lo stream audio (*flusso audio*) mantenga la corretta coerenza temporale. Altro importante problema è

mantenere il tempo di latenza dei pacchetti sufficientemente basso, in modo che l'utente non debba aspettare troppo tempo prima di ricevere le risposte durante le conversazione.

Capitolo 2 Traffico Skype

Un esempio di applicazione VoIP è Skype.

Sviluppato nel 2002 dai creatori di Kazaa, si rivela oggi l'applicazione VoIP più popolare e sta acquistando le attenzioni della comunità di ricerca così come degli operatori Telecom.

Il suo traffico può esser generato tra due end-hosts (E2E, *End-to-End*), ciascuno dei quali risulta un client di Skype; altrimenti può esser procurato da una comunicazione tra un "endhost" e un tradizonale telefono PSTN (E2O, *End-to-Out*), coinvolto nei servizi Skypeout/Skypein.

2.1. Il motivo del suo successo

L'applicazione Skype è basata su un paradigma di comunicazione peer-to-peer.

A differenza dei servizi dei tradizionali PSTNs, i quali utilizzano una tariffa al minuto per le chiamate variabile in relazione all'itinerario, le chiamate Voip confidano su servizi che sfruttano il protocollo TCP/IP eliminando la distinzione tra chiamate locali ed a lunga distanza.

Le chiamate tramite Voip risultano molto più economiche delle tradizionali chiamate a lunga distanza per utenti PSTN (Public Switched Telephone Network) o addirittura gratuite se avvengono direttamente tra due utenti Voip dello stesso provider.

Skype può facilmente lavorare in molti ambienti di rete differenti senza una configurazione aggiuntiva dell'utente.

Offre la possibilità di investigare automaticamente le caratteristiche di rete e sfruttare le migliori opzioni avviabili per inoltrare il suo traffico.

Molte domande interessanti relative ai suoi meccanismi interni, al traffico che genera e al comportamento dei suoi utenti rimangono ancora senza risposta.

La complessità sta nel fatto che i protocolli di Skype sono “proprietari”, ovvero sono sviluppati indipendentemente dalla singola azienda, Nonostante ciò, se si usano sistemi VoIP che usano standard aperti come quello SIP, è possibile combinare qualsiasi terminale con gli altri componenti per avere le funzionalità avanzate, senza la necessità che tali componenti siano proprietari.Inoltre vi è un estensivo uso di crittografia e offuscazione.

Nonostante quanto detto in precedenza, è possibile individuare chiamate nascoste nel traffico Web attraverso il rilevamento della presenza di flussi che differiscono, in un certo numero di caratteristiche chiavi, dal comportamento del traffico generato dalla normale navigazione internet, come mostrato in:[R1], [R2

], [R3], [R4].

2.2. Parametri in gioco per l'identificazione

Per selezionare un piccolo numero di caratteristiche che possono esser d'aiuto nel classificare i flussi Skype, consideriamo i principali aspetti nei quali il traffico Skype si differenzia dal tipico traffico Web.

Data la natura della comunicazione in real-time, un client Skype genera una bassa bit-rate che dura parecchie decine di secondi ed è composto di parecchi piccoli messaggi, la dimensione dei quali dipende dal Codec rate e dal tempo di framing ΔT.

Rifacendosi all'articolo [R2] “Revealing Skype Traffic:When Randomness Plays with You” di Dario Bonfiglio, e suoi collaboratori, si vuole selezionare come caratteristiche:

- la dimensione del messaggio, che corrisponde alla lunghezza del messaggio incapsulato nel segmento a livello di transport.

  In particolare consideriamo una finestra di w messaggi e per ciascuno di loro controlliamo la dimensione del messaggio.

  Perciò abbiamo:

  $x = [s1 , s2 , . . . , sw]$

dove s-i è la dimensione del messaggio dell'i-esimo pacchetto di una finestra di w consecutivi pacchetti (la dimensione varia a seconda dei Codec con cui abbiamo a che fare).

- l' average-Inter Packet Gap (average-IPG), valutato come

1/w volte il tempo intercorso tra la ricezione del primo e l' w-esimo pacchetto in una finestra.

In questo caso abbiamo un classificatore singolo che non dipende dal Codec impiegato:

$y = [\tau\ ] = [(tw - t1\ )/w]$

Definiamo la certezza della dimensione del messaggio e delle caratteristiche dell'average-IPG, rispettivamente come Bs e Bτ, in termini di somme di logaritmi invece di prodotti così da limitare la cancellazione numerica.

Sempre attingendo a tale articolo, si sono prese in considerazione le seguenti formule:

$$B_s(C) = \frac{1}{w}\sum_{i=1}^{w} \log P\{s_i|C\}$$

$$B_\tau(C) = \log P\{\tau|C\}.$$

L' average-IPG, nel modo in cui è definito, è indipendente dal punto specifico della rete nel quale lo strumento di monitoraggio è fissato.

Per la scelta di w, tempo di framing, è stato verificato, secondo quanto riportato nell'articolo [R2] “Revealing Skype Traffic:When Randomness Plays with You” di

Dario Bonfiglio, che l'impatto del suo valore diventa trascurabile a mano a mano che ci si allontana da 10 (settare w a 30 corrisponde ad un tempo di window di 1s.).

Si considera dapprima l'aspetto della dimensione del messaggio.

Tale parametro è caratterizzato da una una distribuzione per ogni possibile Codec e noi rappresentiamo la dimensione del messaggio per un dato punto di lavoro per mezzo di

una distribuzione Gaussiana propriamente adattata alle misure ottenute nelle nostre analisi.

Un punto di lavoro corrisponde a diverse opzioni dei seguenti parametri:

- rate,

- lunghezza dell'header H,

- fattore di ridondanza RF ,

- messaggio framing time ΔT[1].

Per ciascuna delle combinazioni dei valori {Rate, H, RF, ΔT }, la distribuzione della dimensione del messaggio è rappresentata da una distribuzione Gaussiana, N (μ, σ), con valor medio fissato a: $\mu = (Rate\Delta T + len(H))RF + len(SoM)$.

Per quel che concerne l'average-IPG essa è modellata come distribuzione Gaussiana con $\mu = \Delta T$ e $\sigma = 1$, con nessuna distinzione specifica dei Codec.

2.3. Condizione relativa all' identificazione di un flusso Skype.

Per decidere se il flusso sia generato o meno da Skype, la dimensione del messaggio e i classificatori dell'average-IPG vengono combinati attraverso la valutazione del minimo delle certezze precedentemente derivate,

$B = \min (M axBs , E[B\tau])$

e comparandolo ad una soglia Bmin.

Se $|B| > |Bmin|$ il flusso è classificato come flusso vocale Skype.

1 Quando la sorgente trasmette un frame , fa anche partire un timer, Il timer viene impostato in modo da scattare dopo un intervallo abbastanza lungo (tempo di framing),da permettere che il frame giunga alla destinazione, vi venga elaborato, e possa tornare indietro alla sorgente(Reti di calcolatori, Andrew S. Tanenbaum).

Capitolo 3 Identificazione de traffico

3.1. Wireshark.

Nel 1998 Gerald Combs (un laureato in scienze informatiche dell'Università di Missouri-Kansas city) scrisse e rilasciò sotto licenza GPL (General Public License) il codice di un analizzatore di pacchetti, Ethereal.

Combs alla fine fu costretto a continuare lo sviluppo di Ethereal, realizzando un fork (divisione in due rami distinti dello sviluppo di un progetto software) del progetto originale sotto un nuovo nome, Wireshark. La squadra di sviluppo centrale di Ethereal si è riunita attorno a Combs, per mantenere e continuare lo sviluppo di Wireshark, partendo dal punto in cui c'è stata la disputa legale, mentre lo sviluppo originale di Ethereal è stato praticamente abbandonato. Attualmente sono circa 500 gli sviluppatori che, in modo attivo, contribuiscono al suo miglioramento e lo mantengono. Gerald Combs continua a gestire il codice complessivo e rilascia nuove versioni stabili. Wireshark rimarrà liberamente disponibile come un software open source sotto GNU General Public License. Al momento in cui si scrive, la versione rilasciata è la 0.99.6a, con aggiornamenti di una versione stabile ogni 4-6 settimane circa. Wireshark decodifica circa 850 protocolli o tipi di pacchetti. Viene supportato da diversi sistemi operativi: Windows, Unix e Unix-compatibili, incluso Linux, Solaris, FreeBSD, NetBSD, OpenBSD, MAC OS X. Wireshark è universalmente riconosciuto ed accettato come la "vera" versione di Ethereal.

Wireshark è un software per analisi di protocollo, o packet sniffer (letteralmente *annusa-pacchetti*).

Esso permette all'utente di osservare tutto il traffico presente sulla rete utilizzando la modalità promiscua (modalità che permette di catturare tutti i pacchetti circolanti sul media, non solo quelli che si ottengono comunicando col proprio Access Point) dell'adattatore di rete. Wireshark riesce a "comprendere" la struttura di diversi protocolli di rete, è in grado di individuare eventuali incapsulamenti, riconosce i singoli campi e

permette di interpretarne il significato. Per la cattura dei pacchetti, Wireshark non dispone di proprio codice, ma utilizza libpcap/WinPcap, quindi può funzionare solo su reti supportate da quest'ultime. È possibile analizzare dati acquisiti in tempo reale su una rete attiva ("from the wire"), come pure analizzare dati salvati precedentemente su file di cattura.

La potenza e la versatilita' di questo strumento sono state da noi sfruttate per effettuare la cattura di flussi di dati che abbiamo quindi sottoposto ad analisi.

3.2. Identificazione ed analisi del traffico di rete.

Consideriamo una chiamata tramite Skype e soffermiamoci ad osservare il flusso di pacchetti:

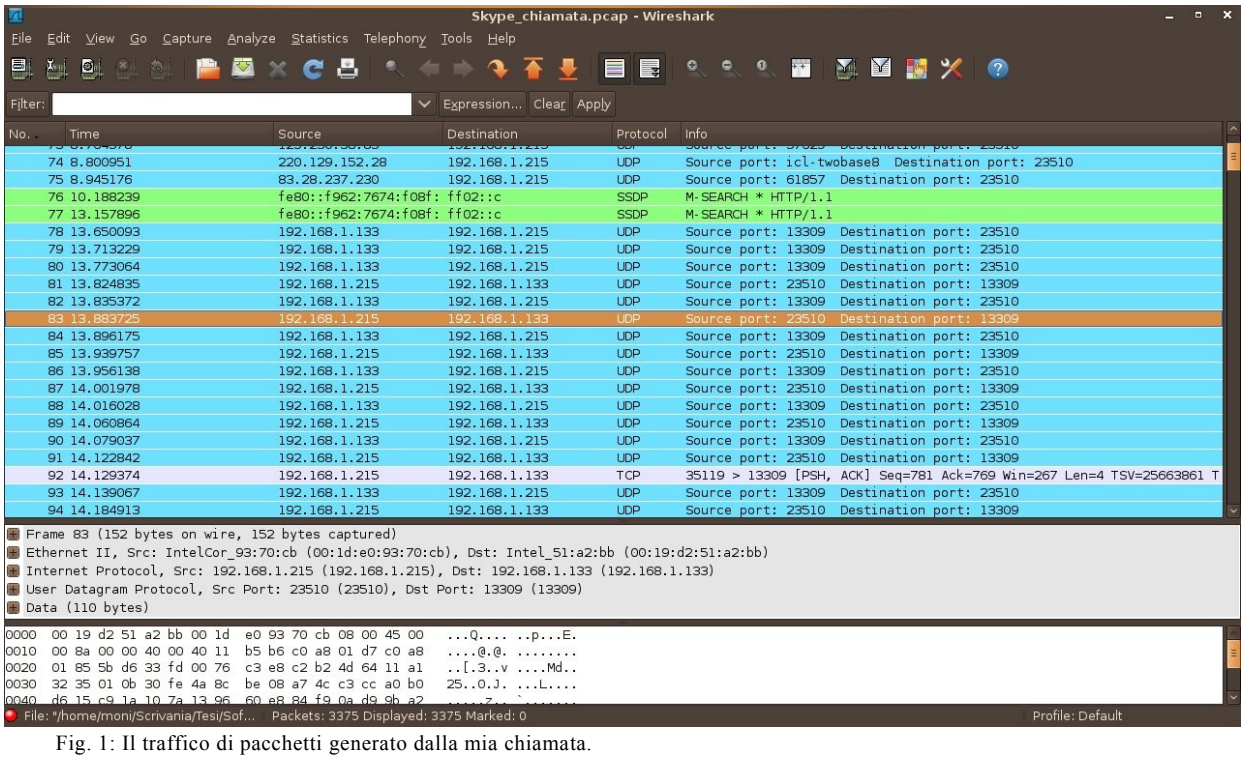


Fig. 1: Il traffico di pacchetti generato dalla mia chiamata.

Ho salvato tale flusso in un file Skype_chiamata.pcap.

Ho creato un programma che mi legge da Dos il file, formato pcap, (con l'ipotesi di analizzare un solo file alla volta) al quale viene associato un descrittore.

Dunque, mi sono preoccupata prima di tutto di includere la libreria “pcap.h”(http://www.tcpdump.org/).

Essendo la chiamata stabilita tra due end-point, nel nostro esempio, tra l'host "192.168.1.133" e l'host "192.168.1.215" ho dichiarato provenienza e destinazione della comunicazione di nostro interesse in una libreria apposita, “ parametri.h” :

#define IP_PC1 "192.168.1.133"

#define IP_PC2 "192.168.1.215"

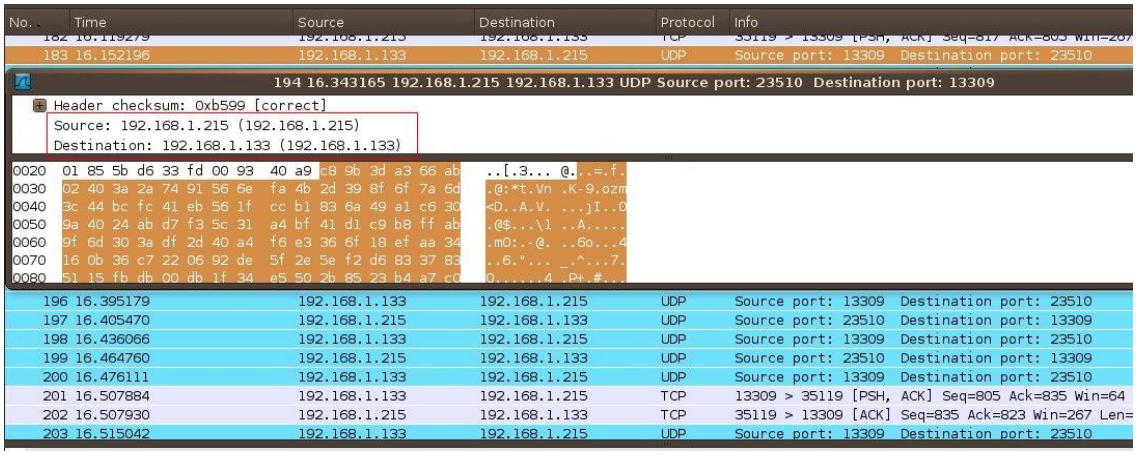


Fig.2: Risulta selezionato un pacchaetto, di cui ho messo in evidenza la source e la destination.

Voglio sottolineare che le nostre analisi fanno riferimento a traffico preso in esame offline, e non vengono effettuate in real-time.

Ho sfruttato la funzione “pcap_open_offline” che mi restituisce un puntatore al nuovo descrittore del pacchetto catturato:

```
pcap_t *descrittore;

char buferr[PCAP_ERRBUF_SIZE];

if(argc<2){

printf("Errore - Parametri sbagliati\n");
```

```
 return -1;
}
descrittore=pcap_open_offline(argv[1],buferr);
if(descrittore==NULL)
{
        printf("Errore.\n %s",buferr);
        return (2);
}
```

Ho analizzato tale file attraverso la funzione “pcap_loop” che mi permette di far eseguire, per ciascuno dei pacchetti, una funzione che ho definito “gotpack”.

```
pcap_loop(descrittore,-1,gotpack,NULL);
```

Per comodità ho voluto analizzare una quantità fissa di pacchetti, perciò ho sostituito al secondo argomento di tale funzione(ovvero al valore“-1”) il numero totale di frame presi in considerazione.

Questo lo ho definito nella mia libreria “parametri.h” sotto la variabile ”NPACK_ANALIZZARE”.

Tale funzione mi stampa a video i parametri di mio interesse di ogni unità di dati, ovvero:

- header del pacchetto,
- indirizzo Mac di provenienza,
- indirizzo Mac di destinazione,
- tipo Ethernet (valore fisso per lo standard: 0x0800),
- indirizzo IP di provenienza,
- indirizzo IP di destinazione,

– Time to Live .

Come si vede in figura:

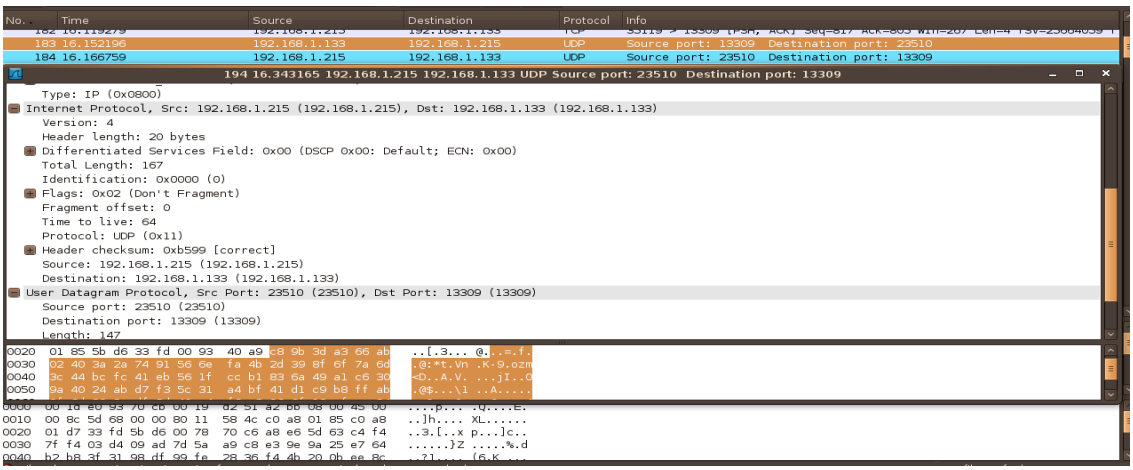


Fig.3: Ho messo in evidenza i vari parametri relativi al pacchetto, di cui abbiamo accennato sopra.

Inoltre si occupa di determinare il tipo di protocollo.

Nel caso si tratti di TCP o UDP viene stampato a video il valore della porta di destinazione e quello di provenienza, per poi proseguire, mentre avviene un semplice return per quel che non risulta di nostro interesse.

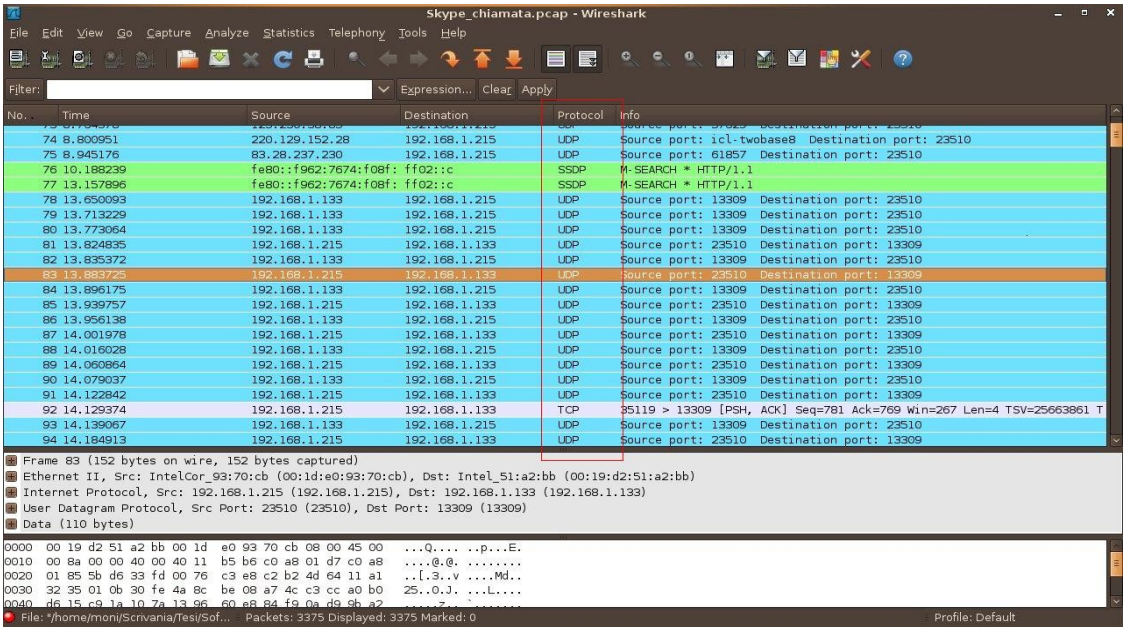


Fig. 4: Ho selezionato in rosso il tipo di protocollo riferito a ciascun pacchetto.

/* determina di che protocollo si tratta */

```
switch(ip->ip_p) {
        case IPPROTO_TCP:
                printf("\tProtocollo: TCP\n");
                tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
                printf("\tSPORT: %d\n", ntohs(tcp->th_sport));
                printf("\tDPORT: %d\n", ntohs(tcp->th_dport));
                return;

        case IPPROTO_UDP:
                printf("\tProtocollo: UDP\n");
                udp = (struct sniff_udp*)(packet + SIZE_ETHERNET + size_ip);
                printf("\tSPORT: %d\n", ntohs(udp->uh_sport));
                printf("\tDPORT: %d\n", ntohs(udp->uh_dport));
                break;

        case IPPROTO_ICMP:
                printf("\tProtocollo: ICMP\n");
                return;
        case IPPROTO_IP:
                printf("\tProtocollo: IP\n");
                return;
        default:
                printf("\tProtocollo: unknown\n");
                return;
}
```

Quindi, controllo che gli indirizzi IP di provenienza e destinazione dei pacchetti corrispondano a quelli di cui sto analizzando la comunicazione:

```
if(
        (strcmp(inet_ntoa(ip->ip_src),IP_PC1)==0) ||
        (strcmp(inet_ntoa(ip->ip_src),IP_PC2)==0) ||
        (strcmp(inet_ntoa(ip->ip_dst),IP_PC1)==0) ||
        (strcmp(inet_ntoa(ip->ip_dst),IP_PC2)==0)) {…
```

All'interno di questa condizione provvedo a passare le mie chiavi caratteristiche, ovvero i due parametri determinanti per le conclusioni relative al tipo di traffico.

Questi vengono mandati come argomento della funzione “updateStats” e mi rimangono memorizzati in un array.

Al termine del loop, ovvero una volta analizzati tutti i pacchetti, stampiamo a video il timestamp e il size message di ciascuno (“stat_size” è la lunghezza dell'array in cui abbiamo salvato i parametri).

```
for(i=0;i<stat_size;i++)
{
printf("size: %04d timestamp: %d:%06d\n", (int)feat[i].sizePack,
(int)feat[i].timeStamp.tv_sec, (int)feat[i].timeStamp.tv_usec);
}
```

Attraverso le seguenti funzioni:

- funzprsize();
- funzprtimestamp();
- min(Bs,Bt);

ci occupiamo di calcolare la certezza della dimensione del messaggio e delle caratteristiche dell'average-IPG, rispettivamente Bs e Bτ, e prendiamo in considerazione il valor minimo tra questi due.

Il valore ottenuto sarà proprio B.

Ho stabilito una soglia, Bmin, a cui ho assegnato il valore ” 5.5 ”, basandomi su prove statistiche, ovvero testando i valori dei parametri relativi al traffico proveniente da una chiamata Voip, da un discorso in chat o da una videochiamata su Skype, e confrontandoli con quelli relativi ad una generica ricerca su Google, o ad un video su Youtube, ad esempio.

Rifacendomi alla condizione di cui abbiamo trattato nel capitolo 1 , identifico il tipo di traffico con cui ho a che fare:

```
if(B<Bmin){
printf("Il flusso analizzato e' traffico proveniente da Skype.\n");
}
else{
printf("Il flusso non e' dovuto a Voip.\n");
}
```

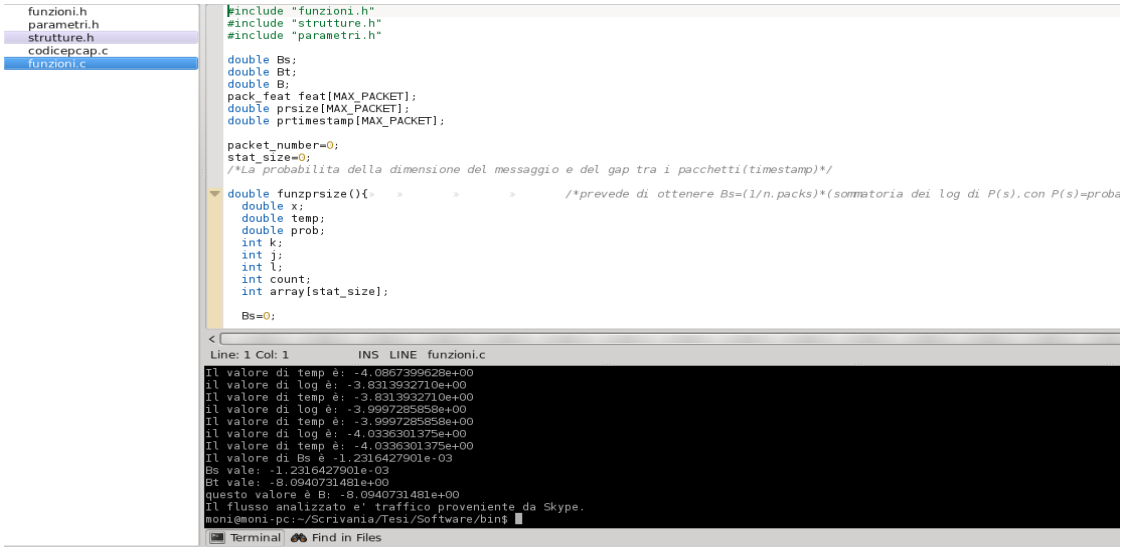


Fig. 5: Il riscontro finale del programma che effettua l'identificazione e il riconoscimento dei pacchetti.

3.3. I modelli per il riconoscimento dei flussi VoIP

Per il riconoscimento del traffico mi sono occupata di prender in analisi il comportamento generato dalla normale navigazione internet.

Quindi, tramite Wireshark, ho analizzato i flussi generati da una navigazione comune su un motore di ricerca, Google, ho fatto accesso ad un suo link, scaricando un file in pdf.

Ho preso ulteriormente in esame il traffico proveniente dalla visione di un video sul sito www.youtube.com.

Inoltre, mi sono interessata ad analizzare il flusso dovuto allo scaricamento di file multimediali di formato mp3 da un Peer-to-Peer[2], LimeWire. In particolare viene osservato il download di due brani in due tempistiche differenti, viene messo in pausa il primo file al fine di lasciar attivo il download del secondo.

Un'altra fonte di generazione di traffico che mi sono occupata di considerare è la Posta elettronica.

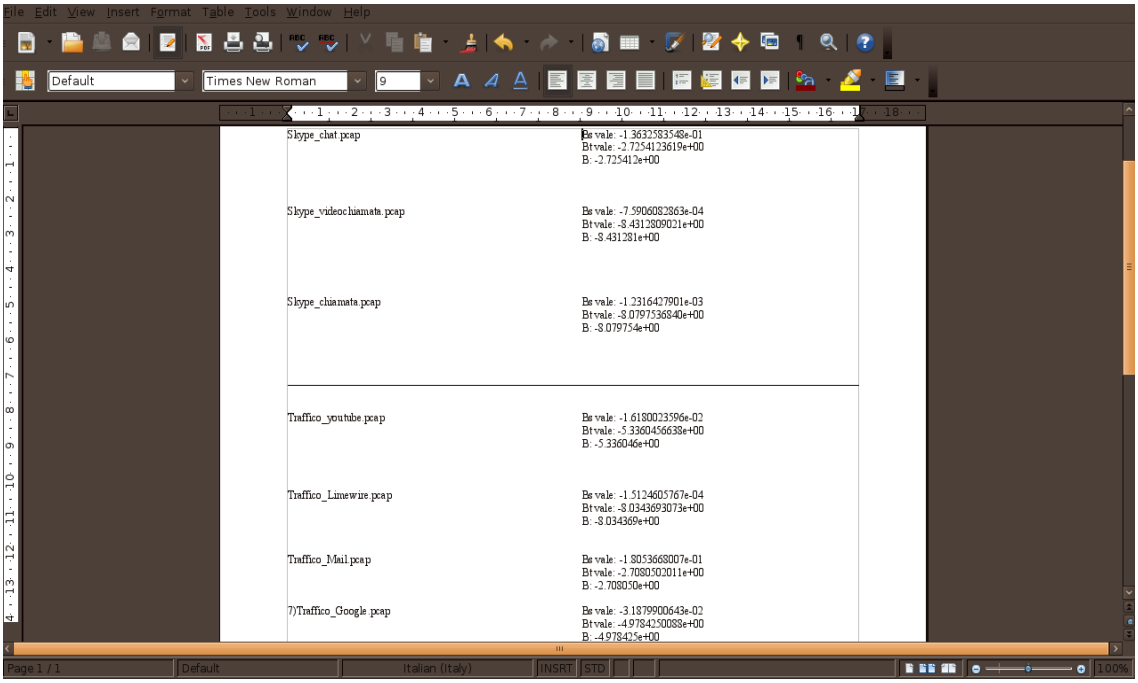
Tali analsi le ho messe a confronto con quelle ottenute dai flussi Skype: mi sono occupata di memorizzare i pacchetti che viaggiono durante una semplice conversazione in chat che prevedeva esclusivamente uno scambio di informazioni e di file.

Inoltre, ho preso in considerazione una comune chiamata voce tra due host (due PC)

e, allo stesso modo, ho analizzato una videochiamata.

Fig. 6: I risultati riscontrati dai modelli presi in esame, ipotizzando di analizzare 5000 pacchetti.

2 Peer-to-peer è una rete di computer o qualsiasi rete informatica che non possiede nodi gerarchizzati come client o server fissi (clienti e serventi), ma un numero di *nodi equivalenti* (in inglese *peer*) che fungono sia da cliente che da servente verso altri nodi della rete(Wikipedia,http://it.wikipedia.org/wiki/Peer-to-peer).



Nella valutazione di questi modelli ho riscontrato un valore soglia approssimativo, Bmin, definito nell'articolo di Bonfiglio, relativamente al quale sono state tratte le conclusioni sul riconoscimento del traffico VoIP.

3.4. Il Makefile

Mi sono occupata di creare un Makefile, ovvero un file che guida l'utente alla compilazione esplicitando quali file compilare, quali moduli linkare e che file binari generare in relazione a macro opportunamente definite.

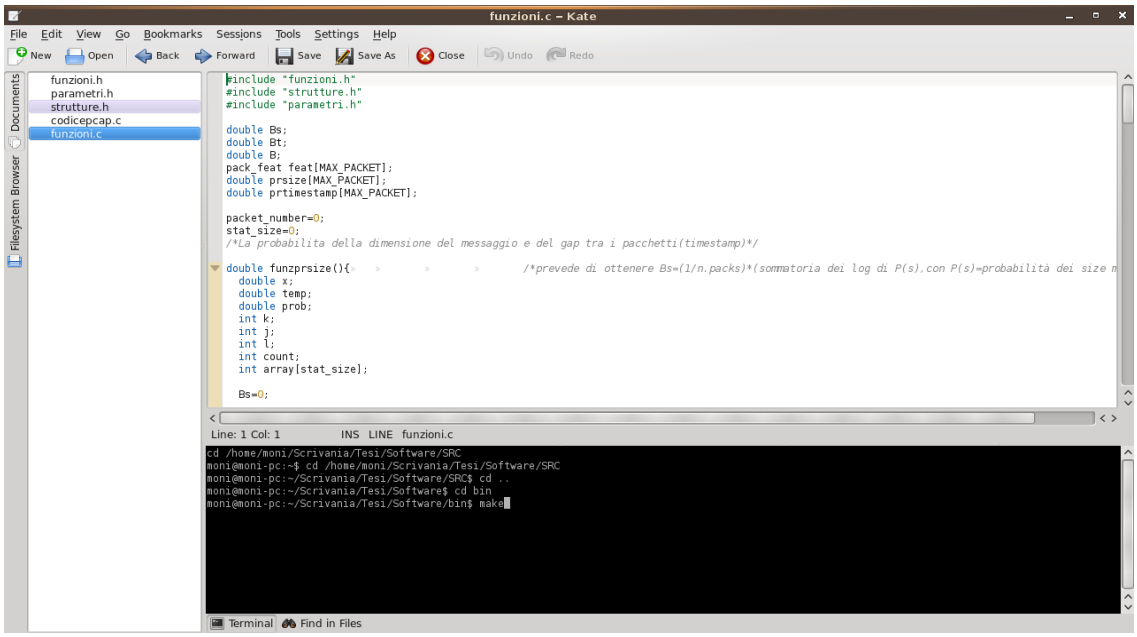


Fig. 7: Il makefile viene richiamato digitando il comando “make” da shell.

Il makefile da me creato è descritto di seguito:

Se è previsto un debug, allora verrà creato un file exe_d.bin, altrimenti un file exe.bin:

_ _ _

ifeq ($(DEBUG_VERSION),yes)

OUT_FILE = exe_d.bin

else

OUT_FILE = exe.bin

endif

_ _ _

Le seguenti linee definiscono qual è il compilatore da usare , l'operazione di copia e di rimozione, rispettivamente:

– – –

CC = gcc

CP = cp

RM = rm -f

– – –

In questo modo la variabile CFLAGS viene istanziata con il contenuto della variabile INCLUDES e della variabile DLINUX. Fondamentalmente la variabile CFLAGS è l'unione di queste.

– – –

CFLAGS = -c $(INCLUDES) -DLINUX -Wall

– – –

Wall: mi farà comparire più Warning.

Il Makefile risulta particolarmente utile nei progetti di dimensione estesa che presentano una gran quantità di moduli, permettendo di non ricompilare tutto il codice ad ogni modifica.

Capitolo 4 Mascheramento del traffico

4.1. Approccio al mascheramento.

Ci si è posti l'obiettivo di mascherare le comunicazioni Voip, così da far apparire tale traffico di natura diversa. A questo scopo abbiamo nuovamente preso in esame i modelli trattati nel capitolo precendente.

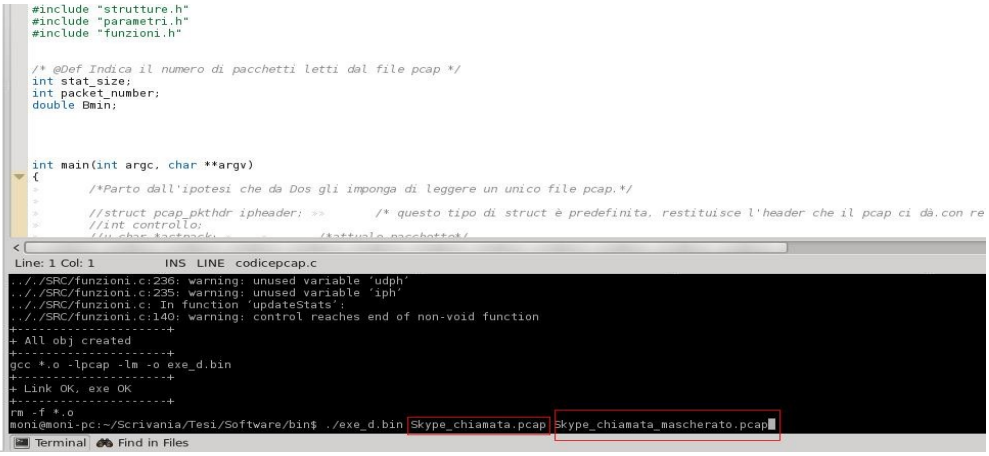
Ciò è stato possibile agendo sui parametri “Bs” e “Bt” suggeriti dall'articolo “Revealing Skype Traffic:When Randomness Plays with You”, di Dario Bonfiglio [R2].

A tale scopo, ho quindi creato un nuovo programma facilmente integrabile al primo.

Attraverso di esso, da Shell si indica come primo argomento il file di formato pcap, del quale si vuole effettuare la modifica.

Come secondo argomento, si definisce il nome del nuovo file, risultato del mascheramento del precedente.

Fig. 8: Sono qui illustrati in sequenza i due argomenti di cui abbiamo accennato sopra.



Quindi otteniamo un file sempre di formato pcap, che presenta al suo interno il gruppo di pacchetti orginali intervallati a pacchetti fittizi.

Quest'ultimi sono stati creati da me, e possono variare nel numero. Ciò è possibile grazie a una semplice variabile che mi sono occupata di definire in “parametri.h”

#define GRUPPO_PAC_FITTIZI seguita dal numero.

La loro struttura è chiaramente identica a quella dei pacchetti originali, con la differenza che in questi ho modificato i parametri di “Total length” per l'IP, “length” per l'UDP e il “Time delta from previous captured frame”, determinanti ai fini del mascheramento, coerentemente a ciò che si è detto nell'articolo di Bonfiglio [R2].

Nonostante ciò è mio dovere precisare che questa modifica è stata fatta tramite un cambiamento forzato dei valori della lunghezza associata all'IP e all'UDP, del quale il CRC (metodo per il calcolo di somme di controllo utile per l'individuazione degli errori casuali) correttamente si accorge.

Ecco un esempio del file prima del mascheramento e dopo:

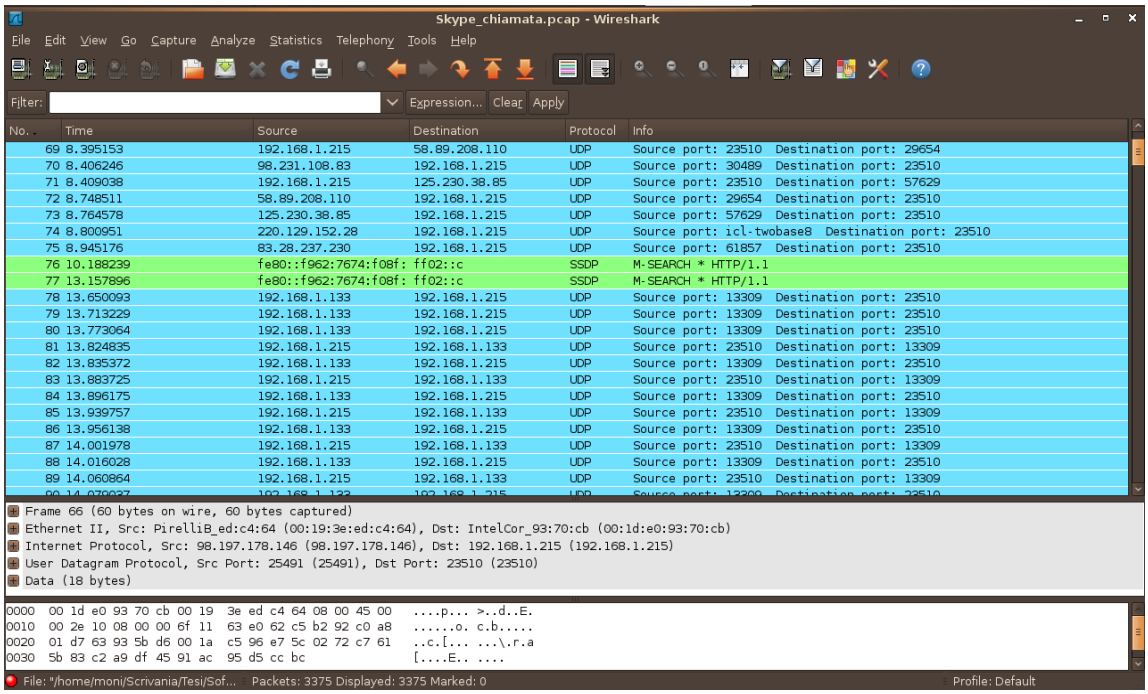


Fig. 9: Prima del mascheramento: questi sono i pacchetti del file Skype_chiamata.pcap.

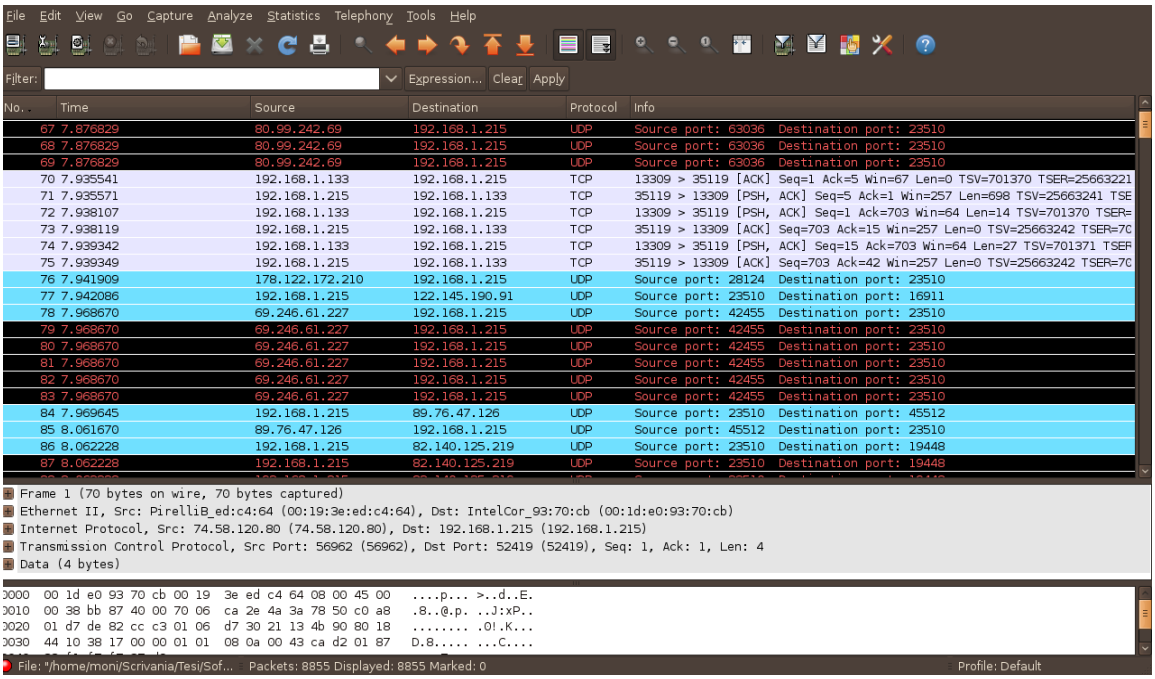


Fig. 10: Dopo il mascheramento: in rosso sono evidenziati i pacchetti fittizi del file Skype_chiamata_mascherato.pcap.

Perchè il CRC non si accorgesse dei cambiamenti dei parametri, avrei potuto effettuare una modifica diretta sui pacchetti stessi, ad esempio, aggiungendo degli zeri per aumentare la lunghezza del frame o ancora meglio, avrei potuto semplicemente ricalcolare il nuovo CRC e inserirlo correttamente nel pacchetto fittizio.

Il principio secondo cui è stato scelto di stabilire una determinata quantità dei pacchetti e lunghezza dei frame, si fonda su una strategia empirica, ovvero testando i modelli di cui abbiamo trattato nel capitolo “Identificazione del traffico”.

Il meccanismo alla base della creazione dei pacchetti fittizi, risulta esser sistematico in modo che, se l'analisi venisse effettuata online, risulterebbe possibile, una volta arrivato il file al ricevente, sfruttare un meccanismo inverso e recuperare i pacchetti originali integralmente.

Alternativamente, sarebbe possibile "marcare" i pacchetti fittizi con un semplice pattern permettendone l'identificazione e la rimozione dal lato ricevente.

Conclusioni

Il lavoro si è svolto partendo da un' analisi dei modelli statistici del traffico di rete, dedicandosi alla lettura di algoritmi trattati negli articoli [R1], [R2], [R3].

In particolare, vista la maggior attendibilità, ho preso in considerazione l'algoritmo Erebos, un classificatore basato sul principio di Bayes presente in “Revealing Skype Traffic:When Randomness Plays with You” [R2].

Ho analizzato flussi di pacchetti di vario tipo tramite il software Wireshark, e, in riferimento ai parametri definiti nell'articolo [R2], ovvero alla lunghezza dei pacchetti e al timestamp, ho stabilito su base empirica un valore soglia per distinguire il traffico proveniente da una conversazione Skype da quello di natura diversa.

Il valore di soglia riscontrato è stato: -8.03437, definito come “Bmin”.

Ho creato dunque un programma capace di identificare se il traffico in esame provenisse da Skype o meno.

Allo scopo di testare se tale soluzione fosse vulnerabile ad un eventuale mascheramento del traffico Voip, ho compiuto modifiche al flusso di pacchetti iniziale, tramite un ulteriore programma da me creato, in cui mi sono occupata di aggiungere pacchetti fittizi intervallati a pacchetti reali.

I pacchetti fittizi sono stati modificati sotto il campo “Total length” relativamente all'IP, sotto “length” relativamente all'UDP e in “Time delta from previous captured frame”.

Il risultato ottenuto è stato significativo: nel caso ideale di un'aggiunta di una gran quantità di pacchetti fittizi tra due pacchetti reali consecutivi, è stata provata un' effettiva vulnerabilità.

Ciò è risultato dall'analisi di diversi flussi. Ad esempio, si è considerato un file di formato pcap che raccogliesse al suo interno pacchetti provenienti da una videochiamata Skype.

Il programma da me creato, dedito all'identificazione, ha restituito come valore “B”: -8.431281e+00.

Quindi, conformemente alle attese, tale software ha riconosciuto la natura del traffico quale VoIP.

Una volta effettuato il mascheramento, il nuovo valore di “B” è divenuto: -7.730997e+00.

Quindi, non rivelatasi rispettata la condizione |B| > |Bmin|, il traffico, dopo le modifiche apportate, non è risultato più traffico proveniente da Skype.

In una situazione reale, non può esser applicata una soluzione di questo tipo per l'occupazione eccessiva di banda.

Un possibile approccio potrebbe essere aggiungere al massimo due o tre pacchetti fittizi nel “gap” presente tra due pacchetti reali consecutivi, prendendosi cura di inserirli con alternanza, a metà del gap, prima della metà e dopo la metà.

Appendice A

Makefile

```
#
# Author: M. Curti
#
# Makefile
#*********************************************
#*   PARAMETRI                               *
#*********************************************
# versione debug: yes/no
DEBUG_VERSION = yes
#*********************************************
#*   DIRECTORY/FILE                          *
#*********************************************
WORK_DIR      = .
PROJECT_DIR   = ../$(WORK_DIR)
INC           = $(PROJECT_DIR)/INC
SRC           = $(PROJECT_DIR)/SRC
DEST_DIR      = $(PROJECT_DIR)/bin
# se è previsto il debug verrà creato il file nomefile_d.bin:
ifeq ($(DEBUG_VERSION),yes)
OUT_FILE      = exe_d.bin
```

```
else
OUT_FILE      = exe.bin
endif
#*********************************************
#*   COMANDI                                 *
#*********************************************
#compilatore
CC      = gcc
#linker: inutile, uso gcc anche x linkare
LD      = ld
#copia
CP      = cp
#rimozione
RM      = rm -f
#*********************************************
#*   PATH DI RICERCA PER INCLUDE FILES       *
#*********************************************
INCLUDES = -I$(INC) \
#*********************************************
#*   OPZIONI COMPILATORE E LINKER            *
#*********************************************
#### opzioni di compilazione per versione release (impianto):
#-c Compile or assemble the source files, but do not link. The linking stage simply is not done.
#   The ultimate output is in the form of an object file for each source file.
```

```
#-Wall generate warnings for everything0
#-DLINUX so Linux
# -m32/m64 indica per qual architettura compilare (32 o 64 bit per sizeof int)
CFLAGS = -c $(INCLUDES) -DLINUX -Wall
#### opzioni di compilazione aggiuntive per versione debug o simulato:
#-g Produce debugging information
#-ggdb Produce debugging information for use by GDB
#-DDEBUG trace info di debug
#per debug:
ifeq ($(DEBUG_VERSION),yes)
CFLAGS += -g -ggdb -DDEBUG
endif
#### opzioni generali di linker:
# -lpcap link alla libreria pcap
LDFLAGS = -lpcap -lm
#*********************************************
#*   ENTRY POINT DEL MAKEFILE                *
#*********************************************
all: exe
#*********************************************
#*   TARGET DEL MAKEFILE                     *
#*********************************************
clean:
        $(RM) $(DEST_DIR)/$(OUT_FILE)
obj:
```

```
	@echo
+-------------------------------------------------------------------------------------------------------------
--------------+
	@echo
+-------------------------------------------------------------------------------------------------------------
--------------+
	$(CC) $(CFLAGS) $(SRC)/*.c
	@echo +----------------------+
	@echo +  All obj created
	@echo +----------------------+
exe: obj
	$(CC) *.o $(LDFLAGS) -o $(OUT_FILE)
	@echo +----------------------+
	@echo +  Link OK, exe OK
	@echo +----------------------+
#	$(CP) $(OUT_FILE) $(DEST_DIR)
	$(RM) *.o
#	$(RM) $(OUT_FILE)
```

Appendice B

Funzioni.h

```
#ifndef FUN_H
#define FUN_H

#include "parametri.h"
#include "strutture.h"
#include <pcap.h>

extern int j;
extern int fittizi;
extern int counter;
extern double Bs;
extern double Bt;
extern double B;
extern double Bmin;
extern pack_feat feat[MAX_PACKET];


extern void gotpack(u_char *args, const struct pcap_pkthdr *header, const u_char
*packet);
extern double funzprsize();
extern double funzprtimestamp();
```

```
extern double min(double Bsc, double Btc);
extern int updateStats(u_int size_m, struct timeval* timestamp);



/* MASKING */


extern void work(u_char *args, struct pcap_pkthdr *header, const u_char *packet);
extern unsigned short csum (unsigned short *buf, int nwords);


#endif
```

Appendice C

Parametri.h

```
#ifndef PAR_H
#define PAR_H

#define IP_PC1 "192.168.1.133"
#define IP_PC2 "192.168.1.215"

#define MAX_PACKET 20000000

#define SOGLIABMIN -8.03437
#define ARBITRARY_SPORT 13309
#define ARBITRARY_DPORT 23510

#define NPACK_ANALIZZARE 5000
#define MAX_PACKET_SIZE 4096
#define GRUPPO_PAC_FITTIZI 700
#define FREQUENZA 1 //ogni tot:"FREQUENZA" pacchetti originali ne metto di fittizi : "GRUPPO_PAC_FITTIZI"
#define COSTANTE 64
#endif
```

Appendice D

Strutture.h

```
#ifndef STRUTT_H
#define STRUTT_H

#include <sys/time.h>
#include <arpa/inet.h>

/* default snap length (maximum bytes per packet to capture) */
#define SNAP_LEN 1518

/* ethernet headers are always exactly 14 bytes [1] */
#define SIZE_ETHERNET 14

/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN	6

/* Ethernet header */
typedef struct{
	u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address */
	u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
	u_short ether_type;                     /* IP? ARP? RARP? etc */
```

```
}sniff_ethernet;

/* IP header */
typedef struct{
        u_char  ip_vhl;                 /* version << 4 | header length >> 2 */
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
        u_short ip_off;                 /* fragment offset field */
        #define IP_RF 0x8000            /* reserved fragment flag */
        #define IP_DF 0x4000            /* dont fragment flag */
        #define IP_MF 0x2000            /* more fragments flag */
        #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
}sniff_ip;

#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* TCP header */
typedef u_int tcp_seq;
```

```
typedef struct{
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
        #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|
TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
}sniff_tcp;


typedef struct{
        u_short uh_sport;               /* source port */
```

```
        u_short uh_dport;               /* destination port */
                        u_short uh_length;              /* urgent pointer */
        u_short uh_sum;                 /* checksum */
}sniff_udp;

typedef struct{
  u_int sizePack;
  struct timeval timeStamp;
}pack_feat;

/*typedef struct{
            u_short lenip;
            struct in_addr ip_source,ip_dest;
}sniffatoip;

typedef struct {
            u_short sourceport;
            u_short destport;

}sniffatotcp;*/

#endif
```

Appendice E

Codicepcap.c

Analisi ed Identificazione

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>                    /*mi serve per correggere qst errore codicepcap.c:11: error: field ‘ip_source’ has incomplete type*/
#include <math.h>
#include <sys/time.h>                     /* Per timeval */
#include "strutture.h"
#include "parametri.h"
#include "funzioni.h"



/* @Def Indica il numero di pacchetti letti dal file pcap */
int stat_size;
int packet_number;
double Bmin;



int main(int argc, char **argv)
```

```
{
        /*Parto dall'ipotesi che da Dos gli imponga di leggere un unico file pcap.*/

        //struct pcap_pkthdr ipheader;                  /* questo tipo di struct e' predefinita, restituisce l'header che il pcap ci dà.con relative info sull header come lunghezza,timestamp.. ecc */
        //int controllo;
        //u_char *actpack;              /*attuale pacchetto*/

        pcap_t *descrittore;    /*descrittore associato al file pcap*/
        char buferr[PCAP_ERRBUF_SIZE];
        //controllo=1;

        /* controllo che i parametri siano almeno due, ovvero file stesso e almeno un file pcap*/
        if(argc<2){
          printf("Errore - Parametri sbagliati\n");
          return -1;
        }
        /*La funzione pcap_open_offline restituisce un puntatore al nuovo descrittore del pacchetto catturato presente nel file pcap, passato con argv[1]*/
        descrittore=pcap_open_offline(argv[1],buferr);
        if(descrittore==NULL)
        {
                printf("Errore.\n %s",buferr);
                //controllo=1;
```

```
        return -1;
    }

    /*
    while(controllo==1)   //alloco variabile, mando in output dest source,
portasource portadest
    {
        actpack=pcap_next(descrittore,&ipheader);
        if(actpack == NULL)
          controllo=0;

        printf("La lunghezza dell'header e' la seguente: %d \n",ipheader.len);

        //pcap_loop(descrittore,-1,gotpack,NULL);
        //in un esempio che ho visto c'era qst, ma questo cast non
    //cambia il tipo, cosa fa??
        //u_char *pkt_ptr = (u_char *)actpack; //cast a pointer to
    the packet data
        countpack++;
    }
    */
    /* cattura più pacchetti e per ognuno catturato esegue la funzione di gestione
gotpack*/
    pcap_loop(descrittore,NPACK_ANALIZZARE,gotpack,NULL);
    /* ho messo -1 come secondo parametro per non dargli un numero predefinito di
pacchetti da prelevare, cosi preleva tutti quelli che ci sono.*/
```

```
        /* i quarto parametro specifica la lunghezza dell header dei pacchetti catturati.*/
        printf("Il numero di pacchetti: %d \n",packet_number);
        pcap_close(descrittore); //chiude il file pcap

        /* lancio analisi statistica sui dati raccolti */
        /* executeAnalisys()*/

        Bmin=SOGLIABMIN;
/*Per ogni pacchetto,quindi per ogni indice dell'array "feat" mi faccio stampare a video i rispettivi parametri*/
        int i;
        for(i=0;i<stat_size;i++)
          printf("size: %04e timestamp: %e:%06e\n", (int)feat[i].sizePack, (int)feat[i].timeStamp.tv_sec, (int)feat[i].timeStamp.tv_usec);

        funzprsize();
        funzprtimestamp();
        min(Bs,Bt);


        printf("B: %e \n",B );
        B=-B;
        Bmin=-Bmin;

        if(B > Bmin)
```

```
        {
          printf("Il flusso analizzato e' traffico proveniente da Skype.\n");
        }
        else{
          printf("Bene, tale flusso non e' dovuto a Voip.\n");
        }
        return 0;
}
```

Appendice F

Masking.c

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>                  /*mi serve per correggere qst errore
codicepcap.c:11: error: field ‘ip_source’ has incomplete type*/
#include <math.h>
#include <sys/time.h>                   /* Per timeval */
#include "strutture.h"
#include "parametri.h"
#include "funzioni.h"

/* @Def Indica il numero di pacchetti letti dal file pcap */

int stat_size;
int packet_number;

int main(int argc, char **argv)
{
```

```
/*Parto dall'ipotesi che da Dos gli imponga di leggere un unico file pcap e di scrivere sul file dopo.*/
        pcap_dumper_t *restituito;
        pcap_t *descrittore;    /*descrittore associato al file pcap*/
        char buferr[PCAP_ERRBUF_SIZE];
        //controllo=1;

/* controllo che i parametri siano almeno tre, ovvero file stesso e almeno un file pcap e un altro file pcap in cui risultino i miei pacchetti modificati*/

        if(argc<3){
          printf("Errore - Parametri sbagliati\n");
          return -1;
        }

/*La funzione pcap_open_offline restituisce un puntatore al nuovo descrittore del pacchetto catturato presente nel file pcap, passato con argv[1]*/

        descrittore=pcap_open_offline(argv[1],buferr);

        if(descrittore==NULL)

        {
                printf("Errore1.\n %s",buferr);
                return -1;
```

```
        }
        restituito=pcap_dump_open(descrittore,argv[2]);
        if(restituito==NULL)
        {
                printf("Errore2.\n %s",buferr);
                return -1;
        }
        /* cattura più pacchetti e per ognuno catturato esegue la funzione di gestione
work*/
        printf("Prima del loop\n");
        pcap_loop(descrittore,NPACK_ANALIZZARE,work,(u_char*) restituito);
        //pcap_loop(descrittore,-1,work,NULL);
        //pcap_loop(descrittore,-1,gotpack,NULL);
        printf("counter dei pacchetti reali modificati: %d", counter);
        printf("Fine\n");
        pcap_dump_close(restituito);
        return 0;
}
```

Appendice G

Funzioni.c

```
#include "funzioni.h"
#include "strutture.h"
#include "parametri.h"
#include <stdlib.h>
#include <string.h>
#include <time.h>

double Bs;
double Bt;
double B;
pack_feat feat[MAX_PACKET];
double prsize[MAX_PACKET];
double prtimestamp[MAX_PACKET];

int count_udp=0;
int counter;
counter=0;
int j;

static int last_rand=0;
```

```
packet_number=0;
stat_size=0;
/*La probabilita della dimensione del messaggio e del gap tra i pacchetti(timestamp)*/

double funzprsize(){                    /*prevede di ottenere
Bs=(1/n.packs)*(sommatoria dei log di P(s),con P(s)=probabilità dei size message.)*/
  double x;
  double temp;
  double prob;
  int k;
  int j;
  int l;
  int count;
  int array[stat_size];

  Bs=0;

  for(k=0;k<stat_size;k++){             /*considero un pacchetto alla volta e per
ognuno:*/
    count=0;                            /*inizializzo count a zero*/
    for(j=0;j<stat_size;j++){
      if(feat[k].sizePack==feat[j].sizePack){   /*confronto l'attuale pacchetto con tutti gli
altri e anche con se stesso */                  /*una volta sarà confrontato con se stesso
quindi nel conto già considero quel valore,percio una volta si verifica di certo, quindi
qst if lo fara sempre*/
```

```
      count++;                          /*numero di pacchetti con lo stesso size message*/
    }
  }
    array[k]=count;                     /*ho un array e per ogni pacchetto gli scrivo quanti altri pacchetti ci sono col suo stesso size message*/
    printf("array vale:%d\n",array[k]);
    printf("stat_size:%d\n",stat_size);

    prob= (double)array[k] / stat_size;          /*il numero di volte che ho quel sizemessage / il num totale di sizemessage(ovvero il num di pacchetti)*/
    printf("la prob di qst pacchetto e' %.10e \n",prob);

    prsize[k]=prob;                     /*memorizzo nell'array le probabilità associate ad ogni pacchetto*/

  }
  for(l=0;l<stat_size;l++){
   temp=0;
   x=0;
   x=log(prsize[l]);
   printf("il valore di log e': %.10e\n",x);
   temp=x + temp;                       /*sommatoria dei log delle probabilita*/
   printf("Il valore di temp e': %.10e\n",temp);
  }
  Bs=(double)temp/ stat_size;           /*tale quantità la divido per il numero dei pacchetti e ottengo il parametro Bs*/
```

```
 printf("Il valore di Bs e' %.10e\n", Bs);
 return Bs;
}

double funzprtimestamp(){                    /*prevede di ottenere Bt=log P(t), ove
P(t) e' la probabiltà che si verifichi quel determinato timestamp. t e' definito come
t=((tkesimo pack - t0)/n.packs k-esimi)*/
        printf("Ecco qu non lo legge");
        int k;
        int i;
        int l;
        int count;
        double tau[stat_size];
        double prtime[stat_size];
        double btau[stat_size];
        double sumbtau;
        double var1;
        double var2;
        Bt=0;
        sumbtau=0;

        for(k=1;k<stat_size;k++){
                printf("timestamp del pacchetto: %e\n",
feat[k].timeStamp.tv_sec);
                printf("timestamp del primo pacchetto:
%e\n",feat[0].timeStamp.tv_sec);
```

```
                var1=feat[k].timeStamp.tv_sec;
                var2=feat[0].timeStamp.tv_sec;
                l=k-1;
                tau[l]=(var1 - var2)/ k; //array in cui metto i vari tau per ciascun
pacchetto
                printf("tau per qst pacchetto: %e\n", tau[l]);
        }

        for(i=0;i<stat_size;i++){
                count=0;
                for(l=0;l<stat_size;l++){
                        if(tau[i]==tau[l]){
                                count++;          //conto quanti
hanno                                                   lo
stesso tau,
                considero anche me
                        stesso
                        }
                }
                printf("count: %d\n", count);
                tau[i]=count; //quante volte si verifica qsto tau
                prtime[i]=(double) tau[i]/stat_size;    //probabilita che si verifichi
una certa tau
                printf("probabilita che si verifichi una certa tau: %e\n", prtime[i]);
                btau[i]=log(prtime[i]);          //il suo logaritmo
                printf("risultato di btau per questo pacchetto: %e\n", btau[i]);
```

```
        }
        for(k=0;k<stat_size;k++){
                        sumbtau +=btau[k]; //per far la media prima soommo tutti i
valori.poi dividerò per il numero!
        }
        printf("somma di tutti i tau: %e\n", sumbtau);
        Bt=(double)sumbtau/stat_size;          /*ottengo il parametro Bt metto iil meno
perche nn voglio considerare tale parametro */
        return Bt;
}

double min(double Bsc, double Btc){                    /*Confronta i due parametri e mi
resttuisce quello con valore minimo, salvandolo nella variabile B*/
    B=0;
    printf("Bs vale: %.10e\n",Bsc);
    printf("Bt vale: %.10e\n",Btc);
    if(Bsc<Btc)
        B=Bsc;
    else
        B=Btc;

    return B;
}

/* La funzione assegna i valori di size message e di timestamp per ogni pacchetto
memorizzandole in un array di tipo pack_feat(struttura che ho definito in strutture.h)*/
```

```
int updateStats(u_int size_m, struct timeval* timestamp){
  /*printf("appena dentro la funz updateStats ");*/
 feat[stat_size].sizePack = size_m;
 memcpy(&(feat[stat_size].timeStamp), timestamp, sizeof(timestamp));

 stat_size++;
 /*printf("fa la funz updateStats ");*/
}
/* La funzione  viene effettuata per ogni pacchetto(in quanto richiamata dal pcap_loop) e restituisce per ciascuno i parametri di nostro interesse*/
void gotpack(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
	/*dichiaro puntatori agli header dei pacchetti*/
	sniff_ethernet* eth_header;
	sniff_ip* ip;
	sniff_tcp* tcp;
	sniff_udp* udp;
	int size_ip;

	/*effettuo un cast e pra come ora dico che l'intero pacchetto e' identificato come l'header di ethernet*/
	eth_header = (sniff_ethernet*)packet;

	/*con quest altro cast il puntatore ip sta puntando esattamente all'indirizzo che viene subito dopo la parte di indirizzi dedicata ad ethernet*/
```

```
ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);

size_ip = IP_HL(ip)*4;

printf("\n- Prossimo Pacchetto: %d -\n", ++packet_number);

/* lunghezza dell'header*/
printf("\tFrame Lenght: %d\n", header->len);

/*indirizzo MAC di provenienza e destinazione*/
printf("\tMAC_DESTINATION: %02x:%02x:%02x:%02x:%02x:%02x\n",
(unsigned int)eth_header->ether_dhost[0],(unsigned int)eth_header->ether_dhost[1],
(unsigned int)eth_header->ether_dhost[2],(unsigned int)eth_header->ether_dhost[3],
(unsigned int)eth_header->ether_dhost[4],(unsigned int)eth_header->ether_dhost[5]);
printf("\tMAC_SOURCE: %02x:%02x:%02x:%02x:%02x:%02x\n", (unsigned
int)eth_header->ether_shost[0],(unsigned int)eth_header->ether_shost[1],(unsigned
int)eth_header->ether_shost[2],(unsigned int)eth_header->ether_shost[3],(unsigned
int)eth_header->ether_shost[4],(unsigned int)eth_header->ether_shost[5]);

/* tipo ethernet, e' uno standard, un valore fisso che e' sempre 0x0800.*/
printf("\tType: 0x%04x\n", ntohs(eth_header->ether_type));

/* stampa provenienza e destinazione degli indirizzi IP */
printf("\tFrom: %s\n", inet_ntoa(ip->ip_src));
printf("\tTo: %s\n", inet_ntoa(ip->ip_dst));
// Time To Live
```

```
printf("\tTTL: %d\n", ip->ip_ttl);

/* determina di che protocollo si tratta */
switch(ip->ip_p) {
        case IPPROTO_TCP:
                printf("\tProtocollo: TCP\n");
                tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
                printf("\tSPORT: %d\n", ntohs(tcp->th_sport));
                printf("\tDPORT: %d\n", ntohs(tcp->th_dport));
                return;

        case IPPROTO_UDP:
                printf("\tProtocollo: UDP\n");
                udp = (struct sniff_udp*)(packet + SIZE_ETHERNET + size_ip);
                printf("\tSPORT: %d\n", ntohs(udp->uh_sport));
                printf("\tDPORT: %d\n", ntohs(udp->uh_dport));
                break;

        case IPPROTO_ICMP:
                printf("\tProtocollo: ICMP\n");
                return;
        case IPPROTO_IP:
                printf("\tProtocollo: IP\n");
                return;
        default:
```

```
                printf("\tProtocollo: unknown\n");
                return;
        }
        /* siccome esco dallo switch e' UDP.Quindi confronto che destinatario e fonte siano uno dei due indirizzi IP che comunicano */
        /*printf("Prima di entrar nell'if\n");
        printf("indirizzo di provenienza %s\n",inet_ntoa(ip->ip_src));
        printf("indirizzo di destinazione %s\n",inet_ntoa(ip->ip_dst));
        */
        if(
          (strcmp(inet_ntoa(ip->ip_src),IP_PC1)==0) ||
          (strcmp(inet_ntoa(ip->ip_src),IP_PC2)==0) ||
          (strcmp(inet_ntoa(ip->ip_dst),IP_PC1)==0) ||
          (strcmp(inet_ntoa(ip->ip_dst),IP_PC2)==0))
        {
            /*dunque mi restituisce la size message e il timeStamp dell'header(info che sono incluse nel pkthdr)*/
            printf("Prima di ottenere size mes\n");
            printf("lunghezza pacchetto %d\n",header->len);
            printf("timestamp %d\n",header->ts);
            updateStats(header->len, &header->ts);
        }

}
```

```
void work(u_char *restituito, struct pcap_pkthdr *header, const u_char *packet){
/*devo leggere i pacchetti che ho, e di questi, prendo le caratteristiche come MAC addrress ecc
        e le assegno come caratteristiche di un nuovo pacchetto(a cui mettero di diverso solo la dimensione)
        che creo adesso e che salvero nel mio nuovo file pcap e
        modifico la lunghezza del pacchetto mettendo degli zero in fondo*/

        sniff_ethernet* eth_header;
        sniff_ip* ip;
        //sniff_tcp* tcp;
        sniff_udp* udp;

        int size_ip;
        int random;
        counter++;

        printf("\ncounter %d\n", counter );
        /*effettuo un cast e ora come ora dico che l'intero pacchetto e' identificato come l'header di ethernet*/
        eth_header = (sniff_ethernet*)packet;
        /*con quest altro cast il puntatore ip sta puntando esattamente all'indirizzo che viene subito dopo la parte
        di indirizzi dedicata ad ethernet*/
        ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
        size_ip = IP_HL(ip)*4;
```

```
printf("\n- Prossimo Pacchetto: %d -\n", ++packet_number);

/* lunghezza dell'header*/

printf("\theader size: %d\n", header->len);

/*indirizzo MAC di provenienza e destinazione*/

printf("\tMAC_DESTINATION: %02x:%02x:%02x:%02x:%02x:%02x\n",
(unsigned int)eth_header->ether_dhost[0],(unsigned int)eth_header->ether_dhost[1],
(unsigned int)eth_header->ether_dhost[2],(unsigned int)eth_header->ether_dhost[3],
(unsigned int)eth_header->ether_dhost[4],(unsigned int)eth_header->ether_dhost[5]);

printf("\tMAC_SOURCE: %02x:%02x:%02x:%02x:%02x:%02x\n", (unsigned
int)eth_header->ether_shost[0],(unsigned int)eth_header->ether_shost[1],(unsigned
int)eth_header->ether_shost[2],(unsigned int)eth_header->ether_shost[3],(unsigned
int)eth_header->ether_shost[4],(unsigned int)eth_header->ether_shost[5]);

/* tipo ethernet, e' uno standard, un valore fisso che e' sempre 0x0800.*/

printf("\tType: 0x%04x\n", ntohs(eth_header->ether_type));

/* stampa provenienza e destinazione degli indirizzi IP */

printf("\tFrom: %s\n", inet_ntoa(ip->ip_src));

printf("\tTo: %s\n", inet_ntoa(ip->ip_dst));

/* Time To Live*/

printf("\tTTL: %d\n", ip->ip_ttl);


/* determina di che protocollo si tratta */

switch(ip->ip_p) {

        case IPPROTO_TCP:

                printf("\tProtocollo: TCP\n");

                //tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);

                pcap_dump((pcap_dumper_t*) restituito,header,packet);
```

```
            pcap_dump_flush((pcap_dumper_t*) restituito);
return;

case IPPROTO_UDP:
            printf("\tProtocollo: UDP\n");
            udp = (struct sniff_udp*)(packet + SIZE_ETHERNET + size_ip);
            pcap_dump((pcap_dumper_t*) restituito,header,packet);
            pcap_dump_flush((pcap_dumper_t*) restituito);
            count_udp++;
break;

case IPPROTO_ICMP:
            printf("\tProtocollo: ICMP\n");
            pcap_dump((pcap_dumper_t*) restituito,header,packet);
            pcap_dump_flush((pcap_dumper_t*) restituito);
return;

case IPPROTO_IP:
            printf("\tProtocollo: IP\n");
            pcap_dump((pcap_dumper_t*) restituito,header,packet);
            pcap_dump_flush((pcap_dumper_t*) restituito);
return;

default:
            printf("\tProtocollo: unknown\n");
```

```
                pcap_dump((pcap_dumper_t*) restituito,header,packet);
                pcap_dump_flush((pcap_dumper_t*) restituito);
        return;
    }
    /* siccome esco dallo switch e' UDP.Quindi confronto che destinatario e fonte siano uno dei due indirizzi IP che comunicano */
    /*printf("Prima di entrar nell'if\n");
    printf("indirizzo di provenienza %s\n",inet_ntoa(ip->ip_src));
    printf("indirizzo di destinazione %s\n",inet_ntoa(ip->ip_dst));
    */
    if(
     (strcmp(inet_ntoa(ip->ip_src),IP_PC1)==0) ||
     (strcmp(inet_ntoa(ip->ip_src),IP_PC2)==0) ||
     (strcmp(inet_ntoa(ip->ip_dst),IP_PC1)==0) ||
     (strcmp(inet_ntoa(ip->ip_dst),IP_PC2)==0)
    ) {
        printf("Entra c'e' un UDP\n");

        /* Inizializzo il seed per generazione numeri random */
        time_t seed=time(NULL);
        srand(seed);

        /* Copia pacchetto */
        //memcpy(datagram, packet, header->len);
```

```
if(counter%FREQUENZA==0)
{ for(j=0;j<GRUPPO_PAC_FITTIZI;j++){
            /* Altero un lunghezza IP */
            random = rand()%100;
            ip->ip_len += htons(random);
            random=random - 20;
 //       if(count_udp %2 == 0)
 //         random=last_rand+200;
 //       else
 //         random=rand()%100;

            ip->ip_len +=htons(random);
            /* Altero un lunghezza IP */
            random = rand()%100;
            udp->uh_length += htons(random);
 //       random= random - 20;
 //       udp->uh_length +=htons(random);
            /* Altero tempo di arrivo */
            header->ts.tv_usec += (rand()%10000);
 //       header->ts.tv_usec += (rand()%10000);

            pcap_dump((pcap_dumper_t*) restituito,header,packet);
            pcap_dump_flush((pcap_dumper_t*) restituito);
  }
}
```

```
        }
        printf("\ncounter %d\n", counter );
}
```

Bibliografia

[R1] Dario Bonfiglio, Marco Mellia, Michela Meo, Nicol` Ritacca

, Dario Rossi, Tracking Down Skype Traffic

. In: Proc. Of the IEEE INFOCOM 2008.

[R2] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, Paolo Tofanelli, Revealing Skype Traffic:When Randomness Plays with You, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 5, NO. 4, DECEMBER 2008

.

[R3]

Emanuel P. Freire, Artur Ziviani, and Ronaldo M. Salles, Detecting VoIP Calls Hidden in Web Traffic, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 5, NO. 4, DECEMBER 2008.

[R4] Mauro Migliardi, Roberto Podesta', Matteo Tebaldi and Massimo Maresca, Hiding Skype VoIP Calls from Parametric Identification. In: Proc. of the First ACM International Conference on Forensic Applications and Techniques in Telecommunications. Adelaide (AUS) January 21-23, 2008 (pp. 1-6). ISBN/ISSN: 978-963-9799-19-6. January 21-23, 2008.

[R5] Tanenbaum Andrew S., Reti di calcolatori, Pearson Education Italia.

[R6] aa.vv., Voip, Available on-line at URL http://it.wikipedia.org/wiki/Voice_over_IP

[R7] aa.vv., Wireshark, Available on-line at URL http://it.wikipedia.org/wiki/Wireshark

[R8] aa.vv., Makefile, Available on-line at URL http://www.pluto.it/files/journal/pj9811/makefile.html

[R9] aa.vv., Pcap, Available on-line at URL http://www.tcpdump.org/pcap.html

[R10] aa.vv., Voip, Available on-line at URL http://www.ngway.it/newwebsite/node/24

[R11] aa.vv., Skype, Available on-line at URL
http://www.ilbloggatore.com/a1/2010/05/20/come-usare-skype-in-tutta-sicurezza/