Università degli Studi di Camerino

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica Classe L-31

DEEP PACKET INSPECTION

Analisi del traffico di rete

RELATORE

Prof. Fausto Marcantoni

STUDENTE

Patrik Rosini

Anno Accademico 2011/2012

a Valeria.

Indice

Abstract

Il networking è una parte essenziale dell'informatica e la deep packet inspection rappresenta un potente strumento del settore, adatta a diversi scopi. In questo documento, partendo da una breve trattazione sulla sicurezza delle reti ci si dedicherà a un'ampia discussione su questa tecnica d'ispezione dei pacchetti, dai metodi d'identificazione, agli utilizzi e gli aspetti legali. In seguito, conosceremo da vicino le specifiche di un programma open-source su cui è stata costruita e descritta una dimostrazione pratica.

Networking is an essential part of computer science and deep packet inspection is a powerful tool in this sector, suitable for several purposes. In this paper, beginning from a short debate on network security, i'm going to give an extensive explanation about this packet inspection technique, from identifications methods, to the uses and legal aspects. Then, i'm going to describe, closely, the specifications of an open-source software i choose, used to build and describe a practice demonstration.

Introduzione

La rete informatica è entrata nella quotidianità di quasi ogni essere umano tanto che, essere connessi stia quasi diventando un bisogno primario. Questo crescente atteggiamento, dovuto a diversi fattori tra cui la pervasione dei dispositivi che fanno uso della connessione di rete, sta portando a un esponenziale aumento del traffico di dati in transito nel mondo delle telecomunicazioni su IP.

Ciò evidenzia la necessità di occuparsi di tutte le conseguenze collegate e cioè, dalla sicurezza alle prestazioni, concentrando le ricerche su metodologie e congegni che possano contribuire a queste esigenze. La rete internet però, in generale, è stata progettata secondo una metodica end-to-end, evitando d'inserire all'interno dello scheletro della rete funzionalità del livello applicativo[1]. Seguendo questa regola, si è assistito al suo successo, avvenuto grazie alla sua semplicità e scalabilità. Del resto questi vantaggi portano una limitazione, poiché non è possibile differenziare il servizio di rete in base al tipo di traffico che la attraversa. Per raggiungere tali obiettivi con elevata efficacia e accuratezza, si è compreso, che è un requisito avere accesso diretto al network, ossia la capacità di analizzare il traffico nei minimi dettagli, per conoscere quali dati si abbiano davanti e come gestirli.

Questa tesi, per l'appunto, si focalizza sullo studio della tecnica di analisi del traffico di rete denominata deep packet inspection in un'ottica di sicurezza, cercando di fornire una documentazione da cui partire per ulteriori approfondimenti.

STRUTTURA DELLA TESI

Nel primo capitolo sarà dato uno sguardo generale alla sicurezza delle reti, con un dibattito sui rischi cui si è esposti e i relativi pericoli, per poi discutere delle principali soluzioni disponibili.

Il secondo capitolo si riferisce all'argomentazione principale. S'introdurrà alla tecnica d'ispezione dei pacchetti e alle sue evoluzioni fino a immergersi nella deep packet inspection, scoprendo tutte le sue sfaccettature.

Il terzo capitolo avvia alla parte pratica della tesi. Qui, dopo un breve discorso sulle soluzioni in commercio, saranno descritte le specifiche del programma scelto per la dimostrazione del caso d'uso e di un altro software correlato.

Il quarto e ultimo capitolo descrive il caso d'uso in tutte le fasi compiute, illustrando come applicare l'analisi del traffico di rete in una circostanza reale.

Capitolo 1

La sicurezza delle reti informatiche

1.1 I rischi della rete

La sicurezza delle reti è un problema attuale e presente dai primi sviluppi d'internet. Le cause dei difetti di affidabilità della rete, potrebbero essere imputate alla progettazione poco robusta di tutte le tecnologie che ne fanno parte e anche agli utenti stessi che ne fanno uso. Il rischio che si presenta è di compromettere la riservatezza, la disponibilità, e l'integrità dei dati trasmessi o immagazzinati così come la raggiungibilità dei vari servizi disponibili in rete. A tal punto, si possono considerare diversi raggruppamenti di minacce alla sicurezza raccolti per diverse categorie di rischio[3]:

Minacce di accesso: questo rischio è rappresentato dall'intrusione illecita nella rete e nel sistema, che può avvenire per mezzo d'intercettazione, accesso non autorizzato o usurpazione d'identità. In particolare, l'intercettazione, potrebbe presentarsi nei punti più vulnerabili delle reti, dove avviene la gestione, la concentrazione e il passaggio del traffico di dati ossia router, switch, server;

Caduta della rete: i principali protagonisti di caduta della rete sono gli attacchi informatici dolosi, come quelli rivolti ai server DNS, ai router e gli attacchi di tipo DoS. Indispensabile, includere anche eventi imprevedibili come le catastrofi naturali e l'errore umano;

Alterazione dei dati: la modifica e la cancellazione dei dati, accadono principalmente con l'esecuzione di *malicious* software,

causata spesso da errori dell'utente. Anche se questa problematica riguarda più un host in se che un'intera rete, potrebbe essere possibile diffondere il malware a tutti gli *end system* collegati allo stesso network.

Requisito	Categoria	Tipi di pericoli
Riservatezza	Minacce di accesso	Intercettazione delle comunicazioni, accesso non autorizzato, usurpazione d'identità
Disponibilità	Caduta della rete	Attacchi informatici dolosi: attacchi ai DNS, attacchi ai router, attacchi DoS. Catastrofi naturali, errore umano
Integrità	Alterazione dei dati	Esecuzione di malicious software, errore dell'utente

Tabella 1.1 Analisi dei rischi della rete

Conosciuti i possibili rischi che minacciano le reti di elaboratori cosi come sono oggi, occorre osservare che i problemi di sicurezza si presentano potenzialmente in differenti contesti, che sia una organizzazione oppure un utente domestico, ed è corretto riconoscere che raggiungere una condizione di totale assenza di vulnerabilità non sia strettamente fattibile e altresì possibile invece, seguire opportuni criteri di protezione ed adottare gli adeguati strumenti disponibili per ridurre al minimo l'esposizione ai vari tipi di pericoli.

1.2 Le soluzioni attuali

Le misure tecnologiche che si possono adottare per avere protezione sono molteplici e sia in forma hardware che software. Le trasmissioni di rete posano le loro fondamenta sulla standardizzazione dell'organismo internazionale ISO/OSI[4], che stabilì un modello di riferimento costruito su sette livelli, ciascun preposto a compiere un ruolo differente. Poiché le informazioni durante una comunicazione transitano da un livello a un altro, è possibile che esse siano compromesse, quindi per questo si rende necessario avere contromisure per ogni strato della pila.

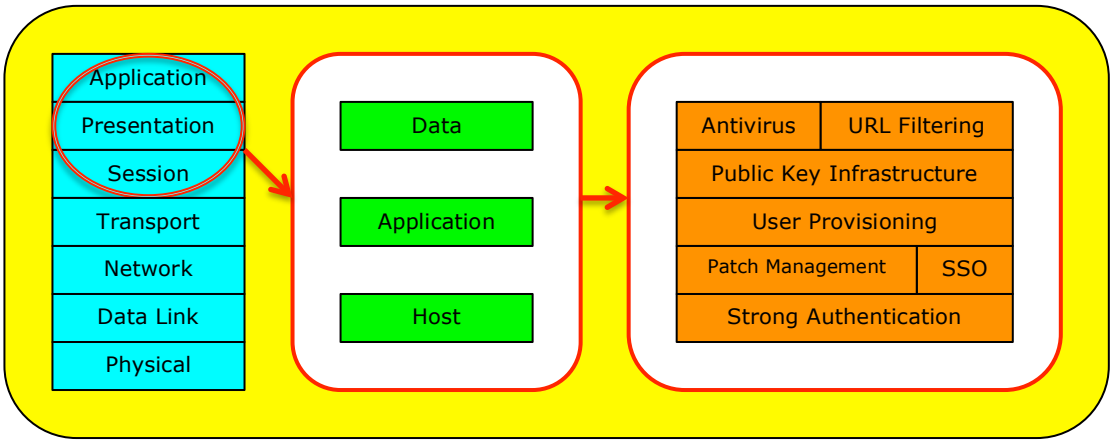

Figura 1.1 Pila ISO/OSI e tecnologie di protezione A

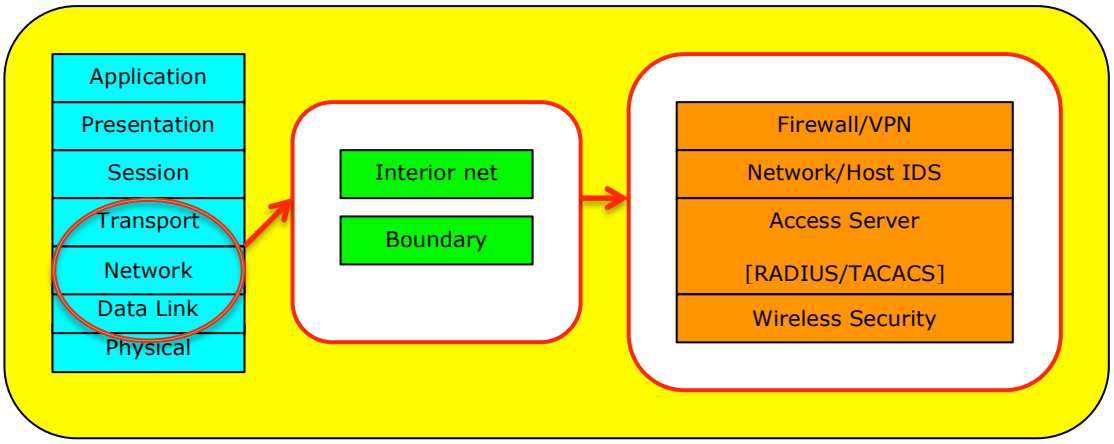

Figura 1.2 Pila ISO/OSI e tecnologie di protezione B

Le Figure 1.1 e 1.2 schematizzano le soluzioni di difesa più diffuse, evidenziate per *asset* nel quale esse operano[3]. Un aspetto abbastanza comune è quello di trovare tecnologie che operano su più livelli, anche se, una buona strategia, probabilmente più efficace, è quella di comprendere il contributo di ogni sistema di protezione nel proprio ambito e poi applicarlo relativamente alla circostanza in cui risulta opportuno usarlo. Assumendo un approccio *top-down* sullo standard ISO/OSI nella rassegna dei dispositivi e delle procedure più diffuse, si può valutare negli strati alti, come gli Antivirus e le PKI, svolgono un ottimo sostegno alla sicurezza, anche con le eventuali anomalie di sviluppo dei software e dei protocolli cifrati, che potrebbero nascondere insidie e poi trasformarsi in minacce. I sistemi di *Patch Management* hanno lo scopo, appunto, di correggere le falle di programmi e sistemi operativi, ma potrebbero arrivare in ritardo sul tempo quando la violazione è già avvenuta o in corso, perché spesso, sono proprio i pirati informatici a conoscere prima dei produttori quali sono le loro debolezze. Le soluzioni di *User Provisioning*, *Single Sign On* e *Strong Authentication,* si occupano di quella cosa, a cui tutte le persone tengono fedelmente nel rapporto con le apparecchiature informatiche: l'identità. Le prime due si concentrano nella gestione dell'utenza, come il controllo dei profili utente nell'accesso alle risorse (ad es. database, gestionali) e l'unificazione delle credenziali multiple. L'ultima cura singolarmente l'identificazione e l'autenticazione di un utente, processo in cui solo le tecniche nel campo della biometria garantiscono maggiori proprietà di sicurezza. Le soluzioni discusse fin qui riducono notevolmente l'errore umano, in questo caso inteso come utente e progettista di un sistema o componente informatico , ma considerando che le minacce proliferano dalla rete stessa, un

metodo di *Url Filtering* può limitare di imbattersi in portali web ricchi di trappole, tuttavia, siccome poco flessibile, si rivela molto restrittivo e quindi poco adottato, ma a volte indispensabile. Proseguendo negli strati bassi, i firewall sono tra i privilegiati mezzi per controllare e proteggere le comunicazioni, perché garantiscono la gestione dei protocolli di basso e alto livello grazie alle diverse tipologie di analisi del traffico dati. Anche gli IDS/IPS, che a differenza dei firewall generici già citati, svolgono un lavoro di solo monitoraggio se IDS, con azione se con IPS, sono notevoli strumenti di autodifesa, ma entrambi sarebbero inefficaci se mal configurati o in caso di attacco dall'interno, visto la loro disposizione perimetrale. Per proteggere l'accesso dall'esterno in un network, reti VPN e i protocolli RADIUS o TACACS, in combinazione tra loro, insieme a una previdenziale *Identity Management*, con le loro caratteristiche di comunicazione crittografata, respingono intercettazioni e intrusioni non autorizzate. Non solo le trasmissioni cablate, ma anche quelle via radio, rappresentano un rischio di accesso, conseguenza di una natura intrinseca di apertura. Quindi, la scelta poco ragionata dei protocolli di sicurezza è risolvibile con una giusta politica di *Wireless Security,* adottando password poco intuibili o sistemi di autenticazione dedicati descritti prima.

Le une e le altre soluzioni, anche se non segnalate, opportunamente congiunte, riuscirebbero a garantire uno scudo abbastanza solido verso quelle falle di tipo tecnico delle tecnologie di rete, ed essere poco contro il fenomeno dell'ingegneria sociale che a oggi, soltanto chi è tra computer e il resto, cioè l'uomo, può contrastare. Tale aspetto fa emergere come la conoscenza dei rischi sia sottovalutata o sconosciuta ad alcuni utenti, ma certamente destinata ad attutirsi.

Capitolo 2

L’analisi del traffico di rete

2.1 L’ispezione dei pacchetti di dati

In questo capitolo ci soffermeremo su una metodologia di controllo e gestione del traffico di rete. Innanzitutto, è ragguardevole fare una piccola premessa, dicendo che ci riferiamo all’analisi del traffico di rete come *packet inspection* che tradotto significa “ispezione dei pachetti”, ossia l’abilità di certi strumenti di networking a catturare i pacchetti di dati ed estrarne le informazioni che contengono. In cosa consiste l’ispezione dei pacchetti di dati?

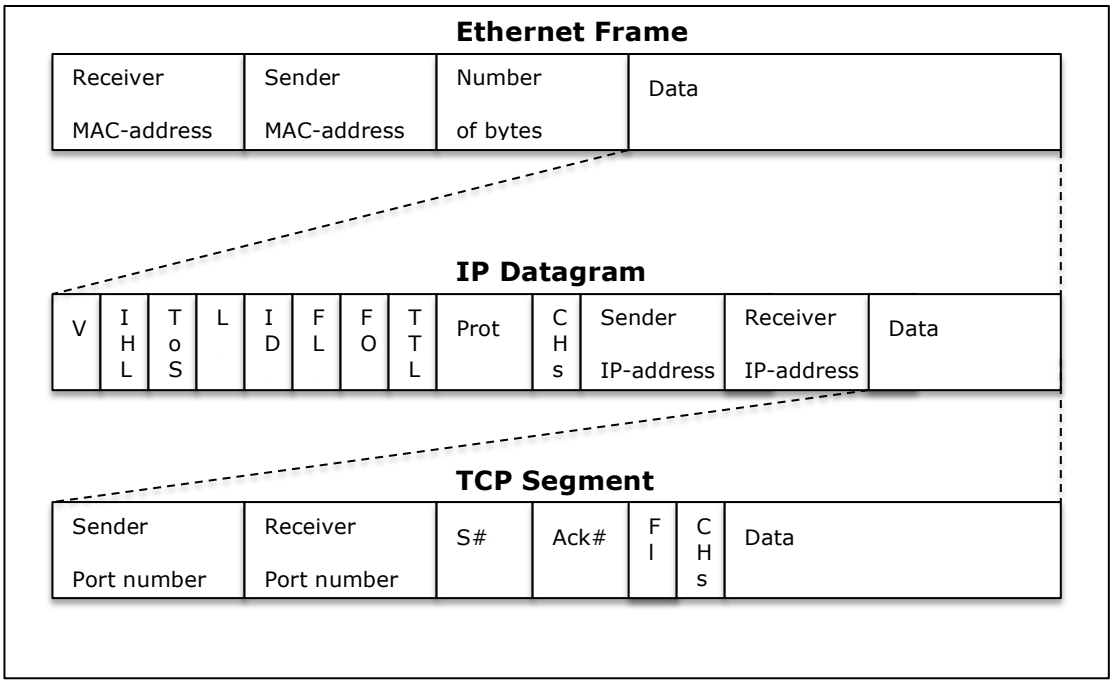

Figura 2.1 Pacchetto di dati e incapsulamento

Vediamo un pacchetto come un blocco, suddiviso in *header* e *data*. L'intestazione o header è la parte che contiene tutte informazioni utili per la trasmissione, mentre la parte data contiene i dati utili tramessi, comunemente detta *payload*. Con dati utili, s'intendono le informazioni che le applicazioni software producono, come il testo di un'email, la pagina html di un sito web, ecc.. L'ispezione dei pacchetti di dati appunto, usa le due parti per avere informazioni sul traffico dei pacchetti che attraversano un segmento di rete.

Esplorando più da vicino in che cosa consiste, bisogna essere coscienti che le tecnologie di analisi dei pacchetti esistono e sono utilizzate da almeno più di un decennio. Possiamo fare riferimento a tre classi di packet inspection ossia *shallow, medium e deep*[5] che differiscono sulla loro capacità di analisi dei pacchetti e qui illustrate nella tabella seguente:

Level	Shallow Packet Inspection	Medium Packet Inspection	Deep Packet Inspection	ISO/OSI
7				Application
6				Presentation
5				Session
4				Tranport
3				Network
2				Data Link
1				Physical

Tabella 2.1 Classi di packet inspection

La categoria definita shallow packet inspection rappresenta la più semplicistica fra tutte, poiché l'ispezione è limitata soltanto sull'header. Il controllo, di tipo *stateless,* è basato sugli indirizzi IP di origine e destinazione ed è usualmente impiegata per bloccare connessioni in ingresso con un metodo di comparazione basato su *blacklist,* che può contenere regole predefinite o create e modificate su misura. Per facilitare la configurazione, gli apparati o i software più avanzati con funzionalità SPI, presente anche nei moderni sistemi operativi, elaborano dei log sugli header in entrata e in uscita tali da poter essere rivisti e studiati per migliorare l'insieme delle regole.

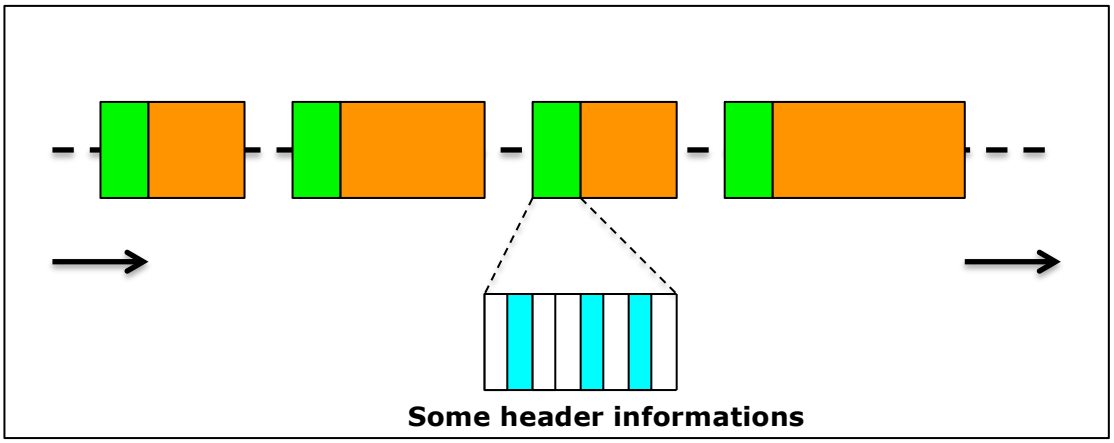

Figura 2.2 Shallow packet inspection

Quindi, questa categoria d'ispezione, non può carpire informazioni oltre a quelle di un'intestazione di pacchetto e cioè, l'indirizzo IP del mittente e del destinatario, il numero di pacchetti in cui il messaggio è frammentato, il numero di *hop* che il pacchetto può attraversare prima che sia scartato dai router, le informazioni di sincronizzazione necessarie per riunire i pacchetti frammentati ricevuti, ed è palese che sia insufficiente e impreciso per avere un'idea sull'applicazione che li ha generati.

La successiva categoria, denominata medium packet inspection, ha questa sistemazione, per così dire tra il più e il meno, poiché l'ispezione si basa sull'header e alcuni aspetti del payload. Il controllo in questo caso è di tipo *stateful,* perché sono considerate tutte le informazioni di trasmissione, includendo lo stato di una connessione su alcuni protocolli, come ad esempio il TCP. Con MPI, si fa spesso riferimento agli *application proxy* o quei apparati che si interpongono tra gli utenti finali e gli ISP, che centralizzano tutti i flussi di dati e sono in grado di confrontarli su una *parse-list*. Ciò, ha il vantaggio di rendere separati la sorgente dalla destinazione di un pacchetto, rendendo l'application proxy un intermediaro tra l'utente finale e la grande rete. La parse-list è una lista di regole più raffinata della blacklist, poiché non agisce basandosi solo sull'intestazione del pacchetto, bensì anche dalle informazioni reperite dai primi byte del payload. In questo modo è possibile impedire la ricezione dei file d'immagini dai siti di social network o dei file flash dai siti di streaming video, come anche dare priorità a pacchetti che contengono comandi di applicazioni, tipicamente limitati a Telnet, HTTP e FTP, o certi tipi di formato dei file. Purtroppo, gli apparati MPI hanno una scalabilità scarsa dovuta alla natura stessa della loro implementazione, perché tutti i pacchetti conferiscono in un solo punto per essere analizzati e ciò aggiunge ritardi nella loro ricezione e consegna, soprattutto per le reti di grandi dimensioni, come quelle degli ISP, dove decina di migliaia di byte sono trasferiti in ogni momento. Alla luce di quanto descritto, in questa categoria si arriva a conoscere il livello di presentazione (*presentation layer)* e una piccola sezione del livello di applicazione *(application layer)* del payload di un pacchetto, con un confine davvero sottile con la categoria pilota di packet inspection, che verrà discussa nel parafrago 2.2.

2.2 La deep packet inspection

Entriamo nel vivo della tesi, trattando la tecnica attuale di analisi del traffico di rete. La deep packet inspection consente di catturare completamente tutto il pacchetto, ossia header e payload, fornendo un controllo completo da livello due al sette. Ciò significa, che è possibile guardare dentro tutto il traffico proveniente da un host e ad esempio, ricostruire una pagina HTTP o un'intera conversazione email. L'esame dettagliato del traffico di rete, permette anche di poter analizzare applicazioni che usano *ephemeral port*, cioè porte di servizio dinamiche e applicare la cross packet inspection[6], dimodoché sia fattibile rilevare contenuti che si suddividono su più pacchetti, cioè frammentati. In generale, con la DPI si possono realizzare diverse tecniche di tracciamento del traffico:

- Packet-Based No State (PBNS): si considera ogni pacchetto singolarmente su alcuni parametri generici;
- Packet-Based per Flow State (PBFS): si considerano i pacchetti per flusso, ossia la sessione di comunicazione instaurata da un'applicazione tra due host, rappresentata da una quintupla (SRC IP, DST IP, SRC PORT, DST PORT, IP PROTOCOL);
- Message-Based per Flow State (MBFS): simile al PBFS, ma tiene conto delle informazioni sulla frammentazione del livello tre e di assemblaggio del livello quattro;
- Message-Based per Protocol State (MBPS): racchiude le precedenti e in più considera tutte le informazioni complete sullo stato del protocollo.

La DPI opera abitualmente su costosi dispositivi, in effetti, è una tecnica usata dai dispositivi firewall con caratteristiche IDS/IPS

ma è comune il loro ambito di impiego su router avanzati, con elevate capacità di calcolo, perché al contrario degli apparati MPI, sono progettati per essere scalabili ed operare in ambienti di rete con grandi dimensioni, essendo in grado di prelevare ogni informazione dai pacchetti in tempo reale su centinaia di migliaia di transazioni al secondo. Sebbene i flussi di dati, siano normalmente verificati a velocità di collegamento, nel caso non sia possibile capire il contenuto del payload all'istante, si potrebbe far ricorso alla deep packet *capture*[7], ovverosia, la raccolta in memoria dei pacchetti per breve o lungo termine, per poi essere esaminati euristicamente in un secondo momento, su modelli di flusso conosciuti.

2.2.1 I metodi di analisi

Il meccanismo della DPI per l'identificazione dei contenuti è basato sulla *signature*. Questa, è un'associazione alle caratteristiche di un'applicazione software o protocollo di comunicazione, come un'impronta digitale, che prende il nome nel gergo tecnico di *pattern*. La signature è usata anche nelle tecniche d'ispezione prima viste e consiste nel confrontare ogni pacchetto su una banca dati univoca per ogni applicazione o protocollo. Certamente, le referenze devono essere aggiornate periodicamente, di pari passo con l'aggiornamento delle applicazioni e i nuovi sviluppi sui protocolli esistenti, con l'obiettivo di contrastare la comparsa dei casi di falso positivo, cioè l'identificazione falsa, attraverso la combinazione di più pattern e dei casi di falso negativo, cioè dell'identificazione non avvenuta, mediante una completa casistica degli scenari di funzionamento delle applicazioni o protocolli.

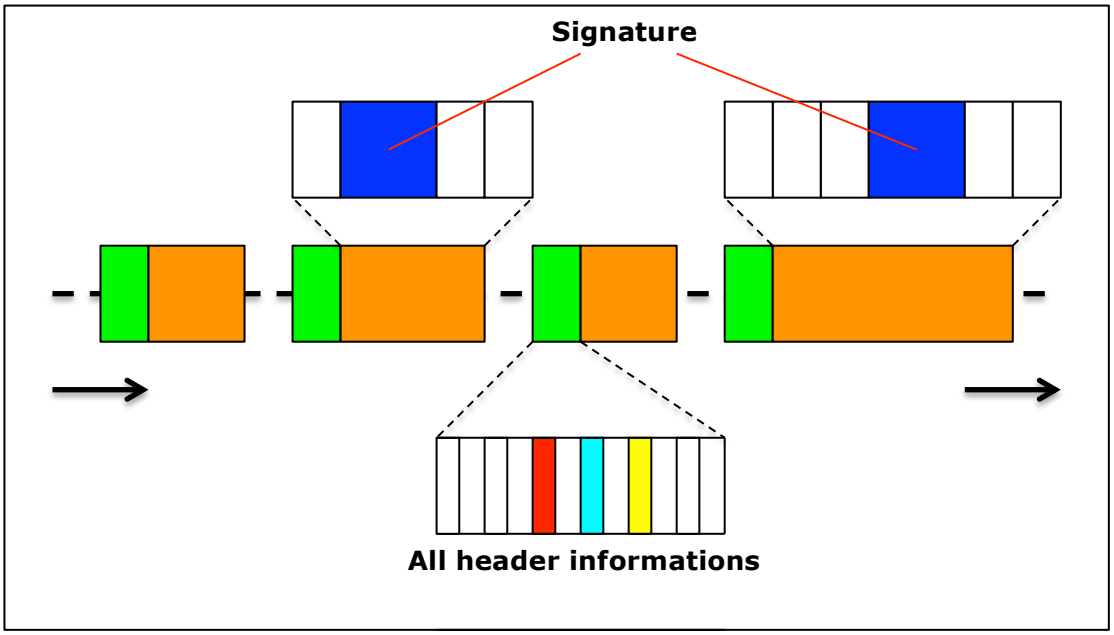

Figura 2.3 Deep packet inspection

Esistono diversi metodi di analisi per costruire una signature e si basano su criteri differenti, riassunti con i seguenti[8]:

- *analisi sulle stringhe;*
- *analisi sulle proprietà numeriche;*
- *analisi comportamentale e statistica.*

Analisi sulle stringhe: l'analisi sulle stringhe o *pattern matching* esegue una ricerca di stringhe o caratteri numerici contenuti nel payload del pacchetto. La posizione di questi elementi non è sempre fissa per ogni applicazione o protocollo e deve essere comunque possibile cercarli. La comparazione di stringhe, è un metodo di analisi basilare e abbastanza efficace, che si basa su molti studi fatti in merito, tra cui, gli algoritmi Knutt-Morris-Pratt (KMP), Bayer-Moore (BM), Aho-Corasick (AC), l'algoritmo AC_BM, Wu-Manber e Commentz Walter (CW), cosi come l'uso delle strutture dati Bloom Filter[9]. La Figura 2.5, illustra il caso dell'applicazione Kazaa, identificata con comparazione di stringa sulla voce *user-agent* di una richiesta HTTP GET.

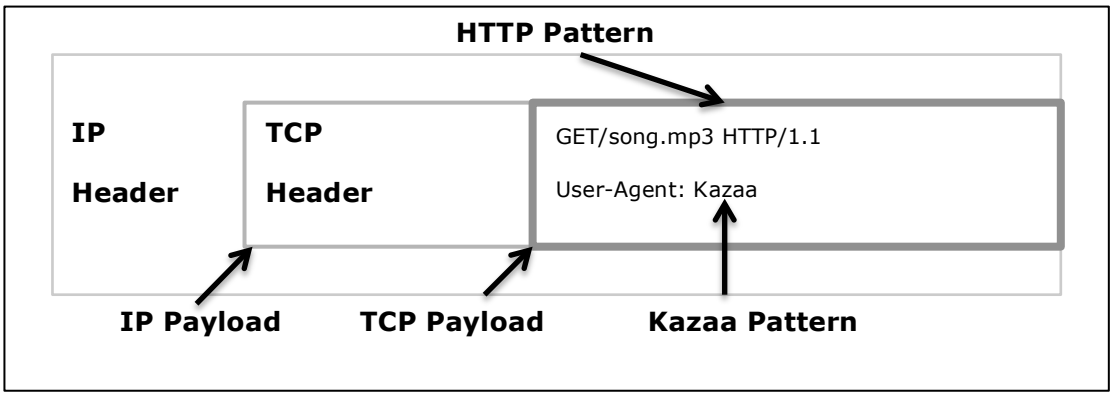

Figura 2.4 Analisi sulle stringhe, Kazaa

Analisi sulle proprietà numeriche: l'analisi sulle proprietà numeriche verifica le caratteristiche numeriche dei pacchetti come ad esempio la grandezza del payload, il numero dei pacchetti scambiati o semplicemente il numero della porta di servizio. Un esempio diretto è quello dell'applicazione Skype fino alla versione 2.0.

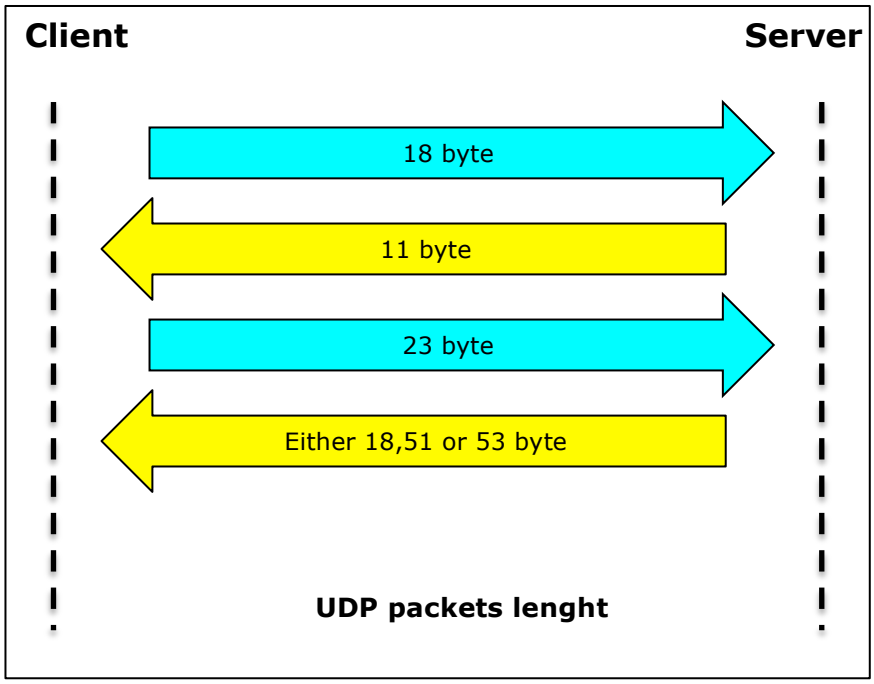

Figura 2.5 Analisi sulle proprietà numeriche, Skype

Analisi comportamentale e statistica: l'analisi comportamentale si riferisce al come un protocollo funziona. Alcune applicazioni seguono una procedura fissa di azioni (ad es. SMTP, ecc.), offrendo diretti riferimenti per l'identificazione. In altre, l'analisi comportamentale deve essere aggregata a un analisi statistica, compiuta raccogliendo parametri statistici sui pacchetti esaminati. Un caso esemplare è distinguere il traffico HTTP da quello P2P sulla stessa porta 80. Questo esame, è improntato sulle differenze di lunghezza dei pacchetti delle due specie di traffico, infatti, se il traffico puro HTTP, cioè ignorando il download e l'upload di file, tende ad avere una lunghezza del payload di poche centinaia di byte, il tipico traffico P2P ha dimensioni molto più corte. La Figura 2.7 ritrae il caso appena descritto.

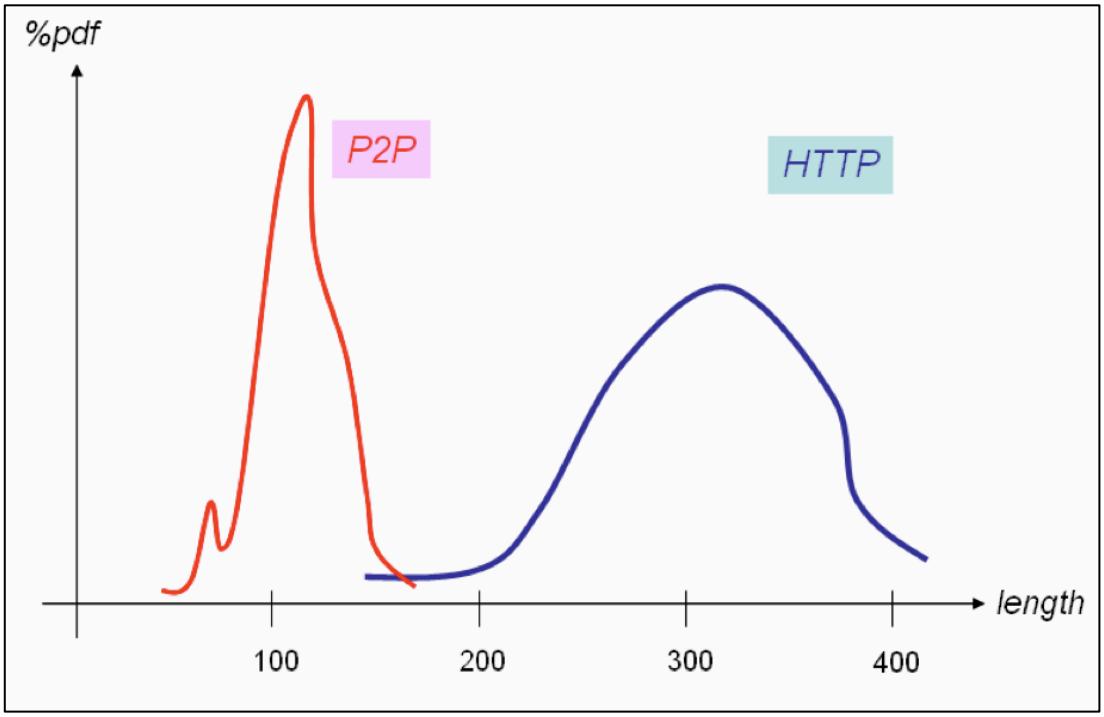

Figura 2.6 Analisi comportamentale e statistica, HTTP vs P2P

Ricapitolando, nessuno di questi metodi da solo provvede all'identificazione di tutte le applicazioni o protocolli software, tantoché quasi ogni volta sono utilizzati insieme.

Un'ultima riflessione sui metodi di analisi del traffico di rete, è rivolto alle connessioni criptate oppure offuscate, che porgono una sfida ardua ai meccanismi di rilevamento per un'accurata identificazione. Anche se criptazione e offuscamento sono codifiche differenti, entrambi hanno l'obiettivo di rendere illegibile il contenuto del payload, quindi, un'analisi comportamentale e statistica potrebbe aiutare a identificare qualche applicazione o protocollo. Nuovi metodi probabilistici si affidando agli algoritmi di *clustering*, ma alcuni prodotti commerciali come SonicWall DPI-SSL[10], cercano di risolvere il problema in maniera più pratica.

2.2.2 I suoi utilizzi

Dopo il breve excursus tecnico sulla DPI andiamo adesso a descrivere i suoi principali utilizzi basati sulle fondamentali funzionalità di questa tecnica d'ispezione, che possiamo dire sono il riconoscimento, la manipolazione o la notifica[11]. Esse, sono combinate sulle esigenze che riflettono le diverse condizioni di chi faccia uso della DPI. Un'altra distinzione che poniamo è sulla modalità d'impiego, cioè passiva o attiva. Un esempio di modo d'uso passivo è la semplice classificazione dei protocolli ai fini statistici, per il monitoraggio dello sfruttamento della banda. Differente può essere, la posizione sulla mappa di rete e faremo, anche qui, una divisione tra online e offline. Infatti, nel caso di manipolazione dei pacchetti è obbligatorio essere direttamente collegati al flusso di rete, quindi online, con lo svantaggio di diventare un possibile punto di guasto, seppur non modificando la topologia di rete. Viceversa, se l'analisi dei pacchetti non accade in

tempo reale, l'ispezione avviene offline, per cui saranno elaborati copia dei pacchetti catturati.

Proprietà della DPI	Riconoscimento	Manipolazione	Notifica
Attiva		X	
Passiva	X		X
Online	X	X	X
Offline	X		X

Tabella 2.2 Proprietà della DPI

Detto ciò, si elencano i principali casi di utilizzo di questa tecnica[12]:

- *Visibilità di rete e Gestione di banda;*
- *Sicurezza di rete;*
- *Identificazione profili utente;*
- *Sorveglianza dei governi*;
- *Protezione dei diritti d'autore;*
- *Censura o regolazione dei contenuti*.

Visibilità di rete e Gestione di banda: è l'utilizzo della DPI per eccellenza e consente di capire e analizzare la composizione del traffico, per intervenire attivamente e in maniera discriminante nella sua gestione. L'obiettivo principale è quello di massimizzare la larghezza di banda, anche sfruttando le abitudini di utilizzo degli utenti, garantendo QoS. La finalità consiste nella classificazione delle applicazioni software e dei protocolli di comunicazione in esecuzione sulla rete, spesso accompagnata da report statistici che offrono un quadro preciso di preziose informazioni, per la

risoluzione di eventuali problematiche, come la regolazione della larghezza di banda e l'ottimizzazione della congestione. Nella pratica, s'interviene limitando la banda allocata e dare o togliere priorità a specifici protocolli o servizi.

Sicurezza di rete: la ricerca sulla sicurezza di rete ha contribuito fortemente allo sviluppo delle capacità della DPI già dagli anni 90, quando per il riconoscimento di malware con la cattura di pacchetti e la loro analisi, furono creati gli IDS, poi estesi dagli IPS che agiscono sugli *exploit* riconosciuti, tipicamente chiudendo la connessione. Con l'ispezione completa dei pacchetti si è in grado ad esempio di rilevare *keylogger*, *botnet*, email di *phishing*, *buffer owerflow* e contribuire alla prevenzione della perdita dei dati, meglio nota come *data loss prevention*. La DLP assicura che non avvenga fuga d'informazioni riservate, specialmente in ambiti aziendali, al di fuori della rete interna, quindi, si sottopone il traffico in uscita costantemente sotto controllo, prevenendo la violazione delle politiche di sicurezza. Un altro uso ai fini della sicurezza è quello dell'analisi forense del traffico di rete, con cui gli investigatori catturano ed archiviano informazioni su intrusioni, crimini informatici e transazioni anomale, nella persecuzione di attività illecite.

Identificazione profili utente: con l'ispezione completa dei pacchetti che transitano sulla rete, si possono creare profili utente, ricavando dati su come questi usano il network. Con informazioni cosi importanti sugli abbonati, un ISP, sotto accordi con una società di pubblicità, potrebbe iniettare annunci affini agli interessi rilevati dal traffico web, fenomeno noto come *targeted advertising*. Altrettanto critico è la discriminazione sull'utilizzo di certi servizi come il VoIP, offrendolo con pagamento di un supplemento sull'abbonamento,

mediante raffinati sistemi di fatturazione, in relazione al volume di traffico usato per il servizio.

Sorveglianza dei governi: le leggi nazionali possono richiedere agli ISP la capacità d'intercettazione o di sorveglianza come regolamento di governo, oppure come strumento per gli organi di sicurezza. Normalmente, nei paesi democratici, l'uso della DPI è regolato da normative severe, ed è permessa solo su persone indagate, stabilendo, quali dati poter raccogliere, per quanto tempo e quali possono avere valore giudiziario. In questo caso, il procedimento è definito "intercettazione legale". Tali poteri, se si trovano in paesi dove il governo ha meno leggi da considerare, la sorveglianza dell'utente senza restrizioni, vincolano fortemente le comunicazioni dei soggetti.

Protezione dei diritti d'autore: la perseveranza del file-sharing, ha sollecitato i titolari dei diritti a collaborare per trovare una soluzione alla protezione del copyright. L'idea ambiziosa, è quella d'identificare in tempo reale i contenuti protetti da copyright che scorrono sulla rete degli ISP e inviare una notifica ai loro utenti quando avviene il rilevamento. Il progetto prevede che ognuno degli innumerevoli contenuti digitali sparsi sul web, debba essere fornito di univoca firma regolarmente registrata, che possa essere individuata al momento dell'ispezione. Questa strada è stata perseguita dall'azienda Audible Magic[13], che ha brevettato un metodo di firma, come un "impronta digitale" dei contenuti multimediali, con algoritmi che considerano le caratteristiche audio del file. Una soluzione classica praticata è invece, quella dell'identificazione e blocco preventivo delle applicazioni e dei protocolli P2P di condivisione dei file. Alternativamente, si procede a una limitazione della banda, ma entrambi anche se soluzioni

meno complesse rispetto alla comparazione delle firme, non distinguono un onesto utilizzo del P2P da quello illecito.

Censura o regolazione dei contenuti: la DPI, permette di praticare la censura dei contenuti o dall'inglese *content filtering*. Può essere realizzata bloccando gli URL di siti proibiti o deviando qualsiasi contenuto che contiene parole chiave o frasi specifiche. L'attivazione della censura può avere luogo sia in ambito professionale che privato, o voluta dal governo di un paese. Di per sé, il blocco degli URL può essere realizzato anche da un proxy, perché la DPI non aggiunge e non toglie migliorie, utile però, nel caso si vogliano conoscere i contenuti nel dettaglio. Tuttavia, la debolezza degli algoritmi basati su pattern matching e la valutazione di concetti vaghi, come "oscenità" o "illegalità" e come notato in precedenza, il complesso riconoscimento sulle firme dei contenuti multimediali, rendono il metodo del blocco degli URL la forma più comune di censura. Le ragioni di restrizione variano dal principio di miglioramento di banda, secondo l'orario del giorno e dei giorni della settimana, alla riduzione delle responsabilità legali (ad es. violazione copyright). Sebbene il web, sia il principale obiettivo di controllo dei contenuti, i filtri potrebbe essere impostati anche sui contenuti di email ed eventuali allegati, sulla base delle norme di conformità delle politiche di un'organizzazione.

2.2.3 Aspetti legali

Abbiamo descritto i casi d'uso della deep packet inspection. Prendiamo ora in esame gli aspetti legali dei suoi utilizzi. Come riferimento a situazioni reali ci si concentrerà sul territorio europeo e nord americano, luoghi dove l'adozione in alcune forme della DPI, esalta maggiormente il contrasto con le attuali leggi di questi paesi democraticamente sviluppati. Si possono definire quattro aeree per

cui l'uso di questa tecnica è in disaccordo con alcune disposizioni legali: la *privacy*, la *libertà di espressione*, la *concorrenza* e i *processi legali*[14].

Il problema della privacy è forse il più delicato, in quanto il comportamento degli utenti su internet viene monitorato e utilizzato per vari scopi come lo dimostrano i casi d'uso della sorveglianza dei governi e dell'identificazione dei profili utente. Negli Stati Uniti, infatti, la EFF ha affermato che le regole della FISA sull'uso della DPI, per l'intercettazione delle comunicazioni dei cittadini è anticostituzionale poiché violano il quarto emendamento della Costituzione che li tutela contro "perquisizioni e sequestri ingiustificati". Nel vecchio continente, in Europa, la privacy è tutelata dall'art. 8 della ECHR[15], cui pone l'obbligo di competenza agli stati contraenti. A questo si aggiunge la direttiva sulla protezione dei dati della UE, disciplinando il trattamento dei dati personali in riferimento alle organizzazioni pubbliche e private. Quindi, i dati personali, per essere trattati, c'è bisogno che sia fatto in modo trasparente, informando l'interessato e cercando il suo consenso, per finalità legittima e conforme al principio di proporzionalità. Riguardo la pubblicità mirata si segnalano due casi, tra i quali quello europeo che ha coinvolto il provider BT Group per l'uso del software Phorm e quello americano simile al precedente dal nome NebuAD utilizzato dall'ISP WideOpenWest[16].

Un altro modo negativo d'impiego della DPI, è la gestione poco neutrale della rete, tanto da poter infrangere il diritto di libertà di espressione su internet. Il principio della neutralità o *net neutrality*, sostenuto da famosi esperti sulla legislatura delle telecomunicazioni, recita il divieto degli operatori a banda larga, in assenza delle prove di un danno, di limitare ciò che gli utenti fanno con la loro connessione alla rete. Studiosi americani hanno fatto

emergere diverse preoccupazioni sulla libertà di espressione nella gestione delle reti. Infatti, il cosiddetto *access-tiering* cioè la priorità data a certi contenuti web, con un aumento di banda dedicata, in assenza di neutralità della rete darebbe la possibilità agli ISP di vendere priorità di accesso ai contenuti di certe aziende surclassando quelli di altre, mettendo in piedi un sistema simile alla TV via cavo, in cui strapotenti società decidono cosa può essere visto e quanto costa. In Europa, l'art. 10 della ECHR[17], riconosce agli utenti finali e ai fornitori di contenuti, il diritto fondamentale di ricevere e trasmettere informazioni, procurando implicazioni al comportamento non neutro degli ISP e, come già detto, tali diritti essendo competenza degli stati che hanno aderito, non garantiscono la coerenza in tutti i paesi, con conseguenti risvolti negativi sulla neutralità di rete. In tal proposito l'UE, nelle sue Direttive di Accesso[18], specialmente con l'art. 5, incoraggia le autorità nazionali a garantire un accesso adeguato e di interconnessione e interoperabilità dei servizi, tenendo conto di ciò che potrebbero beneficiare agli utenti finali, che non equivale ad avere certezza sulla libertà di espressione sull'utilizzo dei mezzi DPI per la manipolazione del traffico, specialmente sui contenuti. Da un altro punto di vista dare priorità su alcuni tipi di servizi come il VoIP si possono considerare eccezioni sul principio di neutralità di rete.

Nell'area della concorrenza commerciale l'uso della DPI genera anche qualche problema. Infatti, ipotizzando uno scenario con access-tiering, si creerebbe un predominio economico e/o un atteggiamento oligopolistico dei mercati per la fornitura dei servizi internet. Negli Stati Uniti, anche se la neutralità della rete, protegge il mercato libero e competitivo dei contenuti internet, l'uso della DPI potrebbe permettere agli ISP, a comportarsi non

neutralmente e anticoncorrenziale, visto che le normative statunitensi hanno permesso il duopolio dell'accesso alla banda larga. Diversa la situazione Europea, dove il mercato essendo molto più competitivo, frenerebbe la problematica, tra l'atro impedito dall'art. 102 delle Direttive di Accesso[19]. Le conseguenze, sostengono alcuni esperti, potrebbero essere la perdita di economia di scala o riscontri negativi sul mercato di rete, perché un fornitore di contenuti potrebbe essere escluso senza ragioni commerciali. Un piccolo beneficio, invece, potrebbe essere tratto dall'incentivazione sulla qualità dei contenuti, anche se poco probabile, poiché il rischio di essere "bannati" supererebbe quello di fare investimenti.

Concludendo, è doveroso segnalare, che l'impiego della DPI per la prevenzione e la violazione del copyright ha sollevato grandi discussioni, tutt'oggi poco chiare. Infatti, le questioni legate al diritto di autore riguardano generalmente civili da una parte e il diritto penale dall'altra, quindi solo l'esame giurisdizionale può operare in merito, per cui se gli ISP controllassero gli utenti della rete come i disegni di legge SOPA,PIPA e ACTA propongono in onore del copyright, si troverebbero a violare i principi per un processo legale dato che il provider di fornitura della rete potrebbe trarne un ingiustificata funzione poliziesca e di giurisdizione. Dimostrazione del fatto, sono le critiche mosse dall'ACDT che ha evidenziato diverse pratiche, con dietro richieste governative o iniziative di ISP, contrarie a processi legali[20]. In Francia però con la legge HADOPI, vige un controllo diretto sul copyright, pur con alcuni dubbi sulla trasparenza di questo provvedimento.

Capitolo 3

I motori per l'ispezione dei pacchetti

3.1 Gli attori del settore

Il mercato dei dispositivi per la deep packet inspection sono in continua evoluzione. Questo fervore nella ricerca e sviluppo è alimentato dal crescente numero di applicazioni sociali come l'instant messaging, VoIP, video on demand e giochi online, dalla proliferazione di portali di pubblicazione come blog, wiki, video sharing e in generale la dipendenza da internet costringendo produttori e ISP ad evolversi per facilitare download e upload (web, P2P), traffico in tempo reale (VoIP), traffico sociale (IM, gioco), fatturazioni in rapporto volume dati/tempo (PPV), la sicurezza e per riassumere il tutto, la qualità dell'esperienza. Quanto appena detto richiede sforzi da parte dell'industria per una sfida di progettazione che segue alcuni pseudo principi[21]:

- capacità di costruire librerie ampie e complete di signature;
- capacità di eseguire rapide ed efficaci ispezioni in tempo reale, riducendo il più possibile le latenze, con ispezioni di pacchetti sull'ordine del multi-gigabit;
- capacità di creare e rendere disponibili database con statistiche e illustrazioni sui dati per le attività di gestione.

Le caratteristiche sopra elencate sono auspicabili per avere un "motore" DPI funzionale e performante sia in forma software che con hardware dedicato. Se una piattaforma con hardware dedicato (ad es. TCAM, FGPA, multi-core CPU) fornisce grande velocità e

versatilità, i costi di questi dispositivi, fanno settorializzare la loro implementazione in ambienti ad alto volume di traffico tipico dei data center. Una piattaforma software ha sicuramente un approccio più conveniente in termini di costo bensì, le performance ridotte, dovute al carico di elaborazione maggiore, cui le unità centrali sono sottoposte, collocano questo tipo di soluzione in contesti di medio basso traffico di rete. Tutti i produttori di dispositivi DPI sul mercato offrono sistemi proprietari, ma con le stesse finalità e a oggi i primi tre sono[22, 23, 24, 25]:

- Sandvine;
- Cisco Systems;
- Allot Communications.

Nonostante le numerose polemiche su questa tecnica di analisi del traffico, nel mercato globale, si stima che da 250 milioni di dollari del 2009, si arriverà a un giro di affari di circa 2 miliardi di dollari entro il 2016[26], con la vendita dei prodotti appartenenti alla categoria, principalmente nelle aree geografiche di Asia ed EMEA.

Anche nel mondo open source, la DPI ha trovato interessi per tirare fuori alcuni progetti software, considerevoli per chi ha bisogno di ridotte capacità di gestione della rete. Di seguito una lista delle applicazioni di maggiore rilievo[27, 28, 29]:

- nDPI;
- L7-Filter;
- Libprotoident.

Nel nostro caso, il software nDPI è stato scelto per la dimostrazione di un caso d’uso e verrà presentato in dettaglio nel successivo paragrafo 3.2.

3.2 Il progetto nDPI

Il progetto nDPI fa parte della schiera dei software open-source nell'area networking dedicate alla tecnica della deep packet inspection. Esso è lo strumento utilizzato per la dimostrazione del caso d'uso di questa tesi, scelta avvenuta grazie alla sua qualità di progettazione e di supporto tecnico che hanno suscitato molto interesse.

nDPI è una libreria rilasciata sotto licenza GNU GPL come duplicazione ed estensione dal famoso *framework* OpenDPI[30] dell'azienda europea Ipoque. Quest'ultimo derivato dal riconosciuto prodotto PACE dell'azienda stessa, è stato pubblicato con condizioni GNU LGPL, quindi con libero utilizzo e sviluppo, come atto di trasparenza commerciale, in risposta alle critiche additate alla DPI sulla neutralità della rete e altre questioni viste al paragrafo 2.2.3, che hanno sporcato la reputazione dei sistemi di ispezione completa dei pacchetti. Anche se non più reso disponibile, la validità tecnica è testimoniata dalla volontà, del gruppo di sviluppatori del già noto software di monitoraggio di rete ntop[31], di creare un *superset* del prodotto tedesco, per essere inglobato in svariati altri progetti e renderlo accessibile e utilizzabile *stand-alone*. L'obiettivo proposto e inseguito dalla comunità di sviluppatori, è l'allargamento della libreria originale con l'aggiunta di nuove signature, che al momento di scrittura di questo testo se ne contano oltre un centinaio (Tabella 3.1), includendo anche protocolli criptati e offuscati, disponibili solo nella versione a pagamento PACE. Lo sviluppo è rivolto al multipiattaforma, rendendo la libreria compatibile, oltre che sui sistemi Unix-like, anche sui sistemi Windows. Inoltre, sono molto gettonate le fusioni con altri prodotti della stessa comunità, come l'acceleratore di

cattura dei pacchetti PF_RING (paragrafo 3.2.2), e al di fuori, come il modulo *kernel* GNU/Linux Netfilter con Iptables.

3.2.1 Le specifiche

Il nucleo di nDPI è una libreria software progettata per classificare il traffico di rete secondo i protocolli di comunicazione delle applicazioni. Se questa è la forma in cui il framework si presenta per com'è costruito originalmente, ma la sua natura open-source permetterebbe sviluppi differenti che potrebbero conferire nuove caratteristiche alla libreria, quindi non solo di riconoscimento e notifica ma anche di manipolazione dei dati. Il software interamente sviluppato nel linguaggio C, ha mancanza di documentazione tecnica poiché non commerciale, quindi costringe lo sviluppatore, ha studiarlo dall'interno, cioè dal codice stesso.

Si potrebbe dare una descrizione dell'architettura in base ad un'analisi sommaria venuta fuori nel corso del suo utilizzo:

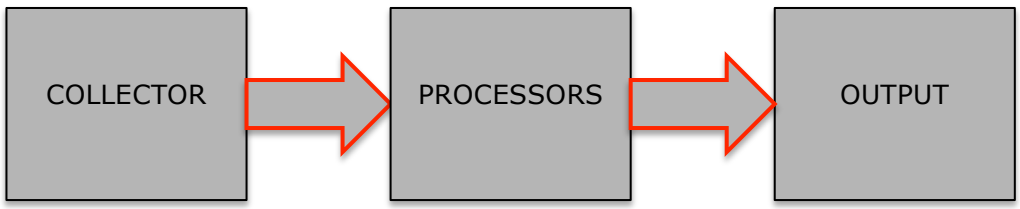

Figura 3.1 Architettura di nDPI

Il blocco COLLECTOR si riferisce alla lettura dei pacchetti, realizzabile sia da un file .pcap o dalla cattura in tempo reale attraverso un interfaccia di rete, meglio se con l'ausilio di PF_RING, utilizzando la nota libreria Libpcap. Il blocco successivo PROCESSORS è il core dell'iter di elaborazione, ed è la fase in cui i

pacchetti sono processati uno a uno e classificati sulla base dei dati contenuti, analizzati dalle signature, sui pattern di riconoscimento dei protocolli o applicazioni supportate. La fase finale, cioè il blocco OUTPUT, di base, si riferisce alla semplice stampa su terminale delle informazioni sui protocolli eventualmente rilevati e alcune cifre di riepilogo, ma si potrebbe riferire anche al prelevamento dei risultati per essere gestiti da un'altra applicazione.

Il framework opera sullo strato *userspace* del sistema operativo evitando che un bug, molto probabile con il continuo aumento delle signature, possa mandare in crash il sistema intero. Esso può utilizzare quasi tutti i metodi di analisi noti, a discrezione del programmatore e dipendente dal caso da analizzare, ma in generale sono l'analisi sulle stringhe del payload, l'analisi numerica e l'analisi comportamentale. A differenza di altri programmi dello stesso genere, non si fa uso delle espressioni regolari poiché, salvo le soluzioni proprietarie, si può affermare che siano troppo semplicistiche per rappresentare la complessità delle strutture dei protocolli e delle applicazioni odierne.

nDPI ha per ogni signature un file .c (Figura 3.2), offrendo tutte le libertà di programmazione che permettono un accurata analisi del pacchetto. La caratteristica, seppur sia la prova di un prodotto robusto ed efficace derivato da un software commerciale di grande valenza come la maggioranza degli strumenti enterprise presenta, ha degli svantaggi. Infatti, per ogni modifica è necessaria la ricompilazione dei sorgenti, un aspetto macchinoso che si contrappone ad una facile configurazione o manutenzione del programma.

```
#include "ndpi_protocols.h"

#ifdef NDPI_PROTOCOL_VNC

static void ndpi_int_vnc_add_connection(struct ndpi_detection_module_struct
                                        *ndpi_struct, struct ndpi_flow_struct *flow)
{
    ndpi_int_add_connection(ndpi_struct, flow, NDPI_PROTOCOL_VNC, NDPI_REAL_PROTOCOL);
}

/*
  return 0 if nothing has been detected
  return 1 if it is a http packet
*/

void ndpi_search_vnc_tcp(struct ndpi_detection_module_struct *ndpi_struct, struct ndpi_flow_struct *flow)
{
    struct ndpi_packet_struct *packet = &flow->packet;

//      struct ndpi_id_struct         *src=ndpi_struct->src;
//      struct ndpi_id_struct         *dst=ndpi_struct->dst;


    if (flow->l4.tcp.vnc_stage == 0) {
        if (packet->payload_packet_len == 12
            && memcmp(packet->payload, "RFB 003.00", 10) == 0 && packet->payload[11] == 0x0a) {
            NDPI_LOG(NDPI_PROTOCOL_POPO, ndpi_struct, NDPI_LOG_DEBUG, "reached vnc stage one\n");
            flow->l4.tcp.vnc_stage = 1 + packet->packet_direction;
            return;
        }
    } else if (flow->l4.tcp.vnc_stage == 2 - packet->packet_direction) {
        if (packet->payload_packet_len == 12
            && memcmp(packet->payload, "RFB 003.00", 10) == 0 && packet->payload[11] == 0x0a) {
            NDPI_LOG(NDPI_PROTOCOL_VNC, ndpi_struct, NDPI_LOG_DEBUG, "found vnc\n");
            ndpi_int_vnc_add_connection(ndpi_struct, flow);
            return;
        }
    }
    NDPI_ADD_PROTOCOL_TO_BITMASK(flow->excluded_protocol_bitmask, NDPI_PROTOCOL_VNC);

}
#endif
```

Figura 3.2 Signature di nDPI, VNC

Il dato principale su cui ogni sistema d'ispezione deve confrontarsi, è il tasso di falsi positivi e falsi negativi. nDPI effettua il tracciamento del traffico in modo PBFS ed è architettato per minimizzare il tasso di falsi positivi. La quasi totalità delle ispezioni, come dichiarato dall'Ipoque, registrano un tasso minore di

$$2^{-64} \sim 5 * 10^{-18} \%$$

per ogni flusso.

Considerando ogni flusso in combinazione con un tempo di fine connessione, detto *idle time-out*, il calcolo è stato eseguito sull'ipotesi di flussi casuali con lunghezza di payload dei pacchetti

compresa tra 0 e 1460 byte. Requisito essenziale per l'obiettivo originario di gestione del traffico del framework, è per l'appunto, ridurre allo zero il tasso di falsi negativi. Sebbene molto efficiente, la libreria ha delle lacune che sono:

- supporto alla frammentazione;
- supporto al protocollo IPv6;
- ottimizzazione delle performance di calcolo (*multithreading*, tabella *hash* per il *connection tracking*);
- ottimizzazione dell'uso di memoria;

Le mancanze sono tuttavia in parte nascoste dall'eccellente possibilità di rilevamento dei protocolli offuscati delle applicazioni come Skype, BitTorrent, eMule, o criptati come HTTP con certificato SSL riuscendo a scovare il relativo servizio web a cui se è connessi (Facebook, Gmail).

INSTALLAZIONE

Prima di dare uno sguardo al suo interno, descriviamo la procedura d'installazione della libreria. Nata da breve, è disponibile soltanto in *subversion* ed è possibile ottenere la versione più recente digitando da linea di comando:

```
svn co https://svn.ntop.org/svn/ntop/trunk/nDPI
```

Esplorando la directory radice, le sottodirectory in cui soffermarsi per comprendere il software sono:

- SRC->INCLUDE: raccoglie tutti i file d'intestazione;
- SRC->LIB->PROTOCOLS: raccoglie tutte le signature;
- EXAMPLE: contiene un esempio di utilizzo della libreria.

La compilazione dei sorgenti è il successivo passo per abilitare nDPI al funzionamento, e avviene tramite i comandi:

```
cd nDPI
./configure –with-pic
make
```

La libreria consente di visualizzare i risultati di un'ispezione solo con la stampa dei risultati su terminale e non esiste, ad oggi, nessuna interfaccia grafica dedicata che permette di interagire con essa. In continuo sviluppo, nDPI si profila ideale per operare in background, come un *service* riservato all'ispezione dei pacchetti, per conto di altre applicazioni. Non a caso, la Ipoque aveva annunciato nel 2010 un contest per l'integrazione del loro OpenDPI in Netfilter. La realizzazione riuscita, ha richiamato di nuovo le attenzioni dopo la pubblicazione di nDPI, ed al momento, sembra essere stato raggiunto lo stesso risultato[32].

IMPLEMENTAZIONE

Seguirà ora una panoramica sull'implementazione della libreria, dando l'opportunità di chiarire il funzionamento del suo nucleo. nDPI usa una struttura globale dove salvare i settaggi e le variabili temporali per l'analisi dei pacchetti definita dall'istruzione:

```
static struct ndpi_detection_module_struct *ndpi_struct = NULL;
```

Dato che ogni sistema operativo ha un *timestamp* diverso, ossia un ordine temporale differente tra un pacchetto e l'altro, nDPI ne ha uno interno, il cui range è compreso tra 1 secondo e 1 millisecondo, ma è espressamente raccomandato scegliere una risoluzione di 1 ms o 10 ms. Il valore, che rappresenta il tempo di risoluzione per secondo, è direttamente introdotto come parametro dichiarando:

```
static u_int32_t detection_tick_resolution = 1000;
```

L'inizializzazione avviene per mezzo di due operazioni. La prima alloca e istanzia la struttura globale con:

```
ndpi_struct =
ndpi_init_detection_module(detection_tick_resolution,
malloc_wrapper, NULL);
```

in cui l'ultimo parametro deve essere posto a NULL in quanto è solo a scopo di debug. Con questa chiamata avviene l'allocazione di memoria, che è realizzata un'unica volta, evitando richieste multiple durante l'elaborazione dei pacchetti per ragioni di stabilità e performance. La seconda, attiva le signature per il rilevamento, appoggiandosi ad una *bitmask* grande abbastanza come il numero di casi da testare disponibili. Se il bit è 1 la signature è attivata, se 0 è disattivata. La bitmask è definita in questo modo:

```
NDPI_PROTOCOL_BITMASK all;
```

e deve essere settata per l'attivazione di tutte le signature:

```
NDPI_BITMASK_SET_ALL(all);
```

mentre l'abilitazione definitiva avviene con la chiamata:

```
ndpi_set_protocol_detection_bitmask2(ndpi_struct, &all);
```

Dopo una corretta inizializzazione, l'elaborazione di ogni pacchetto è avviata con la chiamata ciclica dell'istruzione:

```
unsigned int protocol;

protocol =
ndpi_detection_process_packet(ndpi_struct, ndpi_flow, (uint8_t *)
iph, ipsize, time, src, dst);
```

che restituisce un ID che corrisponde al un nome di un protocollo o applicazione nel caso ci sia stata corretta identificazione o altrimenti, al caso base *unknown*. Per l'identificazione è ovviamente necessario accedere al pacchetto, con il bisogno di alcuni requisiti:

- puntatore all'header dell'IP;
- lunghezza accessibile dall'inizio del pacchetto, cioè dall'header IP;
- timestamp del pacchetto.

La lunghezza accessibile del pacchetto è indispensabile per evitare errori di lettura durante l'ispezione, facendo affidamento sulla lunghezza totale dell'IP. Le dichiarazioni necessarie sono:

```
unsigned char *packet; // pointer to the packet at Layer 3 (IP)

unsigned short int packet_length;

// number of bytes which can be accessed from the packet pointer

unsigned int timestamp; // arrival timestamp of the packet
```

Lo stato di ogni flusso TCP e UDP, è mantenuto in variabili interne e l'occupazione di memoria per il buffer di ogni stato è fissa e dipendente dal numero dei protocolli compilati. Nel codice la variabile seguente è usata per il tracciamento del traffico o connection tracking:

```
void *flow;

// pointer to the final state machine of the connection
```

L'allocazione necessaria è il valore di ritorno della funzione:

```
u_int32_t ndpi_detection_get_sizeof_ndpi_flow_struct(void);
```

Il puntatore non deve avere un riferimento nullo se il pacchetto contiene TCP o UDP ma è accettato quando, ad esempio, è impossibile ricavare informazioni dalla connessione. Altro dato importante per l'analisi è lo stato degli indirizzi IP, anche se facoltativo, poiché questi possono essere differenti in base alla mappatura della rete. Infatti, in uno schema *access gateway* lo

stesso indirizzo IP è sia il mittente che il destinatario quindi sorgente e destinazione sono noti. Se lo schema è *non access gateway* gli indirizzi IP saranno certamente differenti quindi è utile mettere a disposizione due variabili:

```
void *src;
// pointer to the final state machine of the sender subscriber
void *dst;
// pointer to the final state machine of the dest subscriber
```

L'allocazione necessaria di entrambe è il valore di ritorno della funzione:

```
u_int32_t ndpi_detection_get_sizeof_ndpi_id_struct(void);
```

Ad analisi dei pacchetti completata o interrotta, è indispensabile la chiamata della funzione che libera la memoria dalle strutture dati precaricate:

```
ndpi_exit_detection_module(ndpi_struct, free_wrapper);
```

Categoria	Protocollo/Applicazione
Standard	HTTP, DirectDownloadLink, POP, SMTP, IMAP, FTP, TFTP, AFP, BGP, DHCP, DHCPv6, DNS, EGP, ICMP, ICMPv6, IGMP, NFS, NTP, OSPF, PCAnywhere, MySQL, PostgresSQL, MSSQL, TDS, RDP, SMB, SNMP, SSDP, STUN, SCTP, Telnet, USENET, VNC, IPP, MDNS, NETBIOS, XDMCP, RADIUS, SYSLOG, LDAP, i23V5, HTTP Application Activesync, Kerberos, RADIUS, DCE/RPC, NetFlow/IPFIX, sFlow, HTTP Connect (SSL over HTTP), HTTP Proxy, Citrix, CitrixOnline/GotoMeeting, Webex, Apple (iMessage, FaceTime, iCloud, iTunes ecc.), Facebook, Twitter, Dropbox, Gmail, Google Maps, YouTube, Google, WhatsApp, Viber
P2P	BitTorrent, eDonkey , KaZaa/Fasttrack, Gnutella, WinMX, DirectConnect, AppleJuice, Soulseek, XDCC, Filetopia, MANOLITO, iMESH, PANDO, Socrates, OpenFT, StealthNet, Aimini, Thunder/Webthunder
VoIP	Skype, SIP, IAX, RTP, MGCP, Truphone
Instant Messaging	Yahoo, Oscar, IRC, Jabber, GaduGadu, MSN, Popo, QQ, Meebo
Streaming	ORB, Flash, OGG, MMS, MPEG, AVI, Quick Time, Joost, WindowsMedia, RealMedia, TVAnts, SOPCast, TVUPlayer, PPStream, PPLive, QQLive, Zattoo, VeohTV, OFF, Feidian, Ececast, Kontiki, Move, RTSP, SHOUTcast , Icecast, Netflix
Gaming	Warcraft3, Halflife2, Steam, Xbox, Quake, Second Life, Battlefield, Armagetron, CrossFire, Dofus, Fiesta, Florensia, Guildwars, World of Kung Fu, MapleStory
Tunnel	IPsec, GRE, SSL, SSH, IP in IP, PPTP

Tabella 3.1 Protocolli e applicazioni di nDPI

3.2.2 PF_RING

Una difetto che si riscontra nelle applicazioni di acquisizione del traffico di rete è l'elevata capacità di catturare i pacchetti. Sotto questa prospettiva è stato creato PF_RING[33], un *socket* ottimizzato per innalzare le prestazioni di lettura delle interfacce di rete. PF_RING è una libreria progettata sul sistema operativo GNU/Linux per l'acquisizione di pacchetti ad alta velocità, particolarmente utile quando c'è bisogno di gestire molti pacchetti al secondo, abbinando una cattura veloce con una ottimizzazione dei cicli di operazione della CPU.

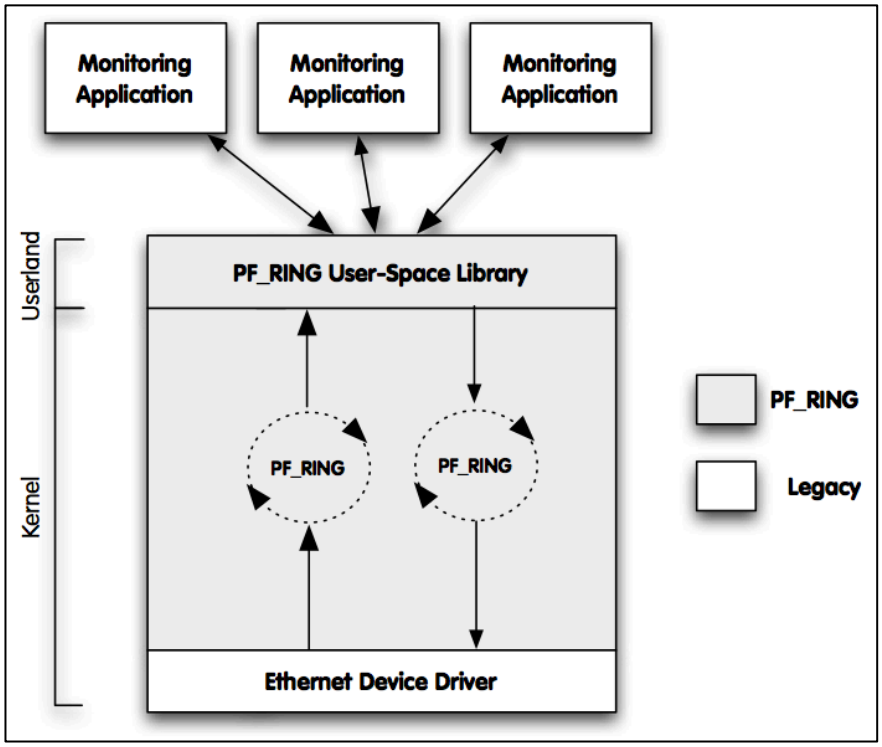

Figura 3.3 Architettura di PF_RING

La Figura 3.3 illustra l'architettura software di PF_RING. Le principali parti sono:

- la SDK a livello userspace che fornisce la libreria di sviluppo per le applicazioni;
- il modulo kernel di accelerazione che fornisce il passaggio a dei pacchetti nel buffer PF_RING;
- driver avanzati per la cattura dei pacchetti senza l'uso delle strutture standard del sistema operativo.

La libreria implementa un nuovo tipo di socket chiamato appunto PF_RING che permette di migliorare le prestazioni ottenute con il *device polling* standard di GNU/Linux, cioè con NAPI, che non riesce a prelevare tutto il traffico se i pacchetti sono di piccola dimensione. La sorgente da cui provengono i pacchetti può essere associata a diversi tipi d'input:

- una interfaccia di rete fisica;
- una coda di ricezione (possibile solo con schede di rete multi-coda);
- una interfaccia di rete virtuale.

I pacchetti una volta catturati sono copiati su un buffer circolare, allocato staticamente al momento della creazione, poi esportato tramite *memory map* nello spazio utente e messi a disposizione dell'applicazione.

Packet Size (bytes)	**GNU/Linux 2.4.23/ RT_IRQ NAPI + PF_RING (packet capture)**
64	550.789 pps (~ 202 Mbit)
512	213.548 pps (~ 850 Mbit)
1500	81.616 pps (~ 970 Mbit)

Tabella 3.2 Prestazioni di PF_RING

La gestione della memoria, come ben chiaro, è il fulcro dell'architettura, poiché dopo il prelevamento di un pacchetto dall'anello, lo stesso spazio è usato per ospitare il successivo, eliminando gli accodamenti nelle strutture dati nel kernel, evitando l'*overhead* dovute alle *system call*, introducendo l'apertura di diversi socket per ogni applicazione, il tutto in un minor tempo di gestione delle interruzioni, che si traduce in maggior uso della CPU nell'elaborazione del traffico. La Tabella 3.2 mostra i risultati ottenuti su un PC con CPU Pentium 4 e scheda di rete Intel a 1 Gbps, laddove limiti, siano dati solo dalle risorse della CPU, soprattutto con i pacchetti di piccola dimensione. Tuttavia, le reti Gbit utilizzano *frame* >= 9.000 byte, detti *jumbo*, quindi si può affermare che è possibile analizzare il traffico a velocità di collegamento.

PF_RING supporta il filtraggio dei pacchetti per via programmatica come i filtri BPF, ed altri introdotti con la libreria stessa, denominati *Wildcard* e *Precise Filters*, tutti inglobati nel modulo kernel e offrendo così una vasta scelta allo sviluppatore. Oltre al filtraggio programmatico è attivo il supporto a schede di rete dedicate all'hardware packet *filtering* come i *chipset Intel 82599* oppure *Silicom Redirector NICs*. I filtri, in generale, specificano un azione da intraprendere quando un pacchetto è in transito che può corrispondere al passaggio/non passaggio dello stesso o l'inoltro. Quest'ultima regola permette di trasferire il pacchetto da un'interfaccia di rete a un'altra, differente da quella dove il pacchetto è stato ricevuto, un'azione gestita direttamente dal modulo kernel senza nessuna elaborazione aggiuntiva. Rispetto ai loro complementari filtri software hanno due grandi vantaggi:

- consentire all'utente di redirigere i flussi direttamente in specifici core della CPU, con migliore gestione della cache;
- consentire l'eliminazione dei pacchetti con impatto zero sulla CPU, evitando qualsiasi elaborazione software.

L'aumento delle prestazioni di cattura sono ottenute anche con il contributo di due meccanismi definiti clustering e *balancing*. Diverse applicazioni, possono sfruttare questa caratteristica, dividendosi l'insieme dei pacchetti da gestire in porzioni, una per ogni cluster. Ogni socket sarà associato a uno specifico cluster ID che gestirà solo una parte dei carico dei pacchetti. L'assegnazione dei pacchetti può avvenire in due modi di bilanciamento. Quello di default è basato sul flusso, mentre l'altro segue lo *scheduling round-robin*, inviando ad ogni cluster la stessa quantità di pacchetti senza alcuna garanzia che quelli appartenenti allo stesso flusso siano processati dallo stesso cluster.

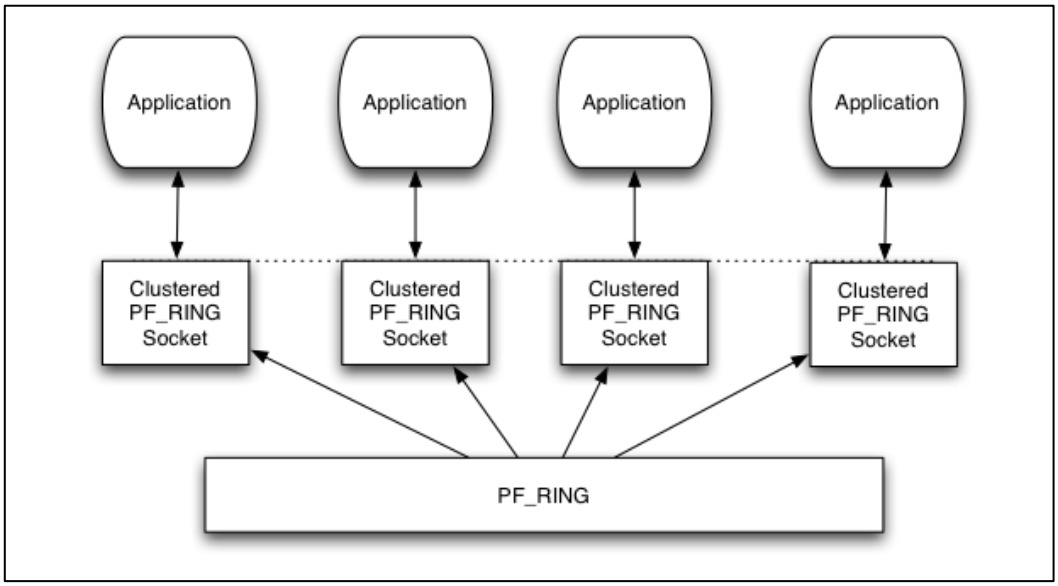

Figura 3.4 Clustering dei pacchetti

La scelta dal punto di vista dell'efficienza è meglio rappresentata dal secondo metodo, in quanto, nel caso d'intenso flusso causato da una specifica applicazione il carico risulterebbe comunque ben distribuito.

Come accennato, PF_RING può operare su driver standard oppure su driver specializzati dell'interfaccia di rete, fermo restando lo stesso modulo kernel, a guadagno di funzionalità e prestazioni diverse. Distinguiamo tre categorie[34, 35]:

- PF_RING-aware driver;
- TNAPI;
- DNA.

I PF_RING-aware driver migliorano la cattura dei pacchetti spingendoli direttamente nel modulo kernel, raggirando il NAPI polling. Le opzioni permesse sono:

transparent_mode 0: default. Uso del solo NAPI.

transparent_mode 1: i pacchetti sono copiati in PF_RING passando per il NAPI;

transparent_mode 2: i pacchetti sono copiati direttamente in PF_RING.

I driver TNAPI o meglio threaded NAPI, ad oggi deprecati, spingono i pacchetti in maniera molto efficiente nella memoria PF_RING grazie all'ausilio di thread a livello kernel attivati dal driver stesso. Essi sono stati sviluppati solo per la cattura dei pacchetti e non viceversa per la trasmissione, con lo sfruttamento della tecnologia RSS separando i flussi in base all'appartenenza e d'inserirli in code di ricezione multiple. Quindi per ogni coda è

assegnato un kernel thread, permettendo di sfruttare le architetture multicore, per eseguire il polling in parallelo.

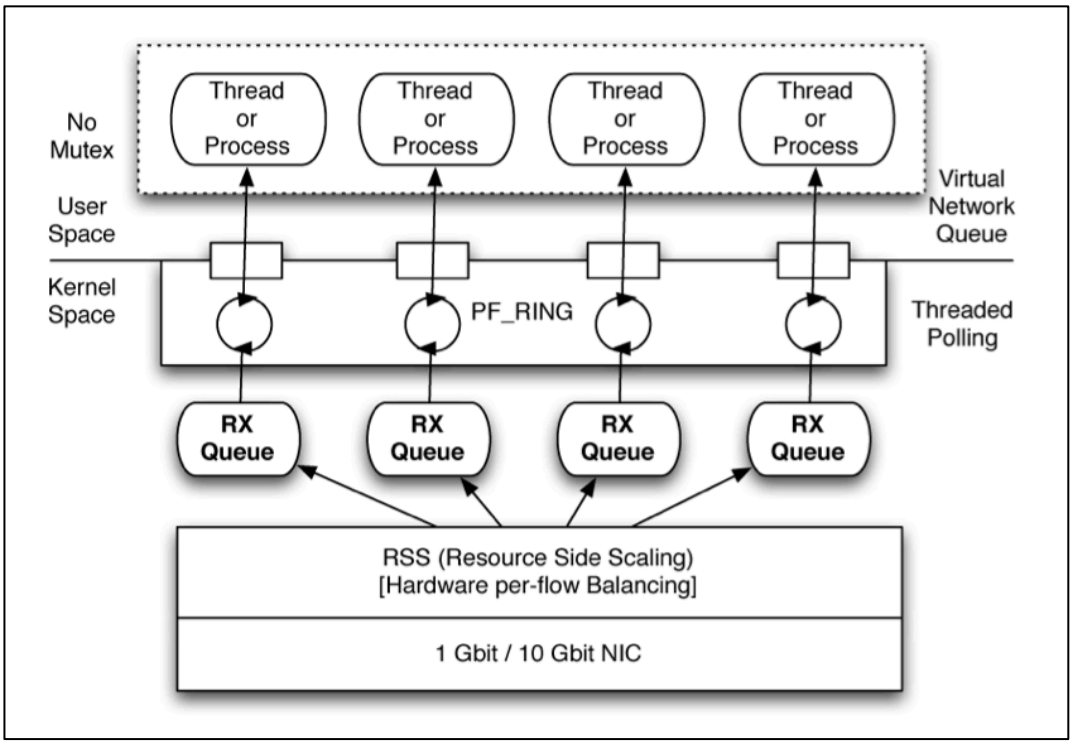

Figura 3.5 TNAPI driver

I driver DNA o *Direct Nic Access* sono la soluzione più quotata, perchè garantiscono la massima velocità di cattura dei pacchetti ed un uso della CPU per il trasferimento dall'interfaccia di rete pari a zero. I pacchetti sono direttamente prelevati dall'adattatore attraverso il NIC NPU, scavalcando il GNU/Linux NAPI, nel così detto modo "zero-*copy*". Con i driver DNA sono supportate le operazioni di trasmissione e ricezione simultanee, anche in alcune schede di rete non sia possibile, mentre lo scavalcamento del NAPI polling toglie qualche funzionalità come il packet filtering (BPF e i filtri PF_RING) e l'inoltro dei pacchetti tra diverse interfacce. Per

questo motivo è stata introdotta la libreria Libzero[36] che comunichi con la libreria DNA e permetta in modalità zero-copy:

- la distribuzione dei pacchetti tra thread e processi distinti;
- flessibilità e configurabilità sullo smistamento dei pacchetti;
- packet filtering;
- inoltro efficiente dei pacchetti tra le interfacce di rete.

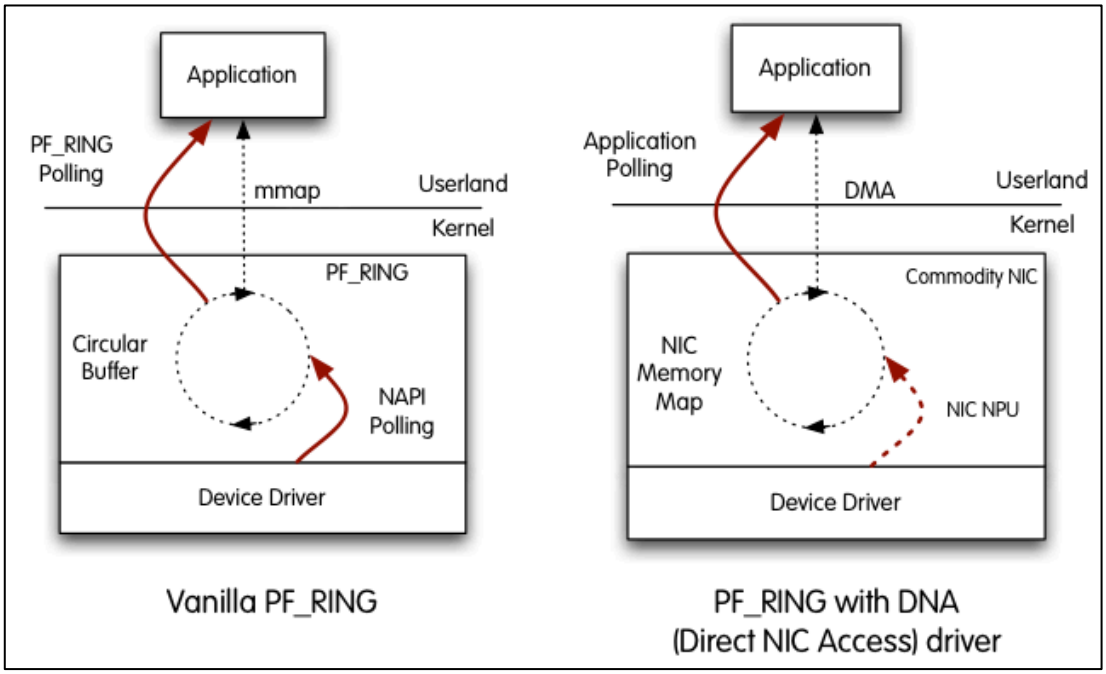

Figura 3.6 DNA driver

I principali componenti della Libzero sono il DNA cluster e il DNA *bouncer.* Il DNA cluster, simile al PF_RING cluster, applica il clustering dei pacchetti, cosicché tutte le applicazioni che implementano lo stesso cluster ID, possano condividere i pacchetti in entrata con bilanciamento delle code. Esso aggiunge la capacità di trasmissione, filtraggio e anche distribuzione e duplicazione tra diversi thread e processi con limitazioni solo alla presenza di un'unica interfaccia DNA.

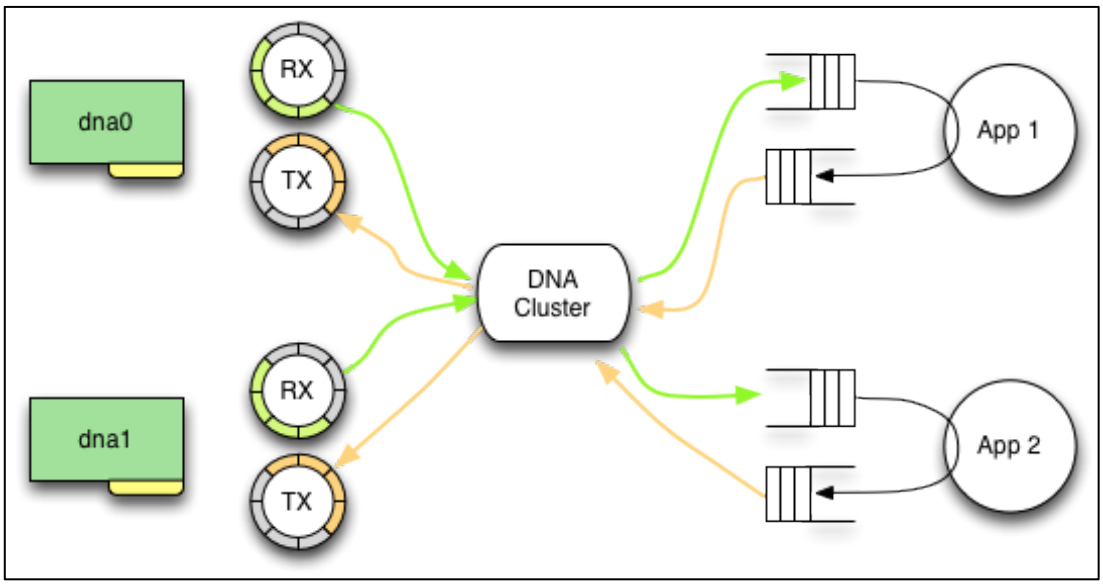

Figura 3.7 DNA clustering

Il DNA bouncer si occupa di passare pacchetti tra due interfacce DNA in zero-copy con facoltà di decidere all'utente quali possono essere inoltrati o meno. Il trasferimento è realizzato in maniera mono direzionale, quindi, nel caso si volesse realizzare una comunicazione *full-duplex,* è necessario istanziare un doppio bouncer.

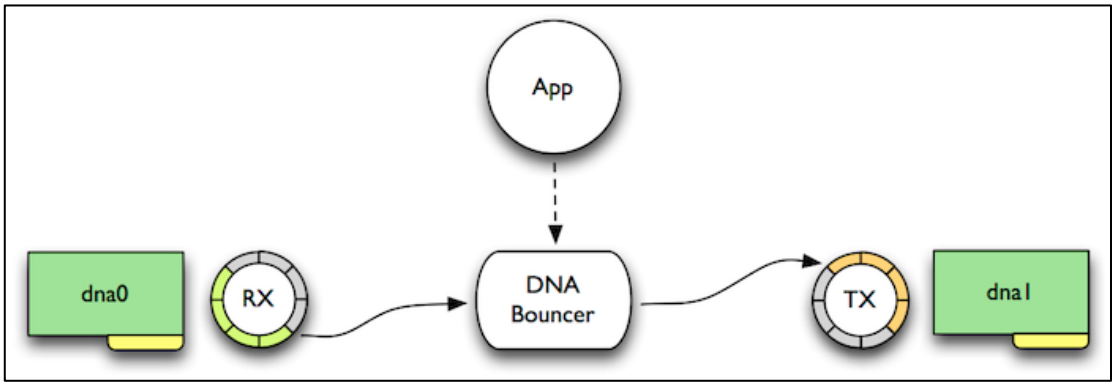

Figura 3.8 DNA bouncer

La nuova tipologia di socket PF_RING, dona miglioramenti anche nel mondo della virtualizzazione con vPF_RING[37]. Estensione della libreria originale, consente il passaggio diretto dal NIC della macchina host all'applicazione della macchina virtuale senza il coinvolgimento dell'*hypervisor*. Questo approccio, che rimuove i colli di bottiglia, aiutato dal supporto dei moderni processori alla virtualizzazione con l'esecuzione del codice direttamente dalla macchina virtuale, avvicina le prestazioni a quelle di un sistema nativo. Inoltre, la capacità di pre-gestire il filtraggio, impedisce che alcuni pacchetti siano eliminati nella macchina virtualizzata sprecando preziosi cicli di CPU. Infine, le applicazioni in esecuzione su diverse macchine virtuali possono essere grado di analizzare lo stesso traffico o code diverse da un'unica cattura di pacchetti.

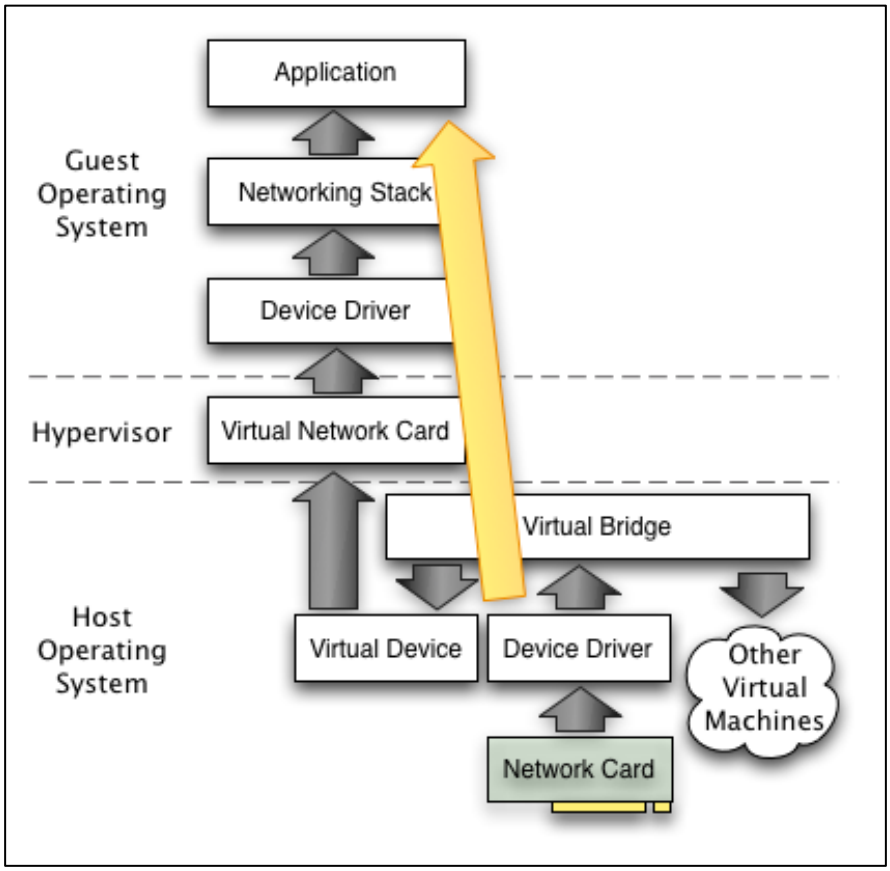

Figura 3.9 vPF_RING hypervisor bypass

Capitolo 4

Dimostrazione della DPI applicata

4.1 Il caso d'uso

In possesso delle conoscenze e degli strumenti adatti sarà descritto in questo capitolo, un caso d'uso di analisi del traffico. Per l'esperimento si è scelto di applicare la deep packet inspection alla pirateria satellitare, in particolare al fenomeno del card-sharing. Questo, ha comportato un certo criterio di lavoro poiché è stato necessario, prima di tutto recuperare informazioni sui programmi e i dispositivi adatti per ricostruire una reale rete di card-sharing, in seguito con l'aiuto di alcuni software di cattura e intercettazione dei pacchetti si è studiato il funzionamento e le caratteristiche del protocollo di comunicazione ed infine è stata implementata una valida signature al sistema di ispezione dei pacchetti prescelto. Nel dettaglio la prova è stata condotta con:

- due decoder satellitari, cloni del Dreambox 500 con porta ethernet e *firmware* GNU/Linux EDG-Nemesis 4.4;
- notebook Macbook Pro con sistema operativo GNU/Linux Ubuntu 12.04;
- il software Ettercap per l'intercettazione tra i due decoder;
- il software Wireshark per la cattura dei pacchetti;
- la libreria nDPI per l'ispezione dei pacchetti.

Essendo alcuni contenuti trattati a carattere illegale, in questa tesi sono considerati solo a scopo didattico.

4.2 Il card-sharing

Il card-sharing[38] è una tecnica di condivisione dell'abbonamento ai canali satellitari in pay-per-view. La pirateria satellitare ha avuto un piccolo arresto dopo la decriptazione delle chiavi attraverso schede programmabili, contrastato dalla complessità delle nuove codifiche adottate dagli operatori televisivi satellitari. L'invenzione di simulatori CAM attraverso reverse engineering e dei decoder dotati d'interfacce di connessione di rete e firmware su misura, si è potuto cominciare a praticare questo nuovo tipo di frode che attualmente è in larga diffusione. Il suo funzionamento è basato sul paradigma client-server, poiché il dispositivo client invierà la richiesta della chiave per la decifratura del canale criptato, mentre il server, svolgerà la funzione di lettura della card, decriptazione degli ECM e rilascio della chiave.

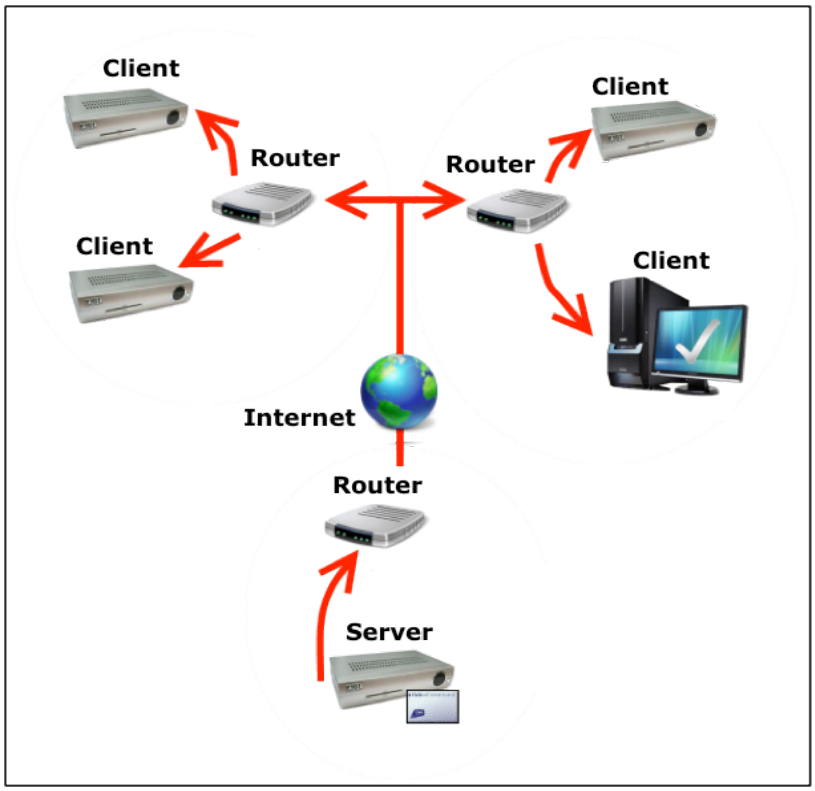

Figura 4.1 Card-sharing

Le chiavi, dette anche Costant Word (CW), si rinnovano dinamicamente ogni pochi secondi e costituiscono in pochi Kbyte di dati, che possono agevolmente essere spedite attraverso la rete, consentendo di creare un sistema di condivisione geograficamente molto esteso. Il ruolo principale è svolto dai dispositivi che si occupano di generare le chiavi avendo accesso alle card. Normalmente, essi sono decoder con sistema operativo GNU/Linux o meglio firmware, oppure, poiché il calcolo delle chiavi è abbastanza esoso di risorse, sono PC, permettendo un numero di client connessi più numeroso, in quanto il sovraccarico di richieste porterebbe ad un evitabile disservizio, ossia ad una visione dei canali scattosa. L'elemento software che legge la card emulando un modulo ad accesso condizionato è chiamato *softcam* (ad es. Cccam, OScam, Mgcamd, GBox, Sbox) e alcuni di essi oltre al loro compito fondamentale di generare le chiavi, offrono funzionalità avanzate come *load balancer* per ripartire il calcolo delle chiavi su più card, cache *exchange* per lo scambio delle chiavi in memoria di card diverse, auto update per l'aggiornamento degli EMM sulle card, anche remote. Le softcam sono sviluppate per sistemi GNU/Linux con compatibilità su diverse architetture hardware (ad es. *powerpc*, *mipsel*, *x86*) e sono considerate non legali da alcuni operatori PPV, che obbligano i loro clienti ad usare apparecchi proprietari. Non tutte le softcam sono a codice aperto, quindi non è possibile conoscere in che modo è costruita la loro progettazione e se includono funzionalità sospette (ad es. *backdoor*). Inoltre, per non farsi mancare nulla, i dati scambiati tra un client e un server sono spesso codificati impedendone un'intercettazione con finalità di semplice furto di credenziali d'accesso alle connessioni sui server o controlli da parte delle forze di polizia.

4.3 Dalla teoria alla pratica

COSTRUZIONE DELLA RETE DI CARD-SHARING

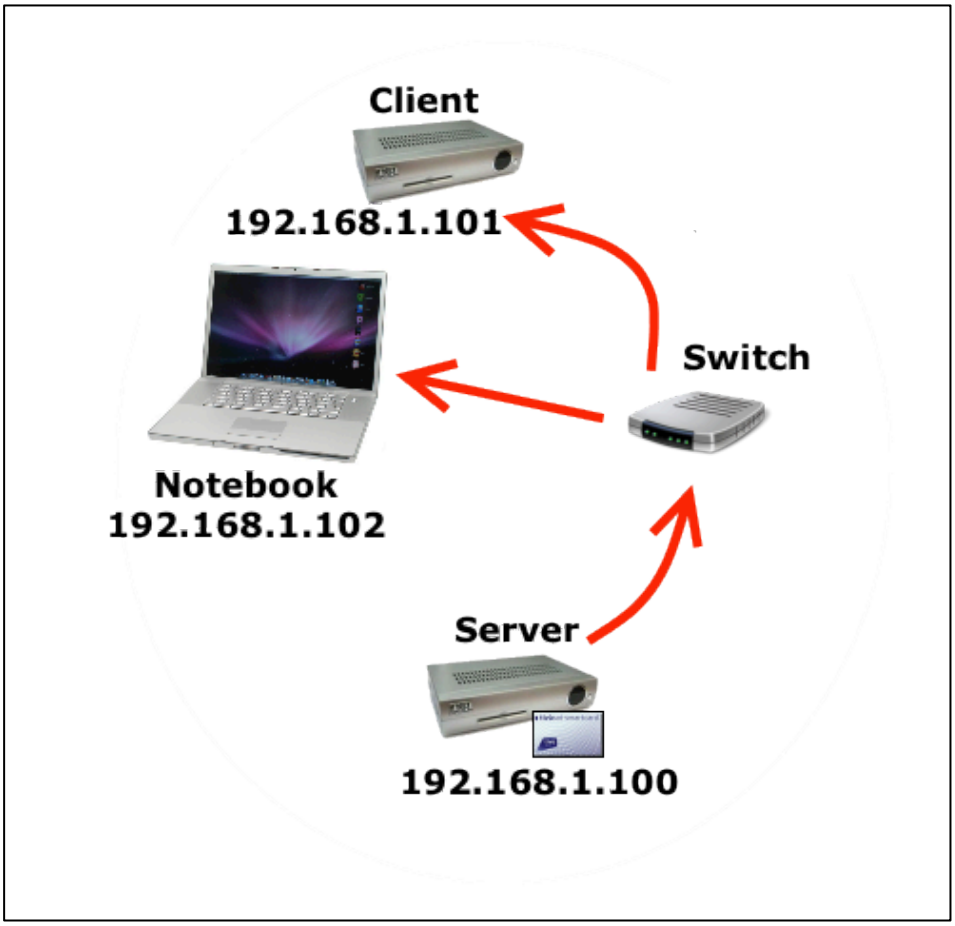

Figura 4.2 Costruzione della rete di card-sharing

La realizzazione della rete è la prima fase per studiare il funzionamento delle softcam. Ritenendo la connessione ad internet non dispensabile, i dispositivi sono stati collegati ad uno switch, visto l'indifferenza del modo di operare in ambito LAN e WAN. La softcam utilizzata è stata CCcam 2.2.1, per via della sua diffusione, grazie alla semplicità d'installazione e configurazione. La card inserita nel decoder server è stata quella della piattaforma satellitare italiana Tivù Sat, scelta in modo arbitrario perché card diverse non alterano le caratteristiche del traffico dati, strettamente legato alla softcam.

INTERCETTAZIONE E CATTURA DEI PACCHETTI

Nella seconda fase è stata praticata l'intercettazione dei pacchetti tra i decoder, attuando un attacco *main in the middle* per mezzo del software Ettercap e la cattura ed analisi degli stessi con Wireshark. Sono state condotte diverse prove per avere un quadro chiaro dei vari processi che la softcam esegue. Dalle prime osservazioni si è intuito uno schema di comunicazione caratterizzato da payload eterogenei se non nella loro lunghezza. Infatti, l'assenza di caratteri ricorrenti, ha fatto intuire e dopo alcune successive verifiche costatare, che i dati fossero criptati oppure offuscati. A causa dell'impossibilità di conoscere il tipo di codifica o di offuscamento usata dal software poiché *closed source*, è stato escluso un metodo di analisi su comparazione di stringhe, a favore di un'analisi di tipo numerica e comportamentale. Partendo dall'ipotesi di una richiesta di chiavi costante e ripetitiva, è stato individuato quando e come questo avviene. La softcam, che si appoggia sul protocollo TCP, durante questo processo avvia una conversazione di pacchetti con lunghezza di payload fissa e ad intervalli regolari. Questo comportamento è certamente il punto fisso dell'applicazione ma può essere anticipato da altri diversi scambi di pacchetti caratterizzato da uno schema fisso, con lunghezze di payload non variabili. Altre attenzioni sono state poste sulle porte e i *flag* utilizzati sul TCP. Fissa a prescindere la porta server, il client dal *three-way handshake* in poi, ad ogni richiesta utilizza la medesima porta, che sta a significare che lo stato della connessione non viene interrotto se non per time-out. Inoltre, per entrambi le direzioni, il passaggio all'applicazione dei dati è forzato dall'attivazione del flag PSH. L'analisi appena descritta, è riuscita a rendere evidente le caratteristiche per l'identificazione

dell'applicazione, praticando un'accurata verifica del processo di scambio dati.

CLIENT PORTA	TCP FLAG	DIREZIONE PACCHETTO	PAYLOAD (bytes)	SERVER PORTA
schema 1				
1027	PSH, ACK	←------	16	12000
1027	PSH, ACK	------→	20	12000
1027	PSH, ACK	------→	26	12000
1027	PSH, ACK	←------	20	12000
1027	PSH, ACK	------→	97	12000
1027	PSH, ACK	←------	4	12000
1027	PSH, ACK	←------	110	12000
schema 2				
1027	PSH, ACK	------→	157	12000
1027	PSH, ACK	←------	20	12000
1027	PSH, ACK	------→	157	12000
1027	PSH, ACK	←------	20	12000
1027	PSH, ACK	------→	157	12000
1027	PSH, ACK	←------	20	12000
…	…	…	…	…
…	…	…	…	…
1027	PSH, ACK	------→	157	12000
1027	PSH, ACK	←------	20	12000
…	…	…	…	…
…	…	…	…	…
sequenza ripetuta ogni 10 secondi…				

Tabella 4.1 Analisi del flusso dei pacchetti

La Tabella 4.1 riporta l'analisi del flusso su cui è stata costruita la signature. Va detto che non sono stati considerati

alcuni pacchetti con il solo flag ACK tra gli scambi, perché ininfluenti per l'identificazione del protocollo. Inoltre, la porta di servizio del client è solo indicativa, mentre quella del server è stata scelta nella configurazione della softcam. La suddivisione in due schemi, vuole significare che il flusso può presentarsi in due forme:

- schema 1 + schema 2;
- solo schema 2.

Questo dipende dal fatto che prima della sequenza ciclica di richiesta e rilascio della chiave (schema 2) può avvenire uno scambio di dati (schema 1) quando il client è avviato da spento e dopo il time-out del TCP, ma purtroppo non si è riuscito a capire bene cosa rappresenti per causa della criptazione dei payload, ma potrebbe indicare un settaggio iniziale necessario alla comunicazione.

IMPLEMENTAZIONE DELLA SIGNATURE

La terza fase di questo caso d'uso è stata la traduzione dell'analisi del flusso dei pacchetti in linguaggio C, per l'implementazione della signature in nDPI. Saranno commentate le parti salienti mentre il suo codice completo è listato in appendice A.

La signature, tiene conto di alcuni parametri, cioè le porte di servizio, lo schema seguito dal flusso, la direzione del pacchetto, il timestamp tra una richiesta di chiave e l'altra, che devono essere sempre disponibili, quindi sono state definite le seguenti variabili globali:

```
u_int8_t cccam_scheme, cccam_direction;

u_int16_t port_client, port_server;

u_int32_t cccam_ts;
```

Poiché il flusso si potrebbe presentare in due forme diverse, uno *statement* iniziale di controllo permette di capire con quale delle due si abbia a che fare:

```
…
if(flow->l4.tcp.cccam_scheme == 0) //Check scheme
{
      //check if scheme 1
      if(packet->payload_packet_len == 16
      && (flow->l4.tcp.cccam_direction = 1
      + packet->packet_direction) == 2 && packet->tcp->psh)
      {
      …
      }
      //Check if scheme 2
      else if(packet->payload_packet_len == 157
      && (flow->l4.tcp.cccam_direction = 1
      + packet->packet_direction) == 1 && packet->tcp->psh)
      {
      …
      }
}
…
```

Nelle due istruzioni condizionali interne, sono discriminanti i valori della lunghezza del payload, della direzione del pacchetto e del flag PSH. Da questo punto, se ci trova con un flusso nella prima forma, ossia schema 1 + schema 2, viene eseguita dapprima la sequenza dello schema 1 e poi lo schema 2, altrimenti, in caso di seconda forma, viene eseguita solo la sequenza dello schema 2. Questo blocco è un passaggio obbligatorio per ogni pacchetto sotto

ispezione. Infatti, lo schema 1, se si presenta, lo fa una sola volta, dopodiché le successive transazioni faranno riferimento solo allo schema 2. La sequenza dello schema 1 consiste in una cascata di istruzioni condizionali costruiti sulla traccia dei dati riportati in Tabella 4.1.

```
…
//Check sequence scheme 1
if(flow->l4.tcp.cccam_scheme == 1
      && packet->payload_packet_len == 20
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 2
      && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 11
      && packet->payload_packet_len == 26
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 2
      && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 12
      && packet->payload_packet_len == 20
      && (flow->l4.tcp.cccam_direction = 2
```

```
      + packet->packet_direction) == 3
      && packet->tcp->psh
      && flow->l4.tcp.port_server == packet->tcp->source
      && flow->l4.tcp.port_client == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 13
      && packet->payload_packet_len == 97
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 3
      && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 14
      && packet->payload_packet_len == 4
      && (flow->l4.tcp.cccam_direction = 3
      + packet->packet_direction) == 4
      && packet->tcp->psh
      && flow->l4.tcp.port_server == packet->tcp->source
      && flow->l4.tcp.port_client == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 15
      && packet->payload_packet_len == 110
      && (flow->l4.tcp.cccam_direction = 4
```

```
      + packet->packet_direction) == 5
      && packet->tcp->psh
      && flow->l4.tcp.port_server == packet->tcp->source
      && flow->l4.tcp.port_client == packet->tcp->dest)
…
```

I valori di *cccam_scheme,* aggiornata ogni qual volta le condizioni siano vere, stanno a indicare con la cifra delle decine lo schema e con la cifra delle unità lo step. Lo schema 2, è strutturato in egual modo:

```
…
//Check sequence scheme 2
if(flow->l4.tcp.cccam_scheme == 2
      && packet->payload_packet_len == 157
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 5
      && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
{
…
}
if(flow->l4.tcp.cccam_scheme == 21
      && packet->payload_packet_len == 20
      && (flow->l4.tcp.cccam_direction = 5
      + packet->packet_direction) == 6
      && packet->tcp->psh
      && flow->l4.tcp.port_server == packet->tcp->source
      && flow->l4.tcp.port_client == packet->tcp->dest)
...
```

In particolare, nella seconda istruzione di controllo è verificata che ci sia già stata una precedente esecuzione dello schema 2, per dare robustezza all'analisi. Poiché esso ricorre all'incirca ogni 10 secondi, per assicurare che il pattern corrisponda, un controllo posto nel blocco iniziale, permette di verificarlo:

```
…
//Actual time minus last time
flow>l4.tcp.time =
(int)((packet->tick_timestamp - flow>l4.tcp.cccam_ts)/1000);
//Timestamp difference must be 9,10 sec
if( flow>l4.tcp.time == 9 || flow>l4.tcp.time == 10)
{
…
}
…
```

Infine, l'identificazione è confermata dall'istruzione:

```
ndpi_int_cccam_add_connection(ndpi_struct, flow);
```

Essa aggiunge la nostra signature alla pila dei protocolli rilevati. Il test effettuato su diverse catture di pacchetti memorizzate in file .pcap, ha evidenziato l'effettiva validità della signature.

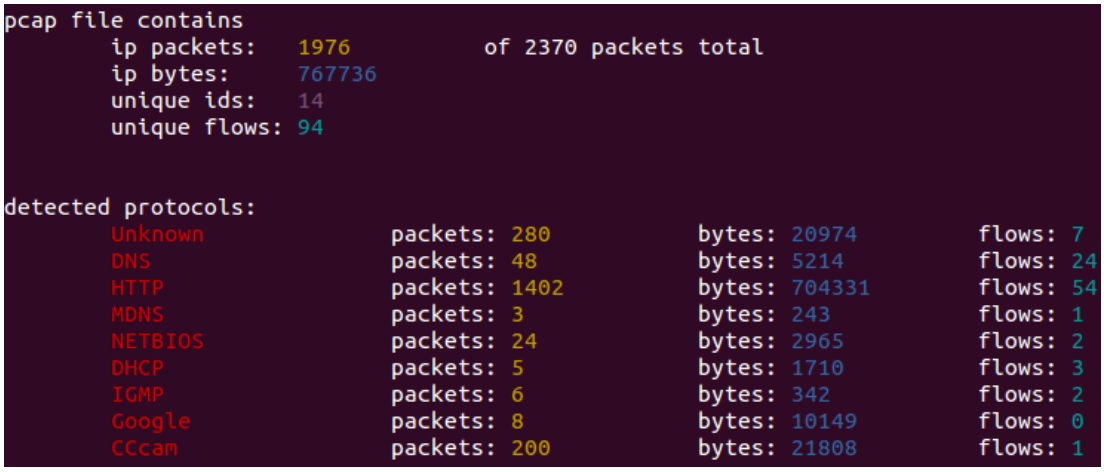

Figura 4.3 Test della signature

Il risultato ottenuto (Figura 4.3) è viziato da alcuni bug minori, dovuti dalla libreria nDPI e da alcuni probabili miglioramenti da effettuare sulla signature stessa. Ci si riferisce, in questo esempio, alla possibilità di pacchetti non riportati nel flusso di riferimento della nostra applicazione e segnalati come sconosciuti. Questa considerazione non comporta una scorretta identificazione, ma fa notare un'imperfetta e migliorabile gestione dell'elaborazione.

Conclusioni

Il campo del networking ha avuto molti progressi negli ultimi tempi. Dal punto di vista sociale, le utenze crescono sempre di più e le infrastrutture di comunicazione devono reggere questo confronto. Su questo sfondo, il punto di vista della sicurezza diventa sempre più delicato e anche se, i provvedimenti esistono da qualche tempo, le novità sono poche, ma incoraggiate dai continui sviluppi sulle soluzioni già trovate.

Come si è cercato di comprendere con questa tesi, la deep packet inspection risponde a tutto questo. Essa, si può intendere come un insieme di tecniche già note, unite per creare uno strumento completo, che offra grandi prestazioni e flessibilità, seppur molto invasiva. Certamente, si tratta di una soluzione complessa e costosa, perché il ciclo di operazioni che i dispositivi devono sostenere non è modesto e la loro installazione e amministrazione richiede personale competente. Perciò, la sua adozione avviene per lo più in ambienti molto specializzati, come i fornitori della connessione di rete, che ne traggono vantaggio soprattutto per la gestione della qualità di servizio del traffico internet. In altre situazioni, o meglio all'interno di aziende, dove la tutela dei dati è vitale per l'attività stessa, le avanzate opportunità di prevenire le emergenze che la laboriosità criminale può causare, sono un altro lato di forte interesse a favore di questa tecnica.

Tuttavia, esistono delle inadeguatezze, scaturite dal suo principio di operatività basato sui pacchetti. Il metodo, seppur con i diversi approcci di analisi sia risolutivo, mostra impedimento sull'esame di certe strutture dei payload difficilmente identificabili,

come alcuni contenuti Web (ad es. codifica binario-ASCII, compressione, criptazione, offuscamento). Da questo presupposto, alcune recenti soluzioni sulla sicurezza di rete, collaudano un nuovo principio d'ispezione del traffico fondato sui MIME *object*, tecnica battezzata come deep *content* inspection[39].

Il caso d'uso ha dimostrato poi che, anche i sistemi d'ispezione open-source impiegati su elaboratori comuni, riescono a svolgere egregiamente la funzione d'identificazione, seppur con alcune limitazioni, a patto di saper ben costruire una regola di rilevamento affidabile.

Elenco delle figure

Elenco delle tabelle

Acronimi

IP Internet **P**rotocol

DNS Domain **N**ame **S**ystem

DoS Denial **o**f **S**ervice

ISO International **O**rganization for **S**tandardization

OSI Open **S**ystems **I**nterconnection

PKI Public **K**ey **I**nfrastructure

IDS Intrusion **D**etection **S**ystem

IPS Intrusion **P**revention **S**ystem

VPN Virtual **P**rivate **N**etwork

RADIUS Remote **A**uthentication **D**ial-**I**n **U**ser **S**ervice

TACACS Terminal **A**ccess **C**ontroller **A**ccess **C**ontrol **S**ystem

SPI Shallow **P**acket **I**nspection

TCP Transmission **C**ontrol **P**rotocol

MPI Medium **P**acket **I**nspection

ISP Internet **S**ervice **P**rovider

HTTP Hyper**T**ext **T**ransfer **P**rotocol

FTP File **T**ransfer **P**rotocol

DPI Deep **P**acket **I**nspection

P2P Peer **to P**eer

UDP User **D**atagram **P**rotocol

SMTP **S**imple **M**ail **T**ransfer **P**rotocol

SSL **S**ecure **S**ockets **L**ayer

QoS **Q**uality **o**f **S**ervice

DLP **D**ata **L**oss **P**revention

VoIP **V**oice **o**ver **I**nternet **P**rotocol

URL **U**niform **R**esource **L**ocator

EFF **E**lectronic **F**rontier **F**oundation

FISA **F**oreign **I**ntelligence **S**urveillance **A**ct

ECHR **E**uropean **C**onvention on **H**uman **R**ights

UE **U**nione **E**uropea

TV **T**ele**v**isione

SOPA **S**top **O**nline **P**iracy **A**ct

PIPA **P**ROTECT **IP** **A**ct

ACTA **A**nti-**C**ounterfeiting **T**rade **A**greement

ACDT **A**merican **C**enter for **D**emocracy and **T**echnology

HADOPI **H**aute **A**utorité pour la **D**iffusion des **O**euvres et la **P**rotection des droits sur **I**nternet

IM **I**nstant **M**essaging

PPV **P**ay-**P**er-**V**iew

TCAM **T**ernary **C**ontent-**A**ddressable **M**emory

FGPA **F**ield-**P**rogrammable **G**ate **A**rray

CPU **C**entral **P**rocessing **U**nit

EMEA **E**urope, the **M**iddle **E**ast and **A**frica

GNU GPL GNU **G**eneral **P**ublic **L**icense

PACE **P**rotocol and **A**pplication **C**lassification **E**ngine

GNU LGPL GNU **L**esser **G**eneral **P**ublic **L**icense

BPF **B**erkeley **P**acket **F**ilter

ID **Id**entificatore

NAPI **N**ew **A**pplication **P**rogramming **I**nterface

RSS **R**eceive **S**ide **S**caling

NIC **N**etwork **I**nterface **C**ontroller

NPU **N**etwork **P**rocess **U**nit

CAM **C**onditional **A**ccess **M**odule

ECM **E**ntitlement **C**ontrol **M**essage

PC **P**ersonal **C**omputer

EMM **E**ntitlement **M**anagement **M**essage

LAN **L**ocal **A**rea **N**etwork

WAN **W**ide **A**rea **N**etwork

MIME **M**ultipurpose **I**nternet **M**ail **E**xtensions

Appendice A

Listato della signature sul caso d'uso (cap. 4).

```
//CCcam signature

#include "ndpi_protocols.h"

#ifdef NDPI_PROTOCOL_CCCAM

//Add CCcam to detected protocol stack

static void ndpi_int_cccam_add_connection
(struct ndpi_detection_module_struct *ndpi_struct,
struct ndpi_flow_struct *flow)
{
ndpi_int_add_connection(ndpi_struct, flow,
NDPI_PROTOCOL_CCCAM, NDPI_REAL_PROTOCOL);
}

//CCcam deep packet inspection

void ndpi_search_cccam
(struct ndpi_detection_module_struct *ndpi_struct,
struct ndpi_flow_struct *flow)
{
      struct ndpi_packet_struct *packet = &flow->packet;
      if(packet->tcp != NULL && packet->payload_packet_len != 0)
      {
      if(flow->l4.tcp.cccam_scheme == 0) //Check scheme
      {
            //Check if scheme 1

            if(packet->payload_packet_len == 16
            && (flow->l4.tcp.cccam_direction = 1
            + packet->packet_direction) == 2 && packet->tcp->psh)
            {
                  flow->l4.tcp.port_client = packet->tcp->dest;
                  flow->l4.tcp.port_server = packet->tcp->source;
                        flow->l4.tcp.cccam_scheme = 1;
                  return;
            }

            //Check if scheme 2

            else if(packet->payload_packet_len == 157
            && (flow->l4.tcp.cccam_direction = 1
            + packet->packet_direction) == 1 && packet->tcp->psh)
```

```
            {
                  //Check timestamp scheme 2

                  if(flow->l4.tcp.cccam_ts == 0)
                  {
                        flow->l4.tcp.cccam_ts =
                        packet->tick_timestamp;
                        flow->l4.tcp.port_client =
                        packet->tcp->source;
                        flow->l4.tcp.port_server =
                        packet->tcp->dest;
                        flow->l4.tcp.cccam_scheme = 21;
                  }


                  else
                  {
                        //Actual time minus last time

                        flow->l4.tcp.time =
(int)((packet->tick_timestamp - flow->l4.tcp.cccam_ts)/1000);

                        //Timestamp difference must be 9/10 sec

                        if( flow->l4.tcp.time == 9
                        || flow->l4.tcp.time == 10)
                        {
                              flow->l4.tcp.cccam_ts =
                              packet->tick_timestamp;
                              flow->l4.tcp.port_client =
                              packet->tcp->source;
                              flow->l4.tcp.port_server =
                              packet->tcp->dest;
                              flow->l4.tcp.cccam_scheme = 21;
                        }
                  }
                  return;
            }

      }

      //Check sequence scheme 1

      if(flow->l4.tcp.cccam_scheme == 1
      && packet->payload_packet_len == 20
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 2 && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
      {
            flow->l4.tcp.cccam_scheme = 11;
            return;
      }
      if(flow->l4.tcp.cccam_scheme == 11
      && packet->payload_packet_len == 26
      && (flow->l4.tcp.cccam_direction
      + packet->packet_direction) == 2 && packet->tcp->psh
      && flow->l4.tcp.port_client == packet->tcp->source
      && flow->l4.tcp.port_server == packet->tcp->dest)
```

```
	{
		flow->l4.tcp.cccam_scheme = 12;
		return;
	}
	if(flow->l4.tcp.cccam_scheme == 12
	&& packet->payload_packet_len == 20
	&& (flow->l4.tcp.cccam_direction = 2
	+ packet->packet_direction) == 3 && packet->tcp->psh
	&& flow->l4.tcp.port_server == packet->tcp->source
	&& flow->l4.tcp.port_client == packet->tcp->dest)
	{
		flow->l4.tcp.cccam_scheme = 13;
		return;
	}
	if(flow->l4.tcp.cccam_scheme == 13
	&& packet->payload_packet_len == 97
	&& (flow->l4.tcp.cccam_direction
	+ packet->packet_direction) == 3 && packet->tcp->psh
	&& flow->l4.tcp.port_client == packet->tcp->source
	&& flow->l4.tcp.port_server == packet->tcp->dest)
	{
		flow->l4.tcp.cccam_scheme = 14;
		return;
	}
	if(flow->l4.tcp.cccam_scheme == 14
	&& packet->payload_packet_len == 4
	&& (flow->l4.tcp.cccam_direction = 3
	+ packet->packet_direction) == 4 && packet->tcp->psh
	&& flow->l4.tcp.port_server == packet->tcp->source
	&& flow->l4.tcp.port_client == packet->tcp->dest)
	{
		flow->l4.tcp.cccam_scheme = 15;
		return;
	}
	if(flow->l4.tcp.cccam_scheme == 15
	&& packet->payload_packet_len == 110
	&& (flow->l4.tcp.cccam_direction = 4
	+ packet->packet_direction) == 5 && packet->tcp->psh
	&& flow->l4.tcp.port_server == packet->tcp->source
	&& flow->l4.tcp.port_client == packet->tcp->dest)
	{
		flow->l4.tcp.cccam_scheme = 2;
		return;
	}

	//Check sequence scheme 2

	if(flow->l4.tcp.cccam_scheme == 2
	&& packet->payload_packet_len == 157
	&& (flow->l4.tcp.cccam_direction
	+ packet->packet_direction) == 5 && packet->tcp->psh
	&& flow->l4.tcp.port_client == packet->tcp->source
	&& flow->l4.tcp.port_server == packet->tcp->dest)
```

```
      {
            if(flow->l4.tcp.cccam_ts == 0)
            {
                  flow->l4.tcp.cccam_ts = packet->tick_timestamp;
                  flow->l4.tcp.cccam_scheme = 21;
            }
            return;
      }
      if(flow->l4.tcp.cccam_scheme == 21
      && packet->payload_packet_len == 20
      && (flow->l4.tcp.cccam_direction = 5
      + packet->packet_direction) == 6 && packet->tcp->psh
      && flow->l4.tcp.port_server == packet->tcp->source
      && flow->l4.tcp.port_client == packet->tcp->dest)
      {
            flow->l4.tcp.cccam_direction = 0;
            flow->l4.tcp.cccam_scheme = 0;
            if(flow->l4.tcp.time != 0)
            {
                  ndpi_int_cccam_add_connection
                        (ndpi_struct, flow);
            }
      }
      return;
    }

    NDPI_ADD_PROTOCOL_TO_BITMASK
(flow->excluded_protocol_bitmask, NDPI_PROTOCOL_CCCAM);

}
#endif
```

Appendice B

Listato esempio di pattern matching. La signature verifica se il destinatario di una mail sia "patrik.rosini@gmail.com".

```
//Mail Destination signature

#include "ndpi_protocols.h"

#ifdef NDPI_PROTOCOL_MAIL_DEST

static void ndpi_int_mail_dest_add_connection
(struct ndpi_detection_module_struct*ndpi_struct,
struct ndpi_flow_struct *flow)
{
ndpi_int_add_connection(ndpi_struct, flow,
NDPI_PROTOCOL_MAIL_DEST, NDPI_REAL_PROTOCOL);
}
void ndpi_search_mail_dest
(struct ndpi_detection_module_struct*ndpi_struct,
struct ndpi_flow_struct *flow)
{
      struct ndpi_packet_struct *packet = &flow->packet;

      //Check payload packet size

      if(packet->payload_packet_len > 2
      && ntohs(get_u_int16_t(packet->payload,
      packet->payload_packet_len - 2)) == 0x0d0a)
      {

      int a;
      int b;
      int end;

      //patrik.rosini@gmail.com in hexadecimal

      char mail_address[] = { 0x70, 0x61, 0x74, 0x72, 0x69, 0x6b,
                              0x2e, 0x72, 0x6f, 0x73, 0x69, 0x6e,
                              0x69, 0x40, 0x67, 0x6d, 0x61, 0x69,
                              0x6c, 0x2e, 0x63, 0x6f, 0x6d };

            //Check port dest

            if(packet->tcp->dest == ntohs(25))
            {
                  NDPI_PARSE_PACKET_LINE_INFO
                  (ndpi_struct, flow,packet);
```

```
                  for (a = 0; a < packet->parsed_lines; a++)
                  {
                        if (packet->line[a].len >= 5)
                        {
                              //Check RCPT command

                              if ((packet->line[a].ptr[0] == 'R'
                              || packet->line[a].ptr[0] == 'r')
                              && (packet->line[a].ptr[1] == 'C'
                              || packet->line[a].ptr[1] == 'c')
                              && (packet->line[a].ptr[2] == 'P'
                              || packet->line[a].ptr[2] == 'p')
                              && (packet->line[a].ptr[3] == 'T'
                              || packet->line[a].ptr[3] == 't')
                              && packet->line[a].ptr[4] == ' ')
                              {
                                    end =
                                    packet->payload_packet_len;
                                    for (b = 0; b < end;  b++)
                                    {
                                          //Check p letter

                                    if(packet->payload[b] == 0x70)
                                          {

                                          //Check mail address

                                    if(b + sizeof(mail) < end
      && memcmp(&packet->payload[b], mail, sizeof(mail)) == 0)

                                                {
                              ndpi_int_mail_dest_add_conection
                              (ndpi_struct, flow);
                              return;
                                                }

                                          }else continue;

                                    }

                              }else return;

                        }else return;

                  }

            }else return;

      }

      NDPI_ADD_PROTOCOL_TO_BITMASK
(flow->excluded_protocol_bitmask, NDPI_PROTOCOL_MAIL_DEST);

}
#endif
```

Bibliografia

[1] Blumenthal, Marjory S. e Clark, David D. (2001). "Rethinking the design of the internet: the end- to-end arguments vs. the brave new world".
http://groups.csail.mit.edu/ana/Publications/PubPDFs/Rethinking%20the%20design%20of%20the%20internet2001.pdf

[2] ISCOM. "La sicurezza delle reti: dall'analisi del rischio alle strategie di protezione".
http://www.isticom.it/documenti/news/pub_002_ita.pdf

[3] "Il Modello ISO/OSI".
http://www.fis.unipr.it/lca/Corsi/iso_osi/iso_osi.html

[4] ISCOM (n2).

[5] Parsons, Christopher (2009). "Deep packet inspection in perspective: tracing its lineage and surveillance potentials".
http://www.sscqueens.org/sites/default/files/WP_Deep_Packet_Inspection_Parsons_Jan_2008.pdf

[6] "Cross-Packet Inspection (xPI)".
http://www.thetanetworks.com/index.php/technology/more/22-cross-packet-inspection-xpi

[7] Tveit, Paal (2008). "The virtues of continuous deep packet capture and stream-to-storage".
http://sharkfest.wireshark.org/sharkfest.08/T1-2_Tviet_Virtues%20of%20Continuous,%20Complete%20Packet%20Capture.pdf

[8] Cisco Systems, Inc. (2008). "Wan and application optimization solution guide".
http://www.cisco.com/en/US/docs/nsite/wan_optimization/WANoptSolutionGd.pdf

[9] AbuHmed, T., Mohaisen, A., Nyag, D. (2008). "A survey on deep packet inspection for intrusion detection systems".
http://arxiv.org/pdf/0803.0037.pdf

[10] SonicWALL. "DPI-SSL".
http://www.sonicwall.com/downloads/sonicos_enhanced_5.6_dpi-ssl_feature_module.pdf

[11] Mueller, Milton (2011). "DPI technology from the standpoint of internet governance studies: an introduction".
http://dpi.ischool.syr.edu/Technology_files/WhatisDPI-2.pdf

[12] Ivi (n11).

[13] "Copyright Compliance".
http://audiblemagic.com/solutions-compliance.php

[14] Daly, Angela (2010). "The legality of deep packet inspection".
http://papers.ssrn.com/sol3/papers.cfm?abstract-id=1628024

[15] ECHR. "Convenzione europea dei diritti dell'uomo".
http://www.echr.coe.int/NR/rdonlyres/0D3304D1-F396-414A-A6C1-97B316F9753A/0/Convention_ITA.pdf

[16] Daly (n14).

[17] ECHR (n15).

[18] "Direttiva 2002/19/CE del parlamento europeo e del consiglio".
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2002:108:0007:0007:IT:PDF

[19] Ivi (n18).

[20] Daly (n14).

[21] Allot Communications (2007). "Digging deeper into deep packet inspection(DPI)".
http://www.datacom.cz/system/files/dpi_white_paper.pdf

[22] Infonetics Research (2010).
http://www.infonetics.com/pr/2010/DPI-Deep-Packet-Inspection-Market-Highlights.asp

[23] "Solutions Overview".
http://www.sandvine.com/solutions/default.asp

[24] "Cisco service control application for broadband".
http://www.cisco.com/en/US/products/ps6135/index.html

[25] "Product Overview".
http://www.allot.com/Products_Overview.html

[26] Infonetics Research (2012).
http://www.infonetics.com/pr/2012/2H11-Service-Provider-DPI-Products-Market-Highlights.asp

[27] "nDpi".
http://www.ntop.org/products/ndpi/

[28] "Application Layer Packet Classifier for Linux".
http://l7-filter.sourceforge.net/

[29] "libprotoident".
http://research.wand.net.nz/software/libprotoident.php

[30] "ipoque's Industry Leading Deep Packet Inspection Engine Goes Open Source".
http://www.ipoque.com/en/news-events/press-center/press-releases/2009/ipoques-industry-leading-deep-packet-inspection-engine

[31] "ntop".
http://www.ntop.org/products/ntop/

[32] "ewildgoose / ndpi-netfilter".
https://github.com/ewildgoose/ndpi-netfilter

[33] "PF-RING".
http://www.ntop.org/products/pf_ring/

[34] "Why TNAPI (Threaded NAPI) ?".
http://www.ntop.org/products/pf_ring/tnapi/

[35] "Direct NIC Access".
http://www.ntop.org/products/pf_ring/dna/

[36] "Libzero for DNA".
http://www.ntop.org/products/pf_ring/libzero-for-dna/

[37] “Virtual PF_RING”.
http://www.ntop.org/products/pf_ring/vpf_ring/

[38] “Card sharing”.
http://it.wikipedia.org/wiki/Card_sharing

[39] “Why Deep Packet Inspection Falls Short”.
http://www.cable360.net/ct/sections/departments/52038.html#.UMHnppPm6bK

Ringraziamenti

Grazie all’opportunità che la vita mi ha dato.

Grazie alle persone con cui ho condiviso questa esperienza universitaria. Specialmente con chi sono cresciuto e ai colleghi di corso.

Grazie...