Introduzione agli Intrusion Detection System

Simone Piccardi
piccardi@truelite.it

Introduzione agli IDS – Prima edizione

Questa documentazione libera è stata sviluppata all'interno delle attività formative effettuate da Truelite S.r.l. Il materiale è stato finanziato nel corso della realizzazione dei corsi erogati dall'azienda, e viene messo a disposizione di tutti sotto licenza GNU FDL.

Questo testo, insieme al resto della documentazione libera realizzata da Truelite S.r.l., viene distribuito su internet all'indirizzo:

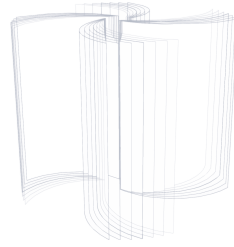
`http://svn.truelite.it/truedoc`

dove saranno pubblicate nuove versioni ed aggiornamenti.

Società italiana specializzata nella fornitura di servizi, consulenza e formazione esclusivamente su GNU/Linux e software libero.

Per informazioni:

Truelite S.r.l
Via Monferrato 6,
50142 Firenze.
Tel: 055-7879597
Fax: 055-7333336
e-mail: info@truelite.it
web: http://www.truelite.it



Indice

Capitolo 1

Sistemi di *Intrusion Detection*

1.1 Cosa sono e a cosa servono gli *IDS*

In questa prima sezione faremo una breve introduzione teorica sul significato e le motivazioni che portano all'uso dei cosiddetti *sistemi di rilevamento delle intrusioni*, a cui, da qui in avanti, faremo sempre riferimento con la sigla IDS, derivante dalla denominazione inglese *Intrusion Detection System*. Forniremo poi una panoramica sulle modalità di classificazione che vengono adoperate per distinguere fra loro le varie tipologie di programmi che rientrano in questa categoria generica.

1.1.1 La sicurezza e gli IDS

Come abbiamo illustrato in sez. **??** nell'informatica sono state adottate varie definizioni del termine sicurezza che ne caratterizzano il significato secondo vari aspetti; nel nostro caso ci siamo rifatti alla definizione che si basa sul soddisfacimento dei tre obiettivi fondamentali della *confidenzialità*, *integrità* e *disponibilità*.

Il paradigma classico della sicurezza cerca di realizzare questi obiettivi identificando utenti e risorse del sistema, e definendo una opportuna relazione fra di essi (vedi sez. **??**). In generale pertanto si deve disporre di un *meccanismo di identificazione* per consentire l'accesso agli utenti, e di un *meccanismo di controllo*, basato su una opportuna serie di permessi, che consente loro di eseguire nel sistema solo le operazioni che gli sono state consentite.

In sostanza il meccanismo si basa sulla risposta alle due domande "*chi sei?*" e "*cosa puoi fare?*". Di norma questo viene fatto attraverso una procedura di autenticazione, e l'uso estensivo dei permessi degli utenti. Il caso classico è quello dell'uso di username e password per l'autenticazione e l'uso dei permessi di utenti e gruppi per l'accesso; entrambe le funzionalità sono state ampiamente estese, ma il concetto di base resta lo stesso.

Tutto questo ovviamente funziona fintanto che l'implementazione dei programmi che gestiscono questi aspetti del sistema è corretta. Il problema è che la perfezione non è di questo mondo, ed esistono e sono esistite molte vulnerabilità dei programmi che opportunamente sfruttate rendono in grado un attaccante di superare o rendere inefficaci sia le procedure di autenticazione che quelle di controllo.

Questo è ancora più evidente nel caso dei sistemi in rete. Internet in particolare non è nata per garantire controllo degli accessi e non è provvista di meccanismi di identificazione o riservatezza; chiunque può, nel momento stesso in cui vi ha accesso, leggere quanto vi transita ed immettervi i suoi pacchetti. A livello del protocollo fondamentale, IP, non esiste nessun meccanismo di controllo o autenticazione. Benché da varie parti si sia corsi ai ripari, introducendo a vari livelli (SSL, IPSEC, VPN) alcune di queste funzionalità, resta il fatto che provvedere un meccanismo di autenticazione e controllo degli accessi a livello di rete è molto complesso e le difficoltà di gestione crescono enormemente.

Gli IDS nascono allora proprio dalla consapevolezza dei limiti intrinseci del paradigma di sicurezza basato sull'autenticazione ed il controllo degli accessi, e si fondano sulla constatazione che usualmente una violazione della sicurezza si manifesta attraverso un comportamento anomalo del sistema, che è quello che un IDS è progettato per rilevare. In sostanza dalle due precedenti domande "*chi sei?*" e "*cosa puoi fare?*" con un IDS si passa a porsi la domanda "*perché stai facendo questo?*".

Tutto ciò ci fa capire che in realtà un IDS non è un sostituto degli altri meccanismi di sicurezza, quanto piuttosto un ausilio a supporto del controllo della loro efficacia, un ausilio di grande efficacia certamente, ma che richiede la presenza di un'efficace politica di sicurezza perseguita con costanza, e l'attenzione da parte di personale qualificato. Perciò non si pensi di installare un IDS per non doversi preoccupare più della sicurezza, sarebbe assolutamente inutile, lo si installi solo quando ci si vuole, per scelta consapevole, *preoccupare* della sicurezza.

1.1.2 Tipologia degli IDS

In generale un IDS non si sostituisce mai ai vari controlli effettuati dai meccanismi di sicurezza posti a salvaguardia di un sistema, ma piuttosto cerca di scoprire un loro fallimento. In caso di violazione, o di tentativo di violazione di un sistema vengono infatti compiute azioni "*anomale*" che nelle normali operazioni non verrebbero mai eseguite; identificando quest'ultime diventa possibile scoprire la presenza di un eventuale intruso. Una delle difficoltà maggiori nell'uso efficace di un IDS è proprio quella di identificare tali azioni, e non confonderle con azioni assolutamente legittime, ancorché poco usuali.

In generale si hanno due metodologie principali per riconoscere le anomalie del sistema. Il primo e più diretto è quello chiamato *misuse detection*, cioè la rilevazione diretta di un utilizzo non consentito di alcune risorse. In genere questo viene fatto attraverso una qualche forma di rappresentazione (di norma con una serie di regole, più o meno complesse) che permetta di definire e osservare quali sono i comportamenti anomali da controllare.

Il limite di questo metodo è che occorre identificare con chiarezza il *misuse*, e se qualcuno utilizza un attacco che non era stato previsto questo verrà ignorato. A questo problema risponde la seconda metodologia, detta *anomaly detection* che cerca di classificare (in genere su base statistica, con vari modelli e meccanismi) il comportamento normale, identificando così come sospette tutte le deviazioni significative da esso; il grande vantaggio di questo approccio è la sua generalità, dato che è automaticamente in grado di adattarsi per riconoscere nuovi schemi di attacco. La difficoltà sta nella scelta dei modelli da utilizzare (cosa misurare, come impostare le soglie di allarme, ecc.) che permettano di selezionare in maniera adeguata, vale a dire sostenibile per il controllo da parte dell'amministratore, le eventuali situazioni sospette.

Una seconda classificazione degli IDS è quella che distingue fra quelli che eseguono le loro operazioni *on-line*, in tempo reale, rispondendo immediatamente quando riconoscono un tentativo di intrusione, e quelli che operano *off-line*, che sono in grado di effettuare il riconoscimento solo eseguendo una verifica in un momento successivo sulla base dei dati raccolti.

Nel primo caso (un esempio di IDS di questo tipo è `snort`, che vedremo in sez. 1.4.2), vengono generati degli opportuni allarmi (i cosiddetti *alert*) all'accadere di certe situazioni. Di norma questo viene fatto esaminando gli eventi correnti all'interno di una certa finestra temporale, e al rilevamento di possibili intrusioni vengono attivati gli allarmi, che a loro volta possono innescare azioni successive.

In questo caso una delle possibilità è quella di approntare delle contromisure automatiche volte a bloccare le operazioni coinvolte nell'azione di intrusione, come ad esempio bloccare il traffico di rete verso certe macchine. Questo è quello che fanno i cosiddetti IPS (*Intrusion Prevention System*), che vengono proposti come il passo successivo agli IDS.[1] Una reazione automatica di questo tipo espone però al rischio di poter essere sfruttata da un attaccante che abbia identificato la presenza di tali contromisure per rendere inutilizzabile il servizio con un attacco simulato, con il risultato di far fallire il terzo degli obiettivi fondamentali della sicurezza, la *disponibilità*.

In genere il limite degli IDS che operano in tempo reale è quello delle risorse necessarie per eseguire i controlli, dato che questi possono essere molto pesanti sia dal punto di vista computazionale che dal punto di vista dello stoccaggio dei dati. Negli IDS *offline* invece l'analisi viene svolta successivamente in un secondo tempo, e pertanto si può spendere più tempo non essendo necessaria la risposta immediata.

In genere con gli IDS che operano *offline* si ha la capacità di una analisi molto più completa e dettagliata, e potendo operare senza limiti di tempo, si possono anche trattare moli di dati non analizzabili in tempo reale; il problema è che essi, per definizione, sono utilizzabili solo dopo che è avvenuto un incidente o un attacco. In alcuni casi però (come `aide`, che vedremo in sez. 1.3.2) l'uso di un IDS di questo tipo può essere estremamente utile per rilevare il tipo di intrusione e porvi rimedio senza essere costretti a soluzioni più drastiche come la reinstallazione.

In molti casi oggi esistono degli IDS (come avviene per lo stesso `snort` che tratteremo in sez. 1.4.2) che sono in grado di fornire una combinazione di queste due caratteristiche, gestendo sia la generazione di allarmi *on-line* che la registrazione degli eventi relativi ad un allarme per consentirne una successiva, e molto più accurata, analisi *off-line*.

L'ultima classificazione degli IDS prevede altre due categorie, la prima è quella degli IDS *host-based*, in cui il controllo viene eseguito su una singola stazione, relativamente alle attività registrate dall'IDS sulla stessa. Un esempio è sempre `aide`, che rientra in questa categoria, così come gli analizzatori dei file di log; ma esistono numerose alternative, spesso anche combinabili fra loro. Torneremo su questo argomento in sez. 1.3.

La seconda categoria è quella degli IDS *network-based*, in cui viene invece controllato il traffico su una rete alla ricerca di tracce di intrusione fra i pacchetti in transito. In genere si tende a identificare questi ultimi, di cui `snort` è il più noto e quello che tratteremo in dettaglio, che più propriamente dovrebbero essere chiamati NIDS (cioè *Network Intrusion Detection System*), con gli IDS generici.

[1] anche il citato `snort` ha la possibilità di implementare questo tipo di politiche.

1.2 Tecniche di rilevamento

Uno dei compiti principali nelle attività di gestione della sicurezza è quello della ricerca delle possibili falle che possono comprometterla. Pertanto diventa fondamentale, per poter affrontare poi il funzionamento degli IDS, conoscere ed analizzare i principali strumenti di analisi, scansione e ricerca delle vulnerabilità, che sono gli stessi che potrebbero essere usati contro di voi per rilevare i vostri punti deboli da un eventuale attaccante.

Per questo motivo in questa sezione prenderemo in esame alcune tipologie di strumenti usati tipicamente per la raccolta di informazioni sulle macchine ed i servizi presenti su una rete, per intercettare ed analizzare il traffico di rete e per la scansione delle vulnerabilità presenti. Per ciascuna tipologia presenteremo il principale programma disponibile su GNU/Linux per svolgere il relativo compito.

1.2.1 I *portscanner*

Uno degli strumenti principali per l'analisi della propria rete, e per la ricerca di eventuali punti deboli e possibilità di accesso non autorizzato è il cosiddetto *portscanner*. Un *portscanner* è semplicemente un programma in grado di rilevare quali servizi di rete sono attivi su una macchina o in una intera rete, eseguendo una scansione sulle porte (in genere TCP, ma anche UDP) per identificare quali di esse sono correntemente utilizzate da un qualche demone di sistema per fornire servizi.

Come tutti gli strumenti di analisi usati per la sicurezza un *portscanner* può essere usato sia per scopi offensivi (la ricerca di servizi vulnerabili da attaccare) che difensivi (la ricerca di servizi vulnerabili o inutili da chiudere), in generale è comunque uno strumento indispensabile per verificare in maniera effettiva la propria rete, dato che in presenza di una macchina compromessa non si può avere nessuna fiducia nei programmi diagnostici eseguiti su detta macchina (ad esempio `netstat` per tracciare le connessioni attive), che potrebbero essere stati a loro volta compromessi.

Infatti, in caso di violazione di una macchina, uno dei primi passi eseguiti di solito da chi ha effettuato l'intrusione è quello di predisporre una *backdoor* e poi modificare lo stesso kernel o i programmi che potrebbero rilevarne la presenza (come `ps`, `ls`, `netstat`) installando un *rootkit*. È chiaro che se il sistema è stato modificato in modo da mascherare una eventuale intrusione, usare i programmi dello stesso per effettuare controlli, in particolare per rilevare porte aperte in ascolto per connessioni remote, non è una procedura affidabile.

In questi casi diventa essenziale poter usare un *portscanner* partendo da una macchina fidata (o da una distribuzione live su CDROM), sia come meccanismo di verifica e controllo della propria rete, che come strumento per la scoperta della presenza di eventuali servizi anomali, abusivi o semplicemente non previsti o non disabilitati. Inoltre è prassi comune (ed anche consigliata) usare un *portscanner* a scopo diagnostico per verificare l'efficacia delle impostazioni di un firewall e l'effettiva funzionalità dello stesso.

Nel caso di GNU/Linux, ma lo stesso vale per la gran parte degli altri sistemi unix-like, ed esiste una versione anche per Windows, il *portscanner* di gran lunga più utilizzato è `nmap`, in genere presente in tutte le distribuzioni,[2] e comunque reperibile su `http://www.insecure.org/nmap/index.html`. Noi esamineremo soltanto la versione a riga di comando, segnaliamo però

[2] nel caso di Debian basta installare il pacchetto omonimo.

l'esistenza di una versione grafica, `zenmap`, con la quale si può perdere un sacco di tempo a muovere le dita fra tastiera e mouse per fare le stesse cose.[3]

La principale caratteristica di `nmap` è la sua enorme flessibilità, il programma infatti è in grado di eseguire i *portscan* con una grande varietà di metodi, ponendo una grande attenzione ad implementare anche quelli che permettono o di mascherare la propria attività o di non farsi identificare. In questo modo è in grado di rilevare la presenza di firewall, di determinare il sistema operativo usato da una macchina e di ricavare le versioni degli eventuali server installati, nonché di analizzare i servizi RPC. Inoltre, sia pure con funzionalità più limitate, può essere utilizzato anche senza avere i privilegi di amministratore.

Nella sua forma più elementare il comando richiede soltanto di specificare l'indirizzo della macchina (o della rete) da analizzare; quello che nella pagina di manuale viene chiamato *bersaglio* (o *target*). Si possono anche indicare più bersagli con una lista separata da spazi e questi possono essere specificati sia attraverso il loro hostname che con l'indirizzo IP numerico. Si possono inoltre specificare intere reti da analizzare che devono essere indicate con la notazione CIDR, e questa è anche una delle caratteristiche più importanti di `nmap`, che lo rende un ausilio di grande utilità per rilevare le macchine presenti su una rete di cui non si sa nulla.

Oltre alla semplice indicazione di un indirizzo di rete il comando supporta anche una sintassi che permette di specificare il bersaglio del *portscan* in maniera alquanto sofisticata, specificando elenchi di indirizzi separati da virgole (cioè con il carattere "`,`"), intervalli separati da "`-`", e anche caratteri jolly come "∗". Inoltre questi possono essere usati in qualunque parte dell'indirizzo, per cui è legittimo, anche se poco leggibile, anche un bersaglio del tipo "`192.168.1,2.1,2,100-200`".

Specificando più bersagli o una intera rete verranno visualizzati in sequenza i risultati per ciascuna delle singole macchine che vengono trovate attive via via che la relativa scansione viene terminata. Dato che la scoperta di quali macchine sono presenti su una rete può essere una operazioni lunga, qualora se ne abbia già un elenco lo si può fornite al comando passandoglielo con l'opzione `-iL`. Questa richiede come parametro un nome di file da cui leggere l'elenco degli indirizzi da controllare, il cui formato richiede semplicemente che questi siano espresse come elenco (separato con spazi, tabulatori, o a capo) di una qualunque delle forme supportate sulla riga di comando; si possono anche inserire commenti, dato che tutto quello che segue il carattere `#` verrà ignorato.

Quando si indicano intere reti si possono escludere dalla scansione un certo numero di macchine con l'opzione `--exclude`, che richiede come parametro l'elenco dei relativi indirizzi separati da virgole (e senza spazi). Questo può risultare utile per rendere più evidente la presenza di macchine non previste rispetto a quelle note, o per evitare di eseguire il *portscan* su macchine critiche.

Un possibile esempio elementare dell'uso di `nmap`, in cui si esegue la scansione per una singola macchina identificata tramite il suo indirizzo IP è il seguente:

```
# nmap 192.168.1.2
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-04 19:38 CEST
Nmap scan report for davis.fi.trl (192.168.1.2)
Host is up (0.0061s latency).
Not shown: 990 closed ports
PORT     STATE SERVICE
```

[3] in realtà con la versione 5.x il programma, un tempo chiamato semplicemente `nmapfe`, si è arricchito della capacità di registrare i dati delle varie scansioni effettuate, e mantenere una sorta di *knowledge base*.

```
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
443/tcp  open  https
636/tcp  open  ldapssl
2049/tcp open  nfs
3306/tcp open  mysql
9102/tcp open  jetdirect
MAC Address: 52:54:00:D3:BD:31 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.14 seconds
```

La grande potenza di `nmap` è nella enorme flessibilità con cui lo si può usare, ed il programma supporta una innumerevole serie di opzioni che consentono di controllare le sue caratteristiche. Data l'ampiezza delle stesse prenderemo in considerazione solo quelle più significative, per un elenco completo si può fare riferimento alla pagina di manuale, accessibile con `man nmap`, che contiene una documentazione molto dettagliata; un breve riassunto può invece essere ottenuto eseguendo direttamente il comando come `nmap -h` o invocandolo senza argomenti.

Le opzioni sono suddivise in varie categorie: per indicare la tecnica di scansione utilizzata, per indicare la tecnica di rilevazione della presenza delle macchine, per indicare le temporizzazioni del *portscan* ed infine quelle generiche. Molte sono, rispetto all'utilizzo normale, identificate da due lettere, la prima viene usata per specificare una funzionalità generica, la seconda per indicare con quale metodo realizzarla.

Si tenga presente che per poter essere usato, dovendo creare nella maggior parte dei casi dei pacchetti costruiti *ad-hoc*, `nmap` richiede i privilegi di amministratore. Qualora questi non siano disponibili e le modalità di *portscan* richieste li prevedano, il programma terminerà con un messaggio di errore. Esiste comunque la possibilità di usarlo, con funzionalità limitate, anche da utente normale.

Le opzioni che indicano la metodologia di scansione sono riassunte in tab. 1.1, esse iniziano tutte con `-s` (con l'eccezione di `-b` che comunque è deprecata) cui segue una seconda lettera che specifica la tecnica da utilizzare. Se non si indica niente la metodologia utilizzata di default dipende da chi esegue la scansione: se il processo ha i privilegi di amministratore verrà eseguito un *SYN scan*, altrimenti `nmap` utilizzerà il *Connect scan*.

Vale la pena addentrarsi nella spiegazione delle tecniche di scansione usate da `nmap` in quanto questo ci permette di capire meglio sia le metodologie utilizzate dagli attaccanti per identificare i servizi presenti sulle macchine, che i criteri con cui i sistemi antintrusione, su cui torneremo più avanti, cercano di rilevare i tentativi di analisi. Una delle caratteristiche di `nmap` infatti è quella di supportare tutte le tecniche che rendono il più difficoltoso possibile individuare la presenza di un *portscan* o identificare chi lo sta eseguendo.

La forma più elementare di *portscan*, e l'unica utilizzabile dagli utenti non privilegiati, è il *Connect scan*, che usa la ordinaria interfaccia dei socket, eseguendo una normale connessione su ciascuna porta. In questo caso le connessioni su delle porte aperte verranno rilevate per la risposta del rispettivo server e chiuse immediatamente dopo, mentre quelle su delle porte chiuse saranno abortite con un errore o non riceveranno affatto risposta; il primo caso è il comportamento standard di uno stack TCP/IP, che richiede l'emissione di un pacchetto RST ad ogni tentativo di connessione su una porta chiusa, il secondo caso è quello un cui si sia installato

Opzione	Descrizione
`-sS`	*TCP SYN scan.* Esegue una scansione dei servizi attivi su TCP inviando un singolo pacchetto TCP con il flag SYN attivo.
`-sT`	*TCP Connect scan.* Esegue una scansione dei servizi attivi su TCP, effettuando una normale connessione TCP.
`-sF`	*Stealth FIN scan.* Invia un singolo pacchetto TCP con il flag FIN attivo.
`-sX`	*Xmas Tree scan.* Esegue una scansione dei servizi attivi su TCP inviando un singolo pacchetto TCP con i flag FIN, URG, e PUSH attivi.
`-sN`	*Null scan* Esegue una scansione dei servizi attivi su TCP inviando un singolo pacchetto TCP senza nessun flag attivo.
`-sM`	*Maimon scan.* Esegue una scansione con la tecnica scoperta da Uriel Maimon, usa un pacchetto TCP con i flag FIN/ACK che viene scartato per le porte aperte su molti sistemi derivati da BSD.
`-sI`	*Idle scan.* Esegue una scansione dei servizi attivi su TCP sfruttando un'altra macchina.
`-sA`	*ACK scan.* Analizza il comportamento di un firewall inviando pacchetti TCP con il solo flag ACK per verificare se le relative porte sono filtrate.
`-sW`	*Window scan.* Analizza il comportamento di un firewall per verificare il filtraggio delle porte come il precedente *ACK scan* ma permette di distinguere in alcuni casi porte aperte da porte chiuse fra quelle non filtrate.
`-sU`	*UDP scan.* Esegue una scansione sulle porte UDP.
`-sY`	*SCTP INIT scan.* Esegue una scansione sulle porte del protocollo SCTP analoga al *SYN scan.*
`-sZ`	*SCTP COOKIE ECHO scan.* Esegue una scansione sulle porte del protocollo SCTP con una modalità alternativa.
`-sO`	*IP protocol scan.* Cerca di determinare i protocolli supportati direttamente sopra IP.
`-b`	esegue un *FTP bounce attack*, una tecnica basata sull'appoggiarsi ad un server FTP vulnerabile, ormai inefficace e deprecata.
`-sR`	*RPC scan.* Cerca di verificare se le porte trovate aperte corrispondono a servizi di tipo RPC, può essere usato in congiunzione con gli altri tipi di scansione.
`-sV`	*Version detection.* Esegue un controllo sui servizi trovati attivi per identificarne la versione, può essere usato in congiunzione con gli altri tipi di scansione.
`-sC`	*Script scan.* Esegue una scansione facendo ricorso agli appositi script di analisi forniti con il programma, consentendo ricerche di informazioni molto complesse e sofisticate.
`-sL`	*List scan.* Stampa semplicemente la lista degli IP della rete specificata, con le eventuali risoluzioni inverse.
`-sP`	*Ping scan.* Verifica soltanto la presenza attiva di una macchina con la relativa tecnica di *ping* (vedi tab. 1.3).

Tabella 1.1: Opzioni di `nmap` per l'indicazione delle tecniche di scansione.

un firewall che scarta i relativi pacchetti. Lo svantaggio di questo metodo è che può essere rilevato anche soltanto dall'analisi dei file di log, per gli errori generati dai servizi che rilevano una connessione che viene chiusa immediatamente dopo l'apertura, senza che ci sia stato nessun traffico.

Come accennato la tecnica di scansione eseguita di default quando si è amministratori è il *SYN scan*, che viene chiamata anche *semi-apertura* (o *half-open*) in quanto si limita a eseguire la prima parte dell'apertura di una connessione TCP, cioè l'invio di un pacchetto SYN. Questa tecnica identifica le porte aperte per la risposta standard di un SYN+ACK, mentre in caso di porta chiusa si avrà l'emissione di un pacchetto RST.

Questo metodo non necessita di concludere la connessione, ed alla ricezione dell'eventuale SYN+ACK il kernel stesso si incaricherà di rispondere con un RST; questo perché il SYN originale non è stato inviato da un processo nell'aprire una connessione, ma con un pacchetto creato ad arte da `nmap`. Questo significa che il SYN+ACK ottenuto in risposta non corrisponde a nessuna connessione esistente, per cui il nostro kernel reagirà secondo lo standard emettendo a sua volta un pacchetto RST interrompendo così la connessione prima che essa venga stabilita. Il risultato finale è che il server in ascolto non si accorgerà di niente dato che la connessione non inizia neanche. Il problema di questa tecnica di scansione è che è la più comune e anche i programmi di sorveglianza più semplici, come *synlogger*, sono in grado di rilevarla.

Per cercare di rendere meno visibili i *portscan* sono state elaborate altre tecniche che permettono di ottenere lo stesso risultato del *SYN scan* usando dei pacchetti TCP con diverse combinazioni dei flag, che non vengono rilevati dai programmi di sorveglianza più semplici (anche se nessuna di queste tecniche è efficace con programmi di rilevazione evoluti come `snort`) ma permettono di passare attraverso quei firewall che si limitano a bloccare i pacchetti SYN per le nuove connessioni. Questa significa anche che tecnica non è comunque efficace con Linux, se si è usato `iptables` per bloccare le connessioni entranti utilizzando filtro sugli stati.

Tutte queste tecniche di scansione si basano sul fatto che la standardizzazione del protocollo TCP prescrive che se una porta è chiusa deve essere emesso un pacchetto RST come risposta a qualunque pacchetto in arrivo verso di essa, mentre in caso di porta aperta eventuali pacchetti "*spuri*" in arrivo devono essere semplicemente scartati. Questo permette di distinguere i due casi e riconoscere le porte aperte; se invece si ricevono dei messaggi di errore (con i relativi pacchetti ICMP) la porta può considerarsi senz'altro filtrata da un firewall (si potrebbe ottenere questo comportamento con il target `REJECT` di `iptables`) dato che questo tipo di risposta non è prevista dallo standard del TCP.

Nel caso del cosiddetto *Stealth FIN* (`-sF`) viene utilizzato un pacchetto con il solo flag FIN (quello usato per la chiusura di una connessione), per il cosiddetto *Xmas Tree scan* (`-sX`) si usa un pacchetto, illegale dal punto di vista del protocollo, che attiva i flag FIN, URG e PUSH, mentre per il cosiddetto *Null scan* (`-sN`) si usa un pacchetto in cui vengono disattivati tutti i flag. Il programma inoltre supporta l'opzione `--scanflags` che permette di usare una qualunque combinazione di flag del protocollo TCP, anche se non tutte le combinazioni possono dare risultati utili. Il problema con questo tipo di scansione è che alcune implementazioni del protocollo TCP (in particolare quella di Windows) non seguono lo standard e rispondono con dei pacchetti RST anche quando le porte sono aperte, rendendo inefficace la scansione; in compenso questo comportamento permette di identificare il sistema operativo.

Una ulteriore variante di scansione è il cosiddetto *Maimon scan* (`-sM`), che sfrutta una particolarità delle risposte riscontrate con alcuni sistemi derivati da BSD scoperta da Uriel Maimon,

(la tecnica è descritta sul numero 49 di *Phrack Magazine*) per i quali in risposta all'invio di un pacchetto con i flag FIN/ACK non viene sempre inviato il canonico RST, ma il pacchetto viene scartato in caso di porta aperta, consentendone così il riconoscimento.

Infine una delle tecniche più sofisticate per il *portscan* su TCP utilizzata da `nmap` è quella del cosiddetto *Idle scan*, che permette di eseguire una scansione completamente anonima, facendo figurare come sorgente della scansione l'indirizzo IP di una altra macchina usata come intermediario (che viene chiamata *zombie host*). La tecnica si attiva con l'opzione `-sI` seguita dall'IP dello *zombie host*; ;a scansione usa di default la porta 80 dello *zombie host* per le sue rivelazioni, si può specificare una porta diversa specificando l'indirizzo di quest'ultimo come `indirizzo:porta`.

L'*Idle scan* funziona utilizzando il campo di *fragmentation ID* dell'intestazione dei pacchetti IP, che viene usato nella gestione della frammentazione per identificare, in fase di riassemblaggio, tutti i pacchetti appartenenti allo stesso pacchetto originario che è stato frammentato.[4] La maggior parte dei sistemi operativi si limitano ad incrementare questo campo ogni volta che inviano un pacchetto, cosicché se la macchina non è sottoposta a traffico, è immediato determinarne e prevederne il valore, basta eseguire un *ping* sulla macchina bersaglio e leggerlo dai pacchetti di risposta.

Se allora si esegue un *SYN scan* usando come indirizzo IP sorgente quello dello *zombie host* questo riceverà le risposte al posto nostro, risposte che saranno un SYN+ACK nel caso di porta aperta (cui risponderà con un RST, non avendo iniziato la connessione) o un RST in caso di porta chiusa. Nel primo caso lo *zombie host* invia un pacchetto, nel secondo no, per cui osservando il *fragmentation ID* si potrà dedurre la risposta della macchina sottosta a *portscan*.

Ovviamente perché questo sia possibile occorre trovare un candidato valido per fare da *zombie host*, una macchina cioè che presenti una implementazione dello stack TCP/IP vulnerabile (quasi tutte, fanno eccezione Linux, Solaris e OpenBSD), e che non presenti traffico; in generale le stampanti di rete sono dei candidati ideali. La verifica della fattibilità dell'*Idle scan* viene fatta automaticamente da `nmap` che esegue una serie di controlli prima di eseguire la scansione (fermandosi qualora non sia possibile).

Tutto questo mette in grado un attaccante di eseguire una scansione in maniera completamente anonima, facendo figurare l'indirizzo di un'altra macchina come sorgente della stessa. Anche per questo motivo non è mai il caso di prendersela troppo per un *portscan*, considerato che il soggetto di una vostra eventuale reazione potrebbe essere a sua volta una vittima.

Alle precedenti tecniche di scansione completa, si aggiungono due tipologie di scansione che forniscono informazioni parziali, e consentono di capire se una porta è filtrata o meno tramite la presenza di un firewall. La prima è il cosiddetto *ACK scan* (`-sA`), che invia un pacchetto TCP con il solo flag ACK; in tal caso la risposta prevista dallo standard TCP è un pacchetto RST, che se ricevuto consente di identificare la porta come non filtrata (ancorché non si possa stabilire se essa è aperta o chiusa), mentre l'assenza di risposta, o la risposta con un qualsiasi altro messaggio di errore via ICMP è indice della presenza di un qualche meccanismo di filtraggio.

Come variante del precedente si può usare il cosiddetto *Window scan* (`-sW`) che consente di riconoscere in caso di risposta con un pacchetto RST se la porta è aperta o meno, grazie al fatto

[4] il campo viene usato insieme all'indirizzo IP, se non ci fosse si avrebbe la possibilità di ambiguità per pacchetti frammentati provenienti dallo stesso IP.

che alcuni sistemi restituiscono nel pacchetto RST un valore positivo per la dimensione della finestra TCP[5] se la porta è aperta, o nullo se la porta è chiusa.

Le tecniche di scansione illustrate fino a qui riguardano soltanto il protocollo TCP, esistono comunque anche servizi forniti su UDP, ed `nmap` consente di eseguire una scansione anche su di essi con l'opzione `-sU`. La tecnica consiste nell'inviare pacchetti UDP di lunghezza nulla su ogni porta, ma si può anche mandare un pacchetto di dimensione fissa con dati casuali usando l'opzione aggiuntiva `--data-length`. Se la porta è chiusa la risposta dovrebbe essere un pacchetto ICMP di tipo *port unreachable*, altrimenti si assume che la porta sia aperta o filtrata da un firewall; per alcune porte (53/DNS, 161/SMTP) vengono però usati pacchetti di test specifici per il relativo protocollo, che possono permettere di capire se la porta è aperta o meno ottenendo una risposta.

Questa tecnica di rilevazione però non è molto affidabile per una serie di motivi: anzitutto se un firewall blocca i pacchetti ICMP *port unreachable* di risposta tutte le porte figureranno come aperte o filtrate; lo stesso problema si presenta se ad essere bloccati dal firewall sono i pacchetti UDP indirizzati verso le porte di destinazione. In entrambi i casi non si riceverà risposta, ma non si potrà concludere in maniera certa che il servizio è attivo e le porte pertanto potranno risultare o aperte o filtrate.

La scansione inoltre può risultare particolarmente lenta in quanto alcuni sistemi operativi limitano il numero di pacchetti ICMP di tipo *destination unreachable*[6] che possono essere emessi (nel caso di Linux c'è un massimo di 1 al secondo, con Windows invece il limite non esiste). Anche se `nmap` è in grado di rilevare questo comportamento ed evitare l'invio di pacchetti che comunque non riceverebbero risposta, questo comunque rallenta di molto la scansione e per esempio per ottenere il seguente risultato si è dovuto aspettare circa una ventina di minuti:

```
# nmap -sU 192.168.1.2
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-05 12:34 CEST
Nmap scan report for davis.fi.trl (192.168.1.2)
Host is up (0.00080s latency).
Not shown: 995 closed ports
PORT     STATE         SERVICE
53/udp   open          domain
67/udp   open|filtered dhcps
111/udp  open          rpcbind
123/udp  open          ntp
2049/udp open          nfs
MAC Address: 52:54:00:D3:BD:31 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 1066.72 seconds
```

Con la versione 5 di `nmap` è inoltre possibile effettuare scansioni anche con il nuovo protocollo SCTP, anche se queste restano meno utili in quanto detto protocollo non è molto utilizzato. Per queste scansioni sono previste due apposite opzioni, `-sY` che effettua un *SCTP INIT scan*, analogo al *SYN scan*, ed `-sZ` che effettua un *SCTP COOKIE ECHO scan*, usando una modalità di rilevamento più occulta ma meno affidabile. Un esempio, di scarso interesse non essendovi servizi attivi, è il seguente:

[5] la *Window size* è una delle informazioni inserite nei pacchetti TCP che serve ad indicare all'altro capo della connessione quanti byte di dati si vorrebbero ricevere, in modo da ottimizzare le trasmissioni.

[6] un pacchetto ICMP *port unreachable* è uno dei diversi possibili pacchetti di tipo *destination unreachable*.

```
# nmap -sY 192.168.1.4
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-05 19:13 CEST
Nmap scan report for parker.fi.trl (192.168.1.4)
Host is up (0.00012s latency).
All 42 scanned ports on parker.fi.trl (192.168.1.4) are filtered
MAC Address: 00:0E:A6:F4:DA:AA (Asustek Computer)

Nmap done: 1 IP address (1 host up) scanned in 1.69 seconds
```

L'ultima tecnica di scansione vera e propria è quella che permette di rilevare il supporto per ulteriori protocolli trasportati direttamente su IP (oltre ai classici TCP e UDP ed al nuovo SCTP) pur non essendo propriamente un *portscan*, dato che si esegue la scansione sul numero di protocollo dell'intestazione IP e non su un numero di porta, che non esiste per protocolli che non fan parte del livello di trasporto. Questa viene scansione viene attivata con l'opzione `-sO`, e viene eseguita inviando pacchetti IP vuoti, con la sola indicazione del protocollo nell'intestazione (con l'eccezione dei protocolli TCP, UDP, ICMP, SCTP, e IGMP per i quali esistono appositi pacchetti di prova).

Se il protocollo non è implementato la risposta prevista dallo standard è la restituzione un pacchetto ICMP *protocol unreachable*, che comporta la classificazione del protocollo come chiuso, se si riceve una qualche altra risposta lo si classifica come aperto, mentre in caso di ricezione di un tipo diverso di pacchetto ICMP di *destination unreachable* lo si marca come filtrato. Se non si riceve alcuna risposta lo stato è indefinito ed il protocollo è marcato come filtrato o chiuso. Dato che di nuovo è necessario osservare una risposta di tipo *destination unreachable* valgono le precedenti osservazioni fatte riguardo la scansione su UDP (cioè che la scansione è lenta). Un esempio del risultato è il seguente:

```
# nmap -sO 192.168.1.2
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-05 12:58 CEST
Nmap scan report for davis.fi.trl (192.168.1.2)
Host is up (0.00089s latency).
Not shown: 248 closed protocols
PROTOCOL STATE         SERVICE
1        open          icmp
2        open|filtered igmp
6        open          tcp
17       open          udp
103      open|filtered pim
136      open|filtered udplite
138      open|filtered unknown
253      open|filtered experimental1
MAC Address: 52:54:00:D3:BD:31 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 277.58 seconds
```

Si noti come nei risultati dei *portscan* illustrati finora, `nmap` faccia riferimento ad uno stato delle porte per cui ha effettuato una scansione; i nomi di questi stati, ed il relativo significato, sono riportati in tab. 1.2. I risultati più comuni per la scansione di una porta sono due, che questa risulti chiusa (`closed`) o bloccata da un firewall (`filtered`). Il primo caso è quello che si ottiene normalmente per le macchine a cui si ha accesso diretto (ad esempio quelle sulla propria rete locale) mentre il secondo è comune per le macchine direttamente esposte su Internet, che sono di solito protette da un firewall.

Come si può notare negli esempi precedenti `nmap` riporta sempre (nella riga "`Not shown:`") un riassunto esplicativo dello stato delle porte per le quali non ha ottenuto nessuna informazione significativa, mentre elenca esplicitamente soltanto le porte aperte o non filtrate. In alcuni casi, quando il programma non è in grado di determinare un risultato possono anche essere riportati più stati. In realtà le possibili ambiguità sono solo due:[7] si può ottenere `open|filtered` nelle scansioni in cui è possibile non ricevere risposta anche se la porta è aperta (ad esempio con UDP), ed in questo caso l'assenza di risposta ha lo stesso risultato di un firewall che scarta i pacchetti. Si può invece ottenere `closed|filtered` solo con l'*Idle scan* quando il programma non è in grado di distinguere fra una porta chiusa ed una filtrata.

Stato	Significato
`open`	la porta è raggiungibile ed è stato rilevato un programma in ascolto su di essa con cui è possibile interagire.
`closed`	la porta è raggiungibile ma nessun programma è in ascolto su di essa.
`filtered`	la porta non è raggiungibile a causa della presenza di un meccanismo di filtraggio (firewall o altro) che non consente di determinare niente al suo riguardo.
`unfiltered`	la porta è raggiungibile, ma `nmap` non è in grado di determinare se sia aperta o chiusa.

Tabella 1.2: Gli stati riportati da `nmap`.

Si tenga presente inoltre che `nmap` considera come filtrate sia le porte per le quali non riceve risposta (ad esempio perché il firewall scarta tutti i pacchetti ad esse inviati) o per le quali riceve una risposta che le qualifica come tale (un ICMP di *destination unreachable* con codice *communication administratively prohibited*), o non coerente con quanto dettato dagli standard (ad esempio pacchetti ICMP con codice non corrispondente a quello che ci si aspetterebbe), indice di una risposta generata da un sistema di protezione, come quelle che si possono far generare a `iptables` con il target `REJECT`.

Nonostante quel che possa pensare chi cerca di nascondere la presenza di un firewall con questo tipo di accorgimenti, di queste due possibili situazioni la più frustrante per un attaccante è la prima, perché rende la scansione molto più difficile e lenta, in quanto `nmap` per essere sicuro che l'assenza di risposta è reale e non dovuta a problemi di rete, deve ripetere vari tentativi di accesso. Visto poi che nascondere la presenza di un firewall non sorte nessun effetto maggiore di sicurezza, e comporta un carico amministrativo maggiore, l'uso di questa pratica non ha poi molto senso.

Altre opzioni classificate fra quelle di tab. 1.1, piuttosto che eseguire una scansione vera e propria, servono ad indicare la richiesta di ulteriori controlli. Una di queste è `-sR`, che invia su ogni porta trovata aperta dei comandi RPC nulli; in questo modo è possibile determinare quali di queste porte sono associate ad un servizio RPC.[8] L'opzione può essere attivata per gli scan TCP ed UDP, e se usata da sola implica un *SYN scan*. Un esempio del risultato dell'uso di questa opzione è il seguente:

[7] nella documentazione di `nmap` queste due situazioni sono considerate come altri due stati.

[8] La sigla RPC (*Remote Procedure Call*) indica una tipologia di servizi che viene utilizzata per fornire ai programmi la possibilità di eseguire delle "*funzioni remote*" (cioè chiamate da una macchina, ma eseguite su un'altra) attraverso una opportuna interfaccia; sono caratterizzati dall'usare porte allocate dinamicamente, il cui valore è desumibile contattando un apposito servizio, il *portmapper*, che ascolta sempre sulla porta 111.

```
# nmap -sR 192.168.1.2
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-06 11:17 CEST
Nmap scan report for davis.fi.trl (192.168.1.2)
Host is up (0.0040s latency).
Not shown: 990 closed ports
PORT     STATE SERVICE              VERSION
22/tcp   open  unknown
25/tcp   open  unknown
53/tcp   open  unknown
80/tcp   open  unknown
111/tcp  open  rpcbind (rpcbind V2) 2 (rpc #100000)
443/tcp  open  unknown
636/tcp  open  unknown
2049/tcp open  nfs (nfs V2-4)       2-4 (rpc #100003)
3306/tcp open  unknown
9102/tcp open  unknown
MAC Address: 52:54:00:D3:BD:31 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 1.42 seconds
```

Con l'opzione `-sV` invece si richiede, una volta eseguita la scansione, un tentativo di connessione sui servizi trovati attivi per determinare, sulla base delle risposte ottenute, una versione che verrà poi stampata all'interno del risultato della scansione. Questa opzione ricomprende anche i risultati di `-sR` che pertanto oggi è praticamente inutilizzata. L'opzione `-sV` può essere abbinata a qualunque scansione delle porte, e supporta anche una serie di opzioni ausiliarie che controllano il comportamento del sistema di rilevamento di versione, per le quali si rimanda alla pagina di manuale; se non ne viene specificata nessuna tipologia di scansione verrà eseguito un *SYN scan*. Un esempio dell'uso di questa opzione è il seguente:

```
# nmap -sV 192.168.1.4
Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-06 11:44 CEST
Nmap scan report for parker.fi.trl (192.168.1.4)
Host is up (0.00011s latency).
Not shown: 993 closed ports
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 5.1p1 Debian 5 (protocol 2.0)
25/tcp   open  smtp        Postfix smtpd
80/tcp   open  http        Apache httpd 2.2.9 ((Debian))
111/tcp  open  rpcbind     2 (rpc #100000)
631/tcp  open  ipp         CUPS 1.3
2000/tcp open  cisco-sccp?
6566/tcp open  unknown
MAC Address: 00:0E:A6:F4:DA:AA (Asustek Computer)
Service Info: Host:  parker.truelite.it; OS: Linux

Service detection performed. Please report any incorrect results at http://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.22 seconds
```

La più sofisticata fra le opzioni per la richiesta di scansioni aggiuntive è però `-sC`, che richiede l'intervento dell'*Nmap Scripting Engine* che consente di far eseguire sulle porte identificate da `nmap` tutte le operazioni aggiuntive che si vogliono tramite opportuni script in LUA, in questo modo ad esempio diventa possibile eseguire un programma di password cracking su una porta

aperta, o effettuare ulteriori operazioni di scansione per informazioni relative al servizio presente su di essa.

Con `nmap` vengono forniti anche una serie di script "*ufficiali*" elaborati all'interno del progetto, e si può controllare quali di questi devono essere eseguiti con l'opzione `--script`. Se non si indica nulla vengono eseguiti gli script di default (l'opzione `-sC` è equivalente ad usare `--script=default`) altrimenti si può specificare con questa opzione una lista (in elenco separato da virgole) di nomi di file, directory, categorie di script da eseguire, si possono anche specificare delle espressioni complesse con gli operatori logici `not`, `and` e `or` ed utilizzare i caratteri jolly.

In genere gli script sono installati in una directory di default,[9] usata anche come base per i pathname relativi, ma se ne può specificare un'altra con l'opzione `--datadir`. Quando si indica una directory vengono usati come script tutti i file con estensione `.nse`. Un esempio di invocazione del comando potrebbe essere:

```
# nmap -sC 192.168.1.4
Starting Nmap 5.21 ( http://nmap.org ) at 2011-12-23 21:00 CET
Nmap scan report for parker.fi.trl (192.168.1.4)
Host is up (0.00037s latency).
Not shown: 993 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
| ssh-hostkey: 1024 98:98:89:2b:fc:3f:09:e8:37:06:28:34:77:f0:a6:bc (DSA)
|_2048 26:f6:d3:cb:64:85:88:e3:c5:7a:9a:fc:09:84:cf:94 (RSA)
25/tcp   open  smtp
|_smtp-commands: EHLO parker.truelite.it, PIPELINING, SIZE 10240000, VRFY, ETRN,
 STARTTLS, ENHANCEDSTATUSCODES, 8BITMIME, DSN
80/tcp   open  http
|_html-title: Site doesn't have a title (text/html).
111/tcp  open  rpcbind
| rpcinfo:
| 100000  2    111/udp  rpcbind
| 100024  1  32768/udp  status
| 100000  2    111/tcp  rpcbind
| 391002  2    826/tcp  sgi_fam
|_100024  1  44427/tcp  status
631/tcp  open  ipp
2000/tcp open  cisco-sccp
6566/tcp open  unknown
MAC Address: 00:0E:A6:F4:DA:AA (Asustek Computer)

Nmap done: 1 IP address (1 host up) scanned in 2.10 seconds
```

Esistono infine altre due opzioni di `nmap` che pur essendo elencate in tab. 1.1 non effettuano una vera scansione, ma cercano solo di elencare le macchine presenti: con `-sL` viene semplicemente fatta una lista di tutti gli indirizzi che corrispondono alle specifiche degli argomenti del comando, con relativa, se presente, risoluzione inversa degli stessi. In questo caso non viene inviato nessun pacchetto verso le destinazioni finali, ma talvolta si possono ottenere informazioni interessanti anche solo per i nomi.

Leggermente più sofisticata è l'opzione `-sP` con cui `nmap` si limitata a cercare di verificare la presenza di una macchina sulla rete, senza eseguire nessuna scansione. Da questo deriva il nome *Ping scan* dato a questa opzione, in quanto lo scopo dell'operazione diventa identico a quello del

[9] su Debian è `/usr/share/nmap/scripts`

comando `ping`. Si tenga presente che tutte le volte che si richiede una scansione effettiva, viene comunque effettuato preventivamente un controllo sulla effettiva raggiungibilità di una macchina corrispondente ad un certo indirizzo; questa opzione si limita a riportare solo i risultati di detto controllo, senza poi effettuare la scansione.

Dato che la rilevazione della presenza o meno di una macchina attiva su un certo indirizzo può andare incontro a problemi, `nmap` prevede una serie di metodi di rilevazione, che possono essere attivati con le varie opzioni illustrate in tab. 1.3. L'uso dei normali pacchetti ICMP *echo request* (quelli usati da `ping`) può infatti risultare inefficace, per il fatto che molti amministratori di sistema disabilitano la risposta o filtrano questi pacchetti in varie maniere.

Opzione	Descrizione
`-P0`	Deprecato nelle nuove versioni, usare `-PN`.
`-PN`	Disabilita la rilevazione di attività.
`-PT`	Usa il *TCP ping* (deprecata in favore di `-PS` e `-PA`).
`-PS`	Usa un pacchetto TCP con flag SYN attivo.
`-PA`	Usa un pacchetto TCP con flag ACK attivo.
`-PU`	Usa un pacchetto di prova basato su UDP.
`-PE`	Usa un pacchetto ICMP *echo request* (un normale *ping*).
`-PP`	Usa una pacchetto ICMP *timestamp request*.
`-PM`	Usa una pacchetto ICMP *netmask request*.
`-PO`	Usa dei pacchetti vuoti di alcuni protocolli IP.
`-PY`	Usa un pacchetto SCTP con un *chunk* INIT minimale.
`-PR`	Usa una richiesta ARP (se il bersaglio è sulla propria LAN).

Tabella 1.3: Opzioni di `nmap` per l'indicazione delle modalità di rilevamento dell'attività di una macchina.

Con le versioni recenti del comando se il bersaglio è nella propria rete locale `nmap` esegue sempre la ricerca con una semplice richiesta di ARP, dato che questa sarebbe preliminare all'invio di qualunque altro pacchetto di rete. Anche qualora si specificassero esplicitamente delle opzioni di ricerca queste verrebbero ignorate essendo la risposta alla richiesta di ARP sufficiente a determinare la disponibilità della macchina cercata.

Qualora invece il bersaglio sia posto al di fuori della propria rete locale la ricerca viene eseguita utilizzando sia i normali pacchetti ICMP *echo request*, che attraverso il cosiddetto *TCP ping*. Nelle versioni recenti del programma se non si specifica niente viene usata una combinazione di varie tecniche: per la precisione un *ICMP echo request*, un SYN sulla porta 443, un ACK sulla porta 80, ed un *ICMP timestamp request*, corrispondenti alla combinazione di opzioni `-PE -PS443 -PA80 -PP`.

Le modalità di rilevazione possono comunque essere modificate qualunque sia la tecnica di scansione utilizzata,[10] con le opzioni illustrate in tab. 1.3, che possono essere usate anche in combinazione fra di loro. Specificando cioè una o più di queste opzioni saranno usate le relative tecniche di rilevazione, e si considererà attiva una macchina che risponda ad almeno una di esse. Si possono anche disabilitare del tutto le operazioni di rilevazione con l'opzione `-P0` (deprecata nelle versioni più recenti in favore di `-PN`).

La principale modalità alternativa di rilevazione era `-PT`, usata per indicare l'utilizzazione del cosiddetto *TCP ping*. Questo consiste nell'inviare un pacchetto TCP con il flag ACK attivo

[10] fa eccezione l'*Idle scan*, in tal caso infatti effettuare una rilevazione porterebbe a far conoscere al bersaglio il nostro indirizzo IP, quando invece si sta usando una opzione che serve a nasconderlo, pertanto in tal caso `nmap` si ferma avvisando dell'incoerenza e suggerendo l'uso di `-PN` o `-P0` a seconda della versione.

su una porta non filtrata. Se si riceve un RST in risposta significa che la macchina è attiva, e dato che in genere la porta 80 non viene mai filtrata non specificando nulla questa è quella usata di default. L'opzione funziona anche per utenti senza privilegi di amministratore, nel qual caso invece di un pacchetto ACK viene usata una normale connessione.

Nelle versioni recenti di `nmap` però `-PT` è deprecata in quanto il *TCP ping* è stato suddiviso in due diverse opzioni, `-PA`, per il cosiddetto *TCP ACK ping*, che è equivalente a `-PT`, e `-PS` per il cosiddetto *TCP SYN ping*, in cui al posto di un ACK viene usato un SYN, che riesce pertanto a passare attraverso firewall *stateful* che scarterebbe in quanto non validi i pacchetti con un ACK iniziale.

Con l'opzione `-PU` si effettua la rilevazione inviando un pacchetto UDP vuoto alla porta o alla lista di porte indicate, a meno di non specificare una lunghezza di dati con l'opzione addizionale `--data-length`, valida anche per `-PO`. Il protocollo prevede che se la porta non è aperta venga emesso in risposta un pacchetto ICMP *port unreachable* in risposta. Dato che molti servizi non rispondono quando ricevono pacchetti vuoti il modo più affidabile per utilizzare questa tecnica è di utilizzare porte che si sanno essere chiuse, in modo da avere il messaggio di errore in risposta. La tecnica infatti serve a rilevare l'attività di una macchina, non l'apertura di una porta (per questo di default viene usata la porta 40125 che è altamente improbabile venga usata da qualche servizio). L'assenza di risposte comunque non è significativa, dato che un firewall può filtrare i pacchetti UDP.

Con il supporto per il protocollo SCTP le versioni recenti consentono di effettuare una rilevazione sfruttando anche questo con l'opzione `-PY`, in questo caso viene inviato un pacchetto minimale di tipo INIT sulla porta 80, se questa è chiusa verrà risposto con un pacchetto di ABORT altrimenti verrà inviato INIT-ACK,[11] in entrambi i casi `nmap` potrà concludere che la macchina è attiva.

In tutti questi controlli eseguiti tramite porte si può richiedere, al posto del default, l'uso di un numero di porte qualsiasi da aggiungere come parametri in forma di lista separate da virgole di numeri di porta singoli, o di intervalli separati con il carattere "`-`". Un esempio di uso di queste opzioni potrebbe essere allora "`-PS80,25,443`" o "`-PA110,20-23`". La sintassi richiede che non vi siano spazi fra l'opzione e la susseguente lista, né all'interno di questa.

Sulle versioni recenti del comando è inoltre presente l'opzione `-PO` con si può effettuare la rilevazione inviando dei pacchetti vuoti di alcuni protocolli incapsulati in IP; se non si specifica nulla vengono usati dei pacchetti vuoti dei protocolli ICMP, IGMP e IPIP, ma si può specificare una lista per numeri o nome. I pacchetti generati, a parte ICMP, IGMP, TCP ed UDP, contengono soltanto l'indicazione del protocollo a livello di IP e nessuna ulteriore intestazione. Il rilevamento avviene qualora si riceva una risposta dello stesso protocollo o un pacchetto ICMP *protocol unreachable*.

Le altre tecniche di rilevamento utilizzano tutte pacchetti ICMP: con l'opzione `-PE` si attiva l'uso del classico ICMP *echo request*, come alternative si possono usare `-PP` che utilizza un ICMP *timestamp request* e `-PM` che utilizza un ICMP *netmask request*. Per ciascuno di questi pacchetti lo standard prevede un opportuno pacchetto di risposta, che indica che la macchina interrogata è attiva. Il problema comune di queste tecniche è che molti firewall non fanno passare questo tipo di pacchetti, rendendo impossibile la rilevazione.

[11] se il sistema operativo della macchina su cui gira `nmap` supporta il protocollo (come avviene con Linux), il kernel genererà una direttamente un risposta di ABORT.

Parametro	Descrizione
`Paranoid`	Esegue una scansione serializzata, con intervalli di 5 minuti fra l'invio di un pacchetto ed un altro, per cercare di sfuggire alla rilevazione di eventuali NIDS eseguendo la scansione in modalità estremamente lenta.
`Sneaky`	È analoga alla precedente, ma l'intervallo fra i pacchetti è impostato a 15 secondi, rendendola un po' meno insopportabilmente lenta.
`Polite`	Cerca di eseguire una scansione in maniera *gentile*, evitando di caricare in maniera eccessiva la rete bersaglio: serializza i pacchetti c con un intervallo di 0.4 secondi fra l'uno e l'altro.
`Normal`	É l'opzione di default, parallelizza la scansione, con l'obiettivo di essere il più veloci possibile ma al cercando al contempo di mantenere limitato il carico sulla rete bersaglio.
`Aggressive`	Questa opzione rende alcune scansioni (in particolare il *SYN scan*) nettamente più veloci, ma carica in maniera massiccia la rete.
`Insane`	Mira alla massima velocità possibile, anche a scapito dell'accuratezza, richiede una connessione molto veloce.

Tabella 1.4: Valori dei parametri per l'opzione `-T` di `nmap`.

Un'altra classe di opzioni è quella relativa al controllo delle temporizzazioni utilizzate nell'eseguire una scansione, (le attese per le risposte, i *timeout*, la frequenza dell'emissione dei pacchetti, ecc.). L'elenco completo delle opzioni, che permettono un controllo completo di tutte le tempistiche, è disponibile nella pagina di manuale, fra queste ci limitiamo a segnalare solo l'opzione generica `-T` che permette di specificare, attraverso un ulteriore parametro fra quelli indicati in tab. 1.4, una serie di *politiche* di scansione, che impostano dei valori predefiniti per le varie temporizzazioni, secondo quanto descritto nella tabella citata.

Infine `nmap` supporta una serie di opzioni generali, non classificabili in nessuna delle categorie precedenti, le più significative delle quali sono riportate in tab. 1.5. Molte di queste sono volte ad attivare funzionalità che rendano più difficile riconoscere l'esecuzione di un *portscan*, o la sorgente dello stesso. Al solito per la documentazione completa si può fare riferimento alla pagina di manuale.

Una prima opzione generale che vale la pena di citare è `-p`, che permette di indicare l'insieme di porte da controllare, l'opzione richiede un parametro ulteriore che specifichi detto insieme la cui sintassi prevede sia una lista di porte singole (separate da virgole) che un intervallo fra due estremi separati da un meno, che una combinazione di entrambi. Il default è controllare ogni porta fra 1 e 1024, più tutte quelle elencate nella lista dei servizi allegata al pacchetto, usualmente mantenuta in `/usr/share/nmap/nmap-services`. Qualora la scansione coinvolga sia TCP che UDP si possono specificare degli insiemi separatamente apponendo i parametri "`T:`" e "`U:`", un esempio di valore possibile è `-p T:1-48000,U:0-1024`.

Una seconda opzione significativa è `-D` che permette di specificare una lista di indirizzi IP (separati da virgole) da usare come esca per mascherare l'effettiva provenienza del *portscan*; nella lista può essere inserito il valore `ME` per indicare la posizione in cui si vuole che venga inserito il proprio indirizzo, altrimenti questa sarà scelta casualmente. Si tenga presente che i provider più attenti filtrano i pacchetti in uscita con indirizzi falsificati, per cui il tentativo può fallire; inoltre alcuni programmi di rilevazione dei *portscan* non troppo intelligenti reagiscono

automaticamente bloccando il traffico verso gli IP da cui vedono provenire un *portscan*, il che può permettervi di causare un *Denial of Service*, che può diventare ancora più pesante se fra gli indirizzi suddetti immettete anche il localhost.

Opzione	Descrizione
`-f`	attiva la frammentazione dei pacchetti usati per gli scan su TCP. L'intestazione del protocollo viene suddivisa in tanti pacchetti molto piccoli per cercare di sfuggire alla rilevazione da parte di eventuali NIDS o al filtro dei firewall.[12]
`-v`	aumenta le informazioni stampate e tiene al corrente dello stato della scansione, usata una seconda volta stampa ancora più informazioni.
`-A`	abilita le funzionalità che la pagina di manuale identifica come "*addizionali, avanzate o aggressive*" (a seconda dei gusti). In sostanza una abbreviazione per una serie di altre opzioni come `-sV` o `-O`.
`-6`	abilita il supporto per IPv6.
`-p`	permette di specificare una lista delle porte da analizzare nella scansione.
`-D`	permette di specificare una lista di indirizzi IP da usare come esca per mascherare l'effettiva provenienza del *portscan*.
`-g`	permette di impostare la porta sorgente (da specificare come parametro) nei pacchetti usati per la scansione.
`-n`	blocca la risoluzione degli indirizzi (e la presenza di tracce sui DNS dei bersagli).

Tabella 1.5: Principali opzioni generiche di `nmap`.

Un'ultima nota specifica va dedicata all'opzione `-O`, che attiva il meccanismo per l'identificazione del sistema operativo del bersaglio attraverso la rilevazione della cosiddetta *TCP/IP fingerprint*; un esempio di questo tipo di scansione (dove si è usata anche l'opzione `-v` per incrementare la stampa delle informazioni ottenute) è il seguente:

```
# nmap -O -v 192.168.1.141
Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2004-12-01 15:25 CET
Initiating SYN Stealth Scan against localhost (127.0.0.1) [1663 ports] at 15:25
Discovered open port 25/tcp on 127.0.0.1
Discovered open port 22/tcp on 127.0.0.1
Discovered open port 631/tcp on 127.0.0.1
The SYN Stealth Scan took 0.18s to scan 1663 total ports.
For OSScan assuming port 22 is open, 1 is closed, and neither are firewalled
Host localhost (127.0.0.1) appears to be up ... good.
Interesting ports on localhost (127.0.0.1):
(The 1660 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
```

[12] questo non è efficace con NIDS e firewall evoluti come `snort` o il *netfilter* di Linux che riassemblano i pacchetti prima di esaminarli; inoltre questi pacchetti di dimensioni ridotte possono causare il crash di quei sistemi operativi non in grado di gestirli.

```
Uptime 11.871 days (since Fri Nov 19 18:31:51 2004)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=4416598 (Good luck!)
IPID Sequence Generation: All zeros

Nmap run completed -- 1 IP address (1 host up) scanned in 2.332 seconds
```

In sostanza `nmap` esegue una analisi delle risposte inviate ad una serie di pacchetti di prova per identificare le caratteristiche dello stack TCP/IP del sistema che risponde, identificandolo rispetto ad una serie di risposte note. Come si vede l'opzione permette di determinare (dai tempi inseriti nei pacchetti) l'uptime del sistema analizzato, di verificare il meccanismo di generazione del campo *fragmentation ID* e capire se il bersaglio è utilizzabile per un *Idle scan* (qualora venga riportato il valore `incremental`). Infine l'opzione misura la predicibilità dei numeri di sequenza dello stack TCP, un indice di quanto può essere difficoltoso creare dei pacchetti artefatti per inserirsi su una connessione.

1.2.2 Gli *sniffer*

Uno *sniffer* è un programma utilizzato per leggere ed analizzare tutto il traffico di rete che arriva ad una certa macchina. In genere uno sniffer è utilizzato da chi può avere acceso alla rete locale per osservarne il traffico. Il nome è dovuto all'uso di questi programmi per "*annusare*" le password inviate in chiaro sulle connessioni, ma oggi il loro utilizzo in tal senso (con la diffusione di SSH e della cifratura delle connessioni) è drasticamente ridotto.

In compenso uno *sniffer* resta uno strumento indispensabile per l'analisi del traffico su una rete, ed in genere un NIDS non è altro che uno sniffer molto sofisticato in grado di individuare automaticamente il traffico sospetto. Ma al di là della sicurezza uno *sniffer* ha moltissimi altri impieghi, a partire da quello didattico di poter mostrare il flusso dei pacchetti per capire il funzionamento dei protocolli di rete, a quello di individuare le fonti di maggior *rumore* su una rete, ed isolarle e rimuoverle per aumentarne l'efficienza.

In genere uno *sniffer* opera direttamente al livello di collegamento fisico della rete, che nella stragrande maggioranza dei casi è realizzato attraverso il protocollo *ethernet*. Una delle caratteristiche delle schede ethernet è che esse di norma leggono soltanto il traffico diretto a loro (che corrisponde cioè al *MAC address* della scheda) o inviato in *broadcast* su tutta la rete locale. Le schede sono però anche in grado di lavorare nella cosiddetta *modalità promiscua* (il cosiddetto "*promiscuous mode*" dell'interfaccia) leggendo tutto il traffico che vedono passare.

Inizialmente le reti locali basate su ethernet erano costituite da tante schede interconnesse fra loro attraverso un unico cavo BNC, sostituito in seguito da singoli cavi agganciati ad un *hub*;[13] la caratteristica di entrambe queste configurazioni era che una volta che una scheda inviava un pacchetto sulla rete locale questo veniva reinviato a tutte le schede che ne facevano parte. Pertanto leggendo tutto il traffico in arrivo su una singola scheda si poteva leggere anche quello relativo a comunicazioni fra macchine diverse dalla propria.

[13] si chiamano *hub* quegli apparati di rete in grado di ricevere il traffico da ciascuna scheda e ritrasmetterlo a tutte le altre, la cui sola differenza rispetto al vecchio cavo BNC è quella di replicare il segnale su tutte le schede, senza bloccare tutto in caso di rottura del cavo; oggi sono praticamente scomparsi.

Con l'evolversi della tecnologia però gli *hub* ormai sono stati totalmente sostituiti dagli *switch*,[14] che sono in grado di inoltrare il traffico direttamente da una scheda di rete all'altra, evitando che ad una scheda arrivino pacchetti di rete non indirizzati esplicitamente a lei.

Dato che con l'uso degli *switch* una macchina riceve solo il traffico direttamente diretto a lei, è diventato abbastanza difficile utilizzare uno *sniffer* come strumento diagnostico per tutta una rete, non essendo più possibile accedere direttamente a tutto il traffico che passa su di essa. Può essere possibile farlo ancora se è disponibile la capacità (provvista solo dagli *switch* di fascia alta) di effettuare il *mirroring* del traffico su una porta (cosa che normalmente comporta problemi di prestazioni) altrimenti per poter leggere il traffico diretto ad altri tutto si dovrà usare la macchina con lo *sniffer* come *ponte*[15] (tramite il supporto per il *bridging* del kernel) in una posizione che le permetta di osservare il traffico che interessa.[16]

Vista tutte queste difficoltà nell'intercettare il traffico su reti basate su *switch* si potrebbe pensare che questo abbia avuto la conseguenza di rendere il compito quasi impossibile per eventuali attaccanti; questo purtroppo non è vero, in quanto esistono tecniche con cui è possibile saturare uno *switch*, (in sostanza basta inviare un sufficiente numero di falsi valori per MAC address e saturare la tabella di corrispondenza fra porte e schede) e farlo comportare come un *hub*. E se questo è in genere relativamente semplice ed efficace per un attaccante, non lo è per nulla per l'utilizzo legittimo della rete, che così viene a perdere efficienza nella trasmissione dei dati. Inoltre, come vedremo più avanti, esistono tecniche alternative che consentono comunque di intercettare il traffico fra altre macchine nella stessa rete locale senza dover importunare lo *switch*.

Esistono tuttavia degli *switch* più sofisticati che possono prevenire questo tipo di comportamento, e bloccare l'uso di una porta da parte dei pacchetti ethernet il cui indirizzo fisico non sia quello stabilito. Questo però ha il costo di una maggiore difficoltà di configurazione e della necessità di gestire in prima persona i cambiamenti dovuti alla sostituzione delle schede o allo spostamento dei computer. In ogni caso può essere opportuno installare un programma come `arpwatch` (vedi sez. 1.2.4) che è in grado di tenere sotto controllo una intera rete ethernet e segnalare l'apparire di un nuovo indirizzo e tutti gli eventuali cambiamenti nelle corrispondenze fra MAC address e numeri IP.

Il primo e più elementare *sniffer* utilizzabile su GNU/Linux (ed in moltissimi altri sistemi) è `tcpdump`. Il programma è nato come strumento di controllo di una rete, ed è in sostanza una interfaccia a riga di comando per la libreria libpcap (nome che deriva da *Packet Capture*) che è quella che viene utilizzata anche da tutti gli altri *sniffer* e che esegue il lavoro di cattura dei pacchetti. Dovendo modificare le proprietà dell'interfaccia ed usare l'accesso ai livelli più bassi

[14] anche questo è un dispositivo hardware in grado di connettere varie schede ethernet in una unica rete, ma in maniera più efficiente, creando una tabella dei MAC address associati alle schede collegate su ciascuna porta, ed inviando su detta porta soltanto il traffico ad essa relativo.

[15] in genere in una rete ethernet si chiama *ponte* (*bridge* in inglese) un dispositivo in grado di collegare fra loro due segmenti di rete separati in modo da farli apparire come una rete unica; un *ponte* è anche in grado di sapere quali indirizzi fisici sono situati fra le sue due sponde e gestire opportunamente il passaggio dei pacchetti da una parte all'altra, tenendo conto anche della presenza di eventuali altri ponti (e di diversi possibili cammini di arrivo) nella rete completa.

[16] un risultato analogo si può ottenere inserendo la macchina con lo *sniffer* su un *hub*, posto che se ne possieda ancora uno, a cui collegare tutte le macchine di cui si vuole controllare il traffico, avendo la consapevolezza che questo degraderà notevolmente le prestazioni della rete.

dei protocolli di rete, è chiaro che uno *sniffer* può essere utilizzato soltanto se si hanno i privilegi di amministratore.

Se lanciato senza parametri `tcpdump` imposta in modalità promiscua la prima interfaccia ethernet che trova attiva ed inizia a stampare a video le intestazioni di tutti i pacchetti che vede arrivare fintanto che non lo si interrompe con un SIGINT (cioè premendo `C-c`), nel qual caso stamperà anche una serie di statistiche relative ai pacchetti che è stato in grado di processare. Qualora infatti il traffico sia troppo elevato il programma scarterà quelli che non riesce a trattare.

Usando l'opzione `-w` si possono salvare i pacchetti letti su un file, passato come argomento, così da poterli analizzare in un secondo momento; l'opzione richiede come argomento il nome del file o "-" per indicare lo *standard output*. Quando si catturano pacchetti su un file è in genere il caso di usare anche l'opzione `-s` per indicare al comando di catturare anche i dati in essi contenuti; di default infatti `tcpdump` cattura solo i primi 68 byte, che in genere sono sufficienti per analizzare le intestazioni dei principali protocolli, ma che non bastano se si vuole ricostruire l'intero flusso di dati. L'opzione richiede che si passi come argomento il numero massimo di byte da catturare, ed il valore nullo può essere usato per richiedere la cattura di tutti i dati.

Usando l'opzione `-c` si può specificare un numero totale di pacchetti, letti i quali il comando terminerà. Con l'opzione `-r` si possono leggere i pacchetti precedentemente salvati su un file (da specificare come parametro) invece che dalla rete. Con l'opzione `-n` si richiede di non effettuare la risoluzione inversa degli indirizzi, che oltre a rallentare il programma causa a sua volta traffico di rete per le richieste al DNS.

Infine con l'opzione `-i` si può richiedere al comando di ascoltare su una specifica interfaccia di rete da indicare con lo stesso nome (passato come parametro) con cui viene mostrata da `ifconfig`. L'opzione `-i` può essere ripetuta più volte per ascoltare su più interfacce, o si può usare il nome `any` per richiedere l'ascolto su tutte le interfacce disponibili, questa opzione però comporta che la cattura non venga eseguita ponendo l'interfaccia in modo promiscuo. Le altre opzioni più rilevanti di `tcpdump` sono state riportate in tab. 1.6. Al solito l'elenco completo ed una descrizione dettagliata delle stesse è disponibile nella relativa pagina di manuale, accessibile con `man tcpdump`.

La caratteristica più importante di `tcpdump` è che consente di indicare, specificando come argomento del comando una espressione di filtraggio, quali pacchetti devono essere catturati e visualizzati fra tutti quelli che transitano per l'interfaccia, in modo da non essere sommersi nel normale traffico di rete e selezionare solo quanto interessa analizzare. Dato che questa funzionalità è stata implementata per la prima volta su BSD essa viene anche detta *Berkley Packet Filter* e talvolta viene indicata con la sigla BPF.

In realtà questa funzionalità è fornita direttamente dalla libreria libpcap, che contiene il codice che realizza effettivamente la *cattura* dei pacchetti sulle interfacce di rete. Questo significa che le espressioni del *Berkley Packet Filter* vengono interpretate direttamente dalla libreria, e quindi restano le stesse per tutti quegli *sniffer* (o altri programmi, come `snort`) che si appoggiano a libpcap per la lettura dei pacchetti.

Vale pertanto la pena esaminare un po' più in profondità la sintassi delle regole del *Berkley Packet Filter* dato che questo consente di specificare espressioni generiche che restano valide tanto per `tcpdump` come per molti altri programmi che vedremo in seguito.

Le espressioni di filtraggio sono composte da una o più *primitive*, cioè delle espressioni elementari che permettono di identificare una certa classe di traffico di rete, sulla base dei criteri

Opzione	Descrizione
`-c` *`count`*	legge esattamente `count` pacchetti.
`-e`	stampa anche i dati relativi al protocollo di collegamento fisico (il MAC address).
`-i` *`iface`*	specifica una interfaccia su cui ascoltare.
`-n`	non effettua la risoluzione degli indirizzi.
`-p`	non porta l'interfaccia in modo promiscuo.
`-q`	diminuisce le informazioni stampate a video.
`-r` *`file`*	legge i pacchetti dal file `file`.
`-s` *`bytes`*	cattura il numero di byte specificato per ciascun pacchetto, un valire nullo richiede la cattura di tutto il pacchetto.
`-t`	non stampa la temporizzazione dei pacchetti.
`-v`	aumenta le informazioni stampate a video.
`-w` *`file`*	scrive i pacchetti letti sul file `file`.
`-C` *`size`*	crea un nuovo file con numero progressivo ogni qual volta i dati acquisiti superano la dimensione *`size`* (indicata in milioni di byte).
`-F` *`file`*	usa il contenuto di `file` come filtro.

Tabella 1.6: Principali opzioni di `tcpdump`.

di corrispondenza che esprimono. Tutto il traffico per cui la condizione espressa dal filtro è vera verrà catturato, i restanti pacchetti non verranno considerati.

Ciascuna *primitiva* è composta da un *identificatore* (che può essere un nome o un numero) preceduto da uno o più *qualificatori* che indicano cosa sia il nome o il numero espresso dall'*identificatore*. I *qualificatori* si possono classificare in tre diversi tipi:

di tipo specifica il tipo dell'*identificatore* seguente; sono tipi possibili solo i *qualificatori* "`host`", "`net`" e "`port`" che identificano rispettivamente un indirizzo IP, una rete o una porta, ad esempio "`host davis`", "`net 192.168.2`" o "`port 25`". Qualora non si specifichi nessun tipo si assume che si tratti di un `host`.

di direzione specifica una direzione per il traffico che si intende selezionare, le direzioni possibili sono "`src`", "`dst`", "`src or dst`" e "`src and dst`", ad esempio "`src davis`" o "`dst net 192.168.2`". Qualora non si specifichi nulla si assume che si tratti di `src and dst`.

di protocollo specifica un protocollo di rete a cui limitare la corrispondenza in una regola, i protocolli possibili sono "`ether`", "`fddi`", "`tr`", "`ip`", "`ip6`", "`arp`", "`rarp`", "`decnet`", "`tcp`" e "`udp`"; possibili esempi sono "`arp net 192.168.2`", "`ether dst davis.truelite.it`" o "`tcp port 22`". Qualora non si specifichi nulla verranno utilizzati tutti i protocolli compatibili con il tipo di *identificatore* utilizzato (ad esempio se si specifica `host davis` si sottintende `ip or arp or rarp`).

Oltre ai valori precedenti si hanno disposizione delle ulteriori parole chiavi speciali come `gateway` e `broadcast`, la prima serve per indicare un pacchetto che usa come gateway l'`host` specificato di seguito, mentre la seconda serve per selezionare i pacchetti *broadcast* inviati sulla rete. Altre due espressioni speciali sono `less` e `greater` che permettono di effettuare selezioni sulla dimensione (rispettivamente minore o maggiore del valore specificato).

La potenza delle espressioni di filtraggio di libpcap sta però nel fatto che le primitive supportano anche degli operatori logici e aritmetici e possono essere combinate fra di loro in maniera flessibile e potente. Abbiamo già intravisto l'uso delle delle espressioni logiche `or` e `and`, cui si aggiunge anche la negazione ottenibile con `not`. Qualunque primitiva può essere combinata con queste espressioni, inoltre queste stesse espressioni logiche possono anche essere formulate anche in una sintassi simile a quella del linguaggio C, nella forma "`||`", "`&&`" e "`!`". Le primitive inoltre possono essere raggruppate attraverso l'uso delle parentesi "`(`" e "`)`"[17] ed anche a detti gruppi si riapplicano gli operatori logici, con il solito ovvio significato.

Un breve elenco di possibili primitive è il seguente, per un elenco più lungo e dettagliato si può di nuovo fare riferimento alla pagina di manuale di `tcpdump`:

`dst host ip` vera per i pacchetti il cui indirizzo IP di destinazione, specificato sia in notazione *dotted decimal* che in forma simbolica, è `ip`.

`src host ip` vera per i pacchetti il cui indirizzo IP sorgente è `ip`.

`host ip` vera per i pacchetti che hanno `ip` come indirizzo sorgente o di destinazione.

`dst net nw` vera per i pacchetti con indirizzo di destinazione nella rete `nw`, specificata sia con un nome presente in `/etc/network`, che in formato dotted decimal, che in formato CIDR.

`src net nw` vera per i pacchetti con indirizzo sorgente nella rete `net`.

`dst port port` vera per i pacchetti TCP e UDP con porta di destinazione `port` (specificata sia per valore numerico che nome presente in `/etc/services`).

`src port prt` vera per i pacchetti TCP e UDP con porta sorgente `port`.

`ip proto prot` vera per i pacchetti IP contenenti pacchetti di protocollo `prot`, specificato sia per valore numerico che per nome presente in `/etc/protocols`.

`tcp` abbreviazione per `ip proto tcp`.

`udp` abbreviazione per `ip proto udp`.

`icmp` abbreviazione per `ip proto icmp`.

Allora se per esempio si vogliono leggere tutti i pacchetti relativi ad una certa connessione TCP, una volta che siano noti indirizzi e porte sorgenti e destinazione, si potrà effettuare la selezione con un comando del tipo:

```
# tcpdump \( src 192.168.1.2 and src port 33005 \
and dst 192.168.1.141 and dst port 22 \) or \
\( src 192.168.1.141 and src port 22 \
and dst 192.168.1.2 and dst port 33005 \)
tcpdump: listening on eth0
16:36:53.822759 IP ellington.fi.trl.59485 > 192.168.2.2.ssh: Flags [P.], seq 48:
96, ack 97, win 400, options [nop,nop,TS val 90566618 ecr 3088734915], length 48
```

[17] attenzione che questi, come alcuni dei caratteri usati per le espressioni logiche, sono interpretati dalla shell, e devono pertanto essere adeguatamente protetti quando li si vogliono utilizzare nella riga di comando.

```
16:36:53.829040 IP 192.168.2.2.ssh > ellington.fi.trl.59485: Flags [P.], seq 97:
193, ack 96, win 158, options [nop,nop,TS val 3088743391 ecr 90566618], length 9
6
16:36:53.829061 IP ellington.fi.trl.59485 > 192.168.2.2.ssh: Flags [.], ack
193, win 400, options [nop,nop,TS val 90566619 ecr 3088743391], length 0
...
```

dove nella prima primitiva fra parentesi si selezionano i pacchetti uscenti da una macchina e diretti sull'altra, e nella seconda primitiva si selezionano i pacchetti di risposta uscenti dall'altra che tornano indietro.

L'output del programma varia a seconda del tipo di pacchetti catturati, ma per ciascuno di essi viene sempre stampata una riga iniziante con marca temporale seguita dal tipo di protocollo del contenuto. Se poi i pacchetti sono, come nel caso del precedente esempio, relativi ad un protocollo del livello di trasporto vengono sempre stampati indirizzi e porte sorgente e destinazione, nella forma:

```
nome.macchina.sorgente.numeroporta > IP.MACCHINA.DESTINAZIONE.nomeporta:
```

con tanto di risoluzione dei nomi, a meno che questa non sia stata disabilitata con l'opzione `-n`; a questo possono seguire dati specifici del protocollo, come quelli del TCP dell'esempio precedente.

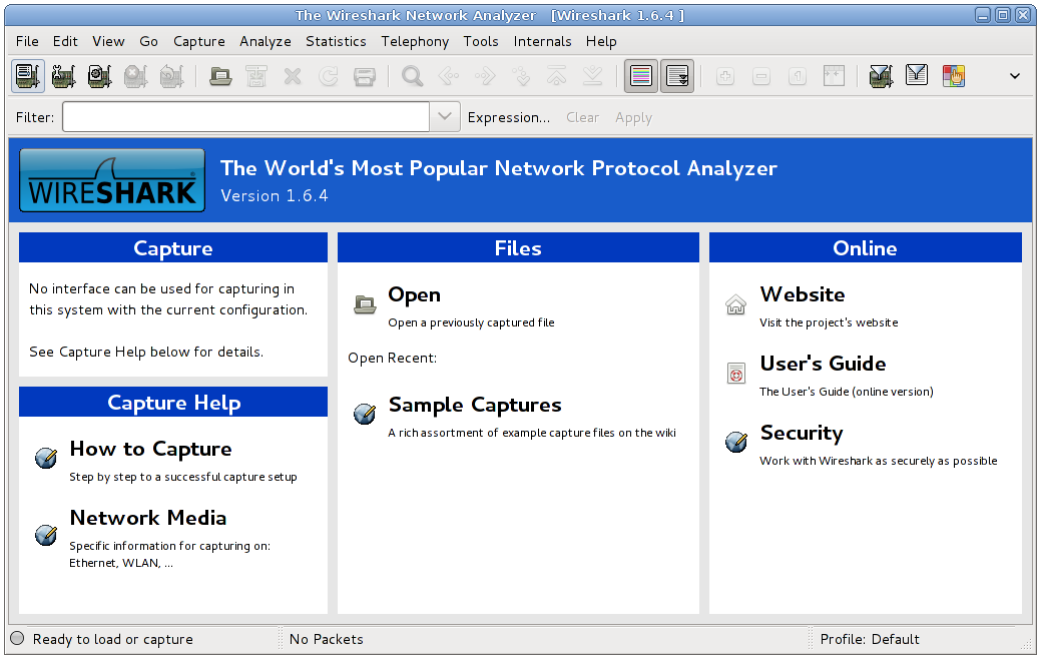


Figura 1.1: Finestra di avvio di `wireshark`.

Benché `tcpdump` offra la possibilità di analizzare nei dettagli tutto il traffico, e possa fornire tutte le informazioni necessarie, richiede anche una certa conoscenza dei protocolli di rete, dato che presenta le informazioni in una modalità non proprio di interpretazione immediata. Per

questo motivo sono stati realizzati altri *sniffer* in grado di fare lo stesso lavoro nella cattura, ma di presentare i risultati in maniera più *amichevole*. Il più importante di questi è senz'altro `wireshark`,[18] la cui schermata di avvio è mostrata in fig. 1.1.

Come si può notare in questo caso si ha a che fare con un programma dotato di una interfaccia grafica molto sofisticata, comprensiva di un sistema di *help on line*. Il programma può eseguire la stampa o il salvataggio su disco dei dati o la lettura degli stessi da precedenti sessioni di cattura, anche se eseguite da altri programmi, con una ampia capacità di decodificare i vari formati in cui questi possono essere stati memorizzati. Ma in realtà la vera potenza di `wireshark` non sta tanto nella ricchezza delle sue funzionalità quanto nella infrastruttura di cui è dotato.

La caratteristica più interessante di `wireshark` infatti è quella di mettere a disposizione una interfaccia ben definita e molto funzionale per i cosiddetti *protocol dissector*. Il programma cioè può essere esteso in maniera relativamente semplice per effettuare una vera e propria *dissezione anatomica* del traffico di rete, in modo da presentare le informazioni in maniera efficace e facilmente comprensibile, provvedendo ad identificare parecchie centinaia di protocolli diversi, dal livello di rete fino a quello di applicazione, e classificare di conseguenza i dati raccolti. In questo modo diventa poi possibile identificare, all'interno dei singoli pacchetti, tutti i dettagli ed i particolari relativi al contenuto.

Si tenga presente però che questa ricchezza di funzionalità ha un costo in termini di sicurezza, dato che eseguire una validazione corretta dei dati in ingresso per qualunque tipo di traffico è estremamente complesso, e questo porta ad una grande quantità di problemi di sicurezza per errori presenti nei *protocol dissector*. La finestra di avvio di fig. 1.1 è quella ottenuta quando il programma viene lanciato da un utente normale, quando lo si lancia con privilegi di amministratore si ottiene un'ulteriore finestra con un messaggio di avviso che ne sconsiglia l'uso dato in questo caso qualcuno potrebbe, sfruttando una di queste vulnerabilità ed inviando pacchetti opportunamente malformati, compromettere la macchina.

Il problema è che per poter effettuare la cattura dei pacchetti i privilegi di amministratore sono necessari, per cui non esiste una soluzione semplice al problema che non sia quella di eseguire la cattura in maniera indipendente salvando i dati su un file, ed eseguire il programma in un secondo tempo per analizzare gli stessi. Il comando `wireshark` infatti prende come argomento un nome di file da cui leggere i dati, ed è in grado di leggere una grande varietà di diversi formati, riconoscendoli automaticamente. Si può comunque caricare un file in un qualunque momento, con la voce Open nel menù File o cliccando sulle icone della sezione *File* della finestra di avvio.

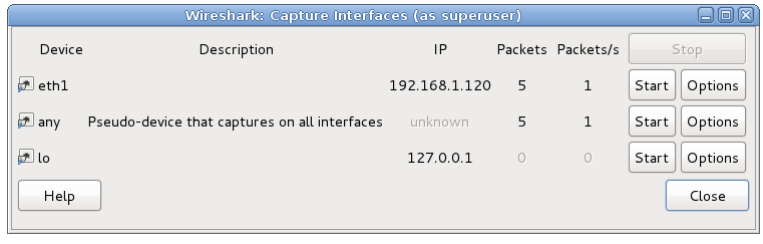


Figura 1.2: Finestra di selezione delle interfacce di `wireshark`.

[18] in precedenza il programma era noto come `ethereal`, il nome è stato cambiato per questioni relative ai marchi.

Qualora lo si voglia usare in maniera interattiva occorrerà invece assumersi il rischio di subire un attacco ed accettare di andare avanti sulla finestra di avvio. A questo punto per far partire una sessione di cattura basterà premere su una delle interfacce mostrate nella sezione *Interface List* della finestra di avvio (presenti solo quando lo si invoca come amministratore), altrimenti si potrà ottenere una finestra di selezione delle interfacce, illustrata in fig. 1.2, cliccando sulla prima icona a sinistra nella barra sotto il menù o selezionando la voce Interfaces nel menù Capture.

La finestra di selezione riporta tutte le interfacce rilevate da `wireshark` elencate con i loro nomi, più una interfaccia virtuale che le accorpa tutte, chiamata `any` come per `tcpdump`. Nella finestra viene anche mostrata tempo reale una statistica del flusso di pacchetti sulle varie interfacce. Si può far partire una cattura premendo sul pulsante Start corrispondente all'interfaccia scelta, ma si possono anche impostare le opzioni di cattura premendo sul pulsante Options, che farà apparire la apposita finestra, illustrata in fig. 1.3. Questa stessa finestra può essere ottenuta direttamente premendo sulla seconda icona nella barra sotto il menù o o selezionando la voce Options nel menù Capture.

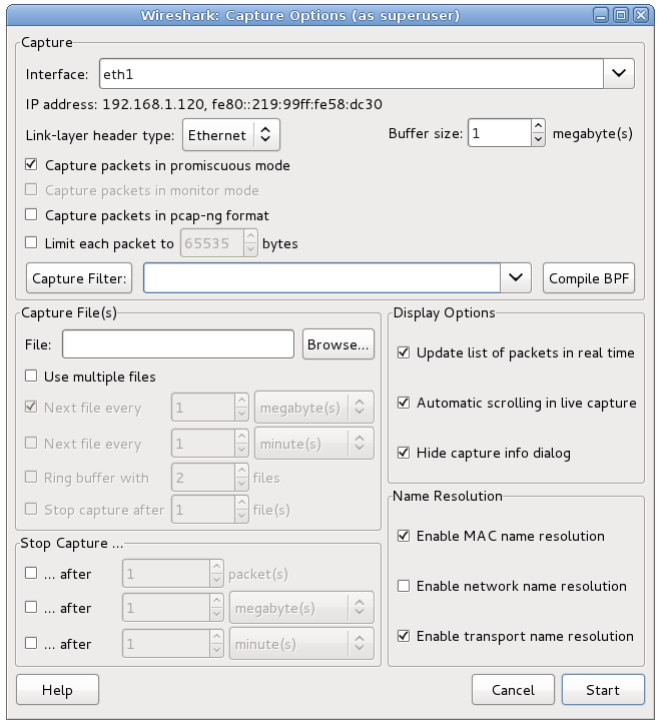


Figura 1.3: Finestra di impostazione delle opzioni di cattura di `wireshark`.

Da questa finestra si possono impostare i parametri della sessione di cattura. In particolare è possibile riscegliere l'interfaccia su cui ascoltare nel campo di testo Interface (dotato anche di tendina per la selezione fra quelle disponibili identificate dal programma), si può indicare un file su cui salvare i dati nel campo di testo File (dotato di pulsante per attivare la finestra di selezione dei file), ma soprattutto si può inserire una espressione di filtraggio per selezionare i pacchetti da

catturare nel campo di testo Capture Filter, usando la sintassi del *Berkley Packet Filter* illustrata in precedenza. Inoltre premendo sul bottone associato si può salvare o selezionare il filtro su di una tabella, assegnandogli un nome in modo da poterlo riutilizzare in seguito senza doverlo riscrivere.

Il resto della finestra consente di impostare tutte le altre caratteristiche della sessione di cattura, come da relative indicazioni auto-esplicative. Una volta completate le impostazioni si potrà far partire la sessione di cattura premendo sul pulsante Start.

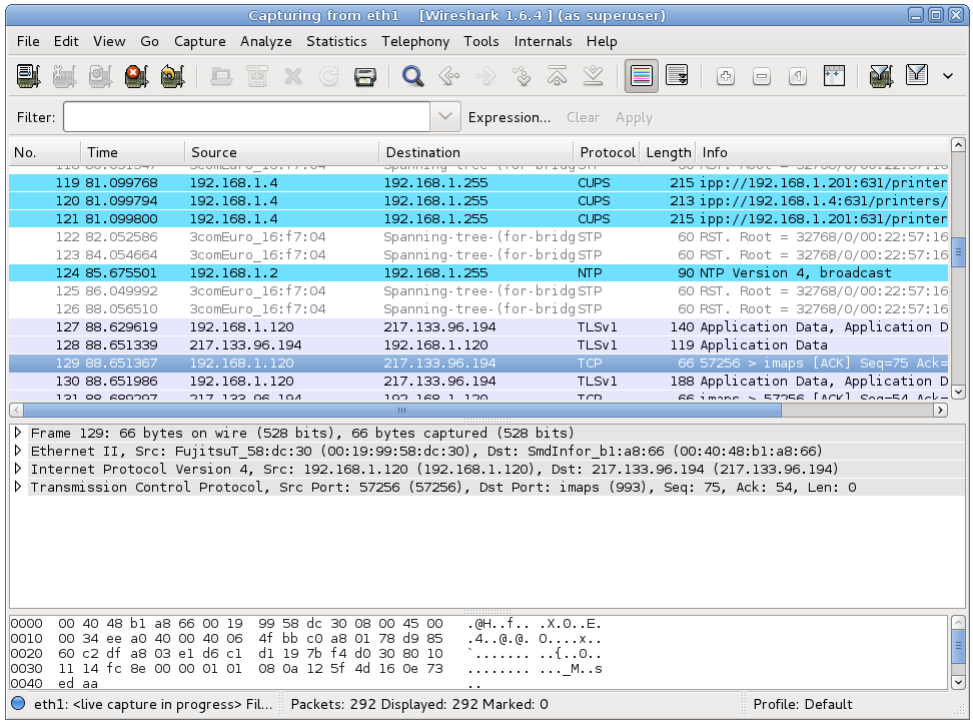


Figura 1.4: Finestra principale di `wireshark`.

Una volta avviata la cattura il programma inizierà a mostrare i pacchetti ricevuti nella sua finestra principale, riportata in fig. 1.4, le cui righe inizieranno a scorrere progressivamente. A meno di non aver impostato dei limiti nelle opzioni di cattura (la sezione Stop Capture in fig. 1.3), questa proseguirà indefinitamente e dovrà essere fermata esplicitamente o premendo sul pulsante di stop (la quarta icona della barra sotto il menù) o selezionando la voce Stop dal menù `Capture`.

Tutti i dati vengono inseriti nella finestra principale che è divisa in tre sezioni principali. Nella prima parte in alto viene mostrata la lista dei pacchetti catturati, numerati progressivamente. Di questi viene riportato il tempo di cattura relativo all'inizio della sessione, gli indirizzi destinazione e sorgente (IP, a meno che non si tratti di pacchetti ARP) il protocollo ed un breve sunto delle informazioni contenute. Per visualizzare il contenuto di un pacchetto particolare basta selezionarlo con il mouse, e nella sezione centrale comparirà il risultato della sua *dissezione*, mentre nella sezione finale verrà visualizzato il contenuto del pacchetto espresso in valori

esadecimali e in caratteri ASCII. Un esempio è mostrato in fig. 1.5 dove si è selezionata la parte relativa al protocollo TCP.

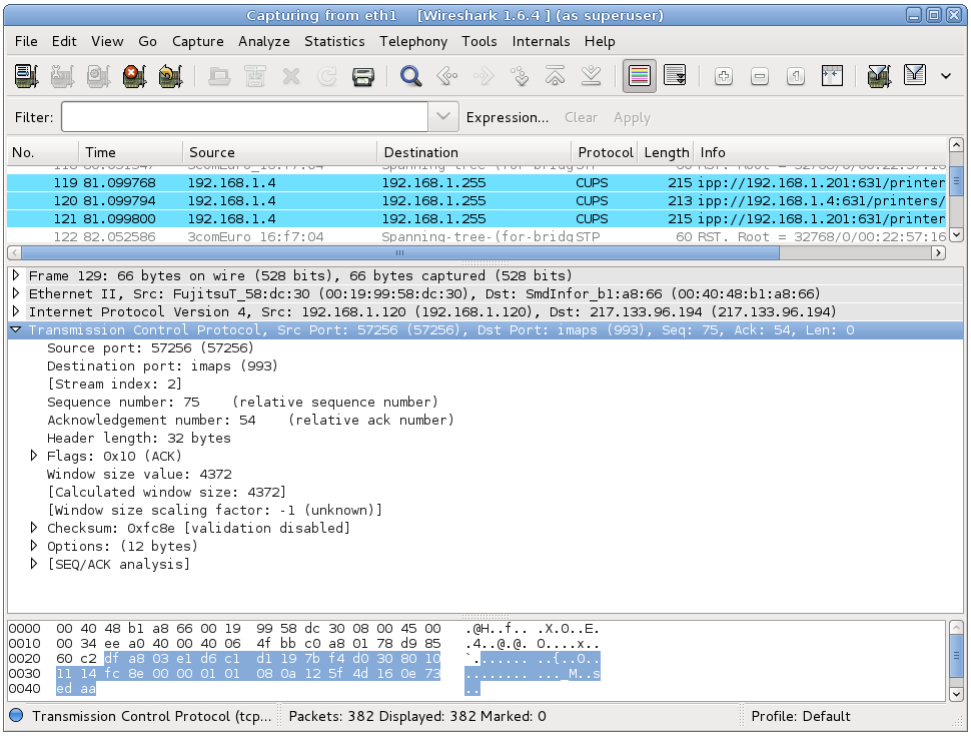


Figura 1.5: Finestra principale di `wireshark`, nella selezione di una parte del pacchetto.

Sopra la tre sezioni viene riportato un campo testo che permette di inserire un ulteriore filtro di selezione. Questo non va confuso con il precedente filtro di cattura, che opera solo in fase di acquisizione dei pacchetti e supporta esclusivamente la sintassi del *Berkley Packet Filter*; in questo caso infatti il filtro è sulla visualizzazione dei pacchetti catturati, e permette di utilizzare tutte le informazioni che il *protocol dissector* ha estratto dai pacchetti. Si può anche costruire il filtro direttamente con l'interfaccia grafica, premendo sul relativo pulsante, che farà comparire una apposita finestra di composizione e salvataggio dei filtri.

Come accennato la sezione centrale della finestra di `wireshark` contiene il risultato della dissezione dei pacchetti, organizzato in una struttura a livelli, a partire dai dati relativi al *frame* ethernet letto direttamente sull'interfaccia (tempo di arrivo, numero progressivo, dimensione), ai dati relativi al protocollo di collegamento fisico, quello di rete (o equivalente posto sopra il collegamento fisico, come ARP), il livello di trasporto, quello di applicazione, ulteriori protocolli interni, ecc.

Per ciascun nuovo protocollo che viene trovato all'interno del precedente viene creata una riga in questa sezione, che porta con sé l'indicazione del protocollo cui fa riferimento, e i relativi dati principali: così per ethernet sono mostrati i MAC address destinazione e sorgente del pacchetto,

per IP i relativi indirizzi, per TCP le porte, i numeri di sequenza e di *acknowledgement*, e così via per gli ulteriori protocolli contenuti nel pacchetto.

Tutte queste righe possono poi essere espanse ad albero per esporre il contenuto completo delle intestazioni dei relativi protocolli, suddivise per i vari campi (ciascuno avrà ovviamente i propri valori). Premendo su una di queste righe, o su uno dei campi all'interno, si avrà l'effetto di evidenziare, nella sezione finale dove sono riportati i dati in forma binaria, la relativa sezione del pacchetto.

La grande flessibilità di `wireshark` comunque non si ferma qui, premendo il tasto destro del mouse su un pacchetto si può attivare un menù contestuale che permette ad esempio di identificare tutto lo stream TCP (o il flusso dati UDP) di cui questo fa parte, visualizzandone il contenuto in una nuova finestra o generare al volo un filtro sulla base delle proprietà di quel pacchetto (queste funzionalità si possono ottenere anche dalle relative voci nel menù Analyze). Se invece si clicca su uno dei campi della finestra di dissezione si può generare un filtro di selezione sulla base di quel campo.

Inoltre dal menù Statistics si hanno a disposizione una serie di funzioni statistiche che permettono di ottenere sommari sul traffico rilevato, come quello strutturato per gerarchia di protocolli (voce Protocol Hierarchy); quello che traccia tutte le conversazioni (voce `Conversations`) identificate dal programma ai vari livelli dei protocolli, quello che traccia i capi della comunicazione (voce `Endpoints`) e molti altri.

Come si può vedere anche da questa breve e sommaria introduzione `wireshark` è uno strumento di analisi estremamente potente e sofisticato. Non staremo ad approfondirne ulteriormente le caratteristiche, essendo quanto illustrato sufficiente per le esigenze di queste dispense, per una trattazione più accurata si può fare riferimento alla guida utente, disponibile su `http://www.wireshark.org/` anche in formato PDF, o esercitarsi un po' con le funzionalità dell'interfaccia utente.

L'ultimo programma che tratteremo in questa sezione è `ettercap`, che più che uno *sniffer* è uno strumento per eseguire attacchi di *man-in-the-middle*. A differenza degli altri *sniffer* infatti `ettercap` non si limita a leggere i pacchetti che arrivano su una interfaccia della propria macchina, ma è in grado di mettere in atto tutta una serie di tecniche che lo rendono in grado di intercettare (e modificare al volo) il traffico fra altre macchine, anche quando ci si trova su una rete basata su switch.

Con la nuova versione 0.7 il programma è stato radicalmente modificato e le sue funzionalità sono state ampliate, in particolare sono state previste due modalità principali di cattura dei pacchetti (*sniffing*), denominate *bridged* e *unified*. La prima, specificata a riga di comando dalla opzione `-B`, prevede che si abbiano due schede di rete con cui creare un bridge da cui far passare il traffico; in tal caso si deve avere la possibilità interporsi fisicamente, ma si ottiene il vantaggio di essere totalmente trasparenti.

Dati gli stringenti requisiti di accesso fisico per poter utilizzare la prima, la modalità usata normalmente dal programma è la seconda (nel senso che se lanciato a riga di comando viene attivata di default senza dover specificare altro) chiamata *unified* proprio in quanto consente di unificare le operazioni di cattura con quelle di attacco e con quelle di eventuale filtraggio e modifica dei pacchetti intercettati.

In questa modalità infatti `ettercap` disabilita il reinoltro dei pacchetti del kernel in quanto questo compito viene assunto dal programma stesso, che può così eseguire quanto serve per gli attacchi *man-in-the-middle* ed applicare eventuali modifiche ai pacchetti in transito. Se

la funzionalità di *IP forwarding* restasse attiva anche il kernel reinoltrerebbe i pacchetti che risulterebbero doppi. Si tenga presente che questo comportamento da luogo a problemi quando si esegue `ettercap` su un *gateway* (cosa che è comunque sconsigliata), poiché il programma opera sempre su una sola interfaccia ed esegue il reinoltro solo su quella, quindi i pacchetti in transito fra interfacce diverse non verrebbero instradati, per questo caso è disponibile un *Unoffensive Mode* (opzione `-u`) che non disabilita l'*IP forwarding*.

Se lanciato a riga di comando il comando richiede sempre due argomenti che identificano le macchine bersaglio poste ai due capi della comunicazione che si vuole intercettare. Non esiste in genere il concetto di sorgente e destinazione in quanto il flusso è bidirezionale, ma qualora si usino tecniche che consentano solo di intercettare una sola direzione della connessione, il primo dei due argomenti indica il sorgente ed il secondo la destinazione.

I bersagli sono espressi nella forma generica `MAC-addr/IP-addr/port`, qualora non si specifichi uno dei tre valori si intende che questo può essere qualsiasi, pertanto `/192.168.1.1/` indica la macchina con quell'indirizzo IP e porta qualunque, mentre `//110` indica un indirizzo IP qualunque e la porta 110. Per specificare l'indirizzo IP si può usare una sintassi estesa che consente di specificare un insieme di indirizzi, con elenchi di indirizzi o intervalli, mentre se si specifica un *MAC address* questo deve essere unico. Infine con l'opzione `-R` si può invertire la selezione dell'indirizzo specificato di seguito.

Opzione	Descrizione
`-i` *iface*	specifica una interfaccia al posto di quella di default.
`-w` *file*	scrive i pacchetti letti sul file `file` nel formato di `tcpdump` (pcap).
`-r` *file*	legge i pacchetti in formato pcap dal file `file`.
`-L` *file*	scrive i dati letti sui file `file.ecp` per i pacchetti e `file.eci` per le informazioni.
`-l` *file*	scrive solo le informazioni essenziali raccolte sul file `file.eci`.
`-k` *file*	scrive la lista delle macchine rilevate come presenti sulla rete.
`-j` *file*	legge la lista delle macchine presenti sulla rete.
`-R`	inverte la indicazione di un bersaglio (seleziona ciò che non corrisponde).
`-z`	non esegue la scansione iniziale della rete.
`-u`	attiva l'*unoffensive mode*, non disabilitando il reinoltro dei pacchetti sulle interfacce.
`-P` *plugin*	esegue il *plugin* indicato.
`-T`	esegue il programma in modalità testuale.
`-C`	esegue il programma in modalità semigrafica.
`-G`	esegue il programma in modalità grafica .
`-D`	esegue il programma come demone.
`-B` *iface*	attiva il *bridged sniffing* usando insieme l'interfaccia di default (o quella specificata con `-i`) e *iface*.
`-M` *mode*	attiva la modalità di attacco per il *man-in-the-middle* indicata da *mode* che può assumere i valori: `arp` per l'*arp poisoning*, `icmp` per l'*ICMP redirect*, `dhcp` per il *DHCP spoofing*, `port` per il *port stealing*.

Tabella 1.7: Principali opzioni di `ettercap`.

Se non si specifica nessun argomento il programma chiede di specificare tramite una opportuna opzione l'interfaccia di gestione che si intende usare. Le scelte possibili sono tre, con `-T` si attiva l'interfaccia testuale semplice, con `-C` si attiva l'interfaccia testuale semigrafica basata

sulla libreria *ncurses*, con `-G` si attiva l'interfaccia grafica basata sulle librerie GTK.[19] In genere l'uso più comune è con una di queste ultime due, ed al solito quella testuale ha il vantaggio di poter essere usata con un uso di risorse molto più ridotto.

Si può però lanciare `ettercap`, con l'opzione `-D`, anche in modalità non interattiva, in cui il programma lavora registrando tutti i risultati su dei file di log. In questo caso occorre specificare con `-L` il file su cui si vuole vengano salvati i pacchetti e dati (nel formato interno di `ettercap`). Detto file potrà essere analizzato in un secondo tempo con il programma `etterlog`, che consente di estrarre dal traffico tutte le informazioni che interessano. Quando il traffico analizzato è molto questa modalità non incorre nei rallentamenti dovuti alla interazione con l'interfaccia di gestione.

In modalità non interattiva si dovrà inoltre indicare con `-l` il file su cui si vuole che vengano salvate le informazioni più importanti ottenute dal programma, come le password intercettate. Si tenga presente che `ettercap` una volta configurata l'interfaccia per la cattura dei pacchetti lascia cadere i privilegi di amministratore per girare come utente `nobody`, per cui si devono specificare un file o una directory scrivibili da chiunque (come `tmp`) o indicare un utente alternativo che possa scrivere nella directory indicata con la variabile di ambiente `EC_UID`.

Le principali opzioni del programma, utilizzate in genere soltanto in questo caso, dato che in modalità interattiva il comportamento viene controllato tramite le operazioni effettuate sull'interfaccia di gestione, sono illustrate in tab. 1.7.

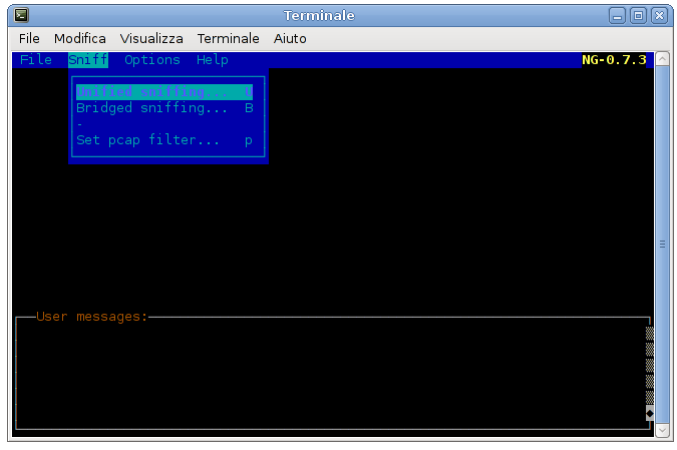


Figura 1.6: La schermata iniziale `ettercap` nella versione semigrafica.

In fig. 1.6 si è riportata la schermata iniziale che si ottiene con l'interfaccia semigrafica delle *ncurses*, che è quella che tratteremo più estensivamente. In questa schermata si devono impostare le modalità di cattura dei pacchetti, che una volta scelte si applicheranno per il resto della sessione di lavoro; dal menù Sniff si seleziona la modalità di cattura (fra *unified* e *bridged*), e dal menù Options si impostano le eventuali opzioni ulteriori (come l'*unoffensive mode*). La selezione, oltre che dai menù, può essere effettuata premendo i relativi caratteri sulla tastiera,

[19] se si è installato la versione con il relativo supporto, che su Debian ad esempio questa è disponibile solo se si è installato il pacchetto `ettercap-gtk`.

corrispondenti in genere alle lettere delle opzioni di tab. 1.7. Dal menù File si possono anche leggere dati già acquisiti, ma non tratteremo questo caso.

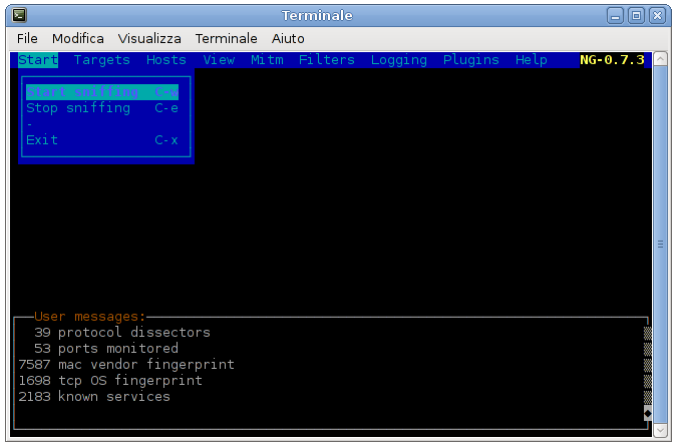


Figura 1.7: La schermata di `ettercap` per la lista delle macchine sulla rete.

Una volta scelta la modalità verrà presentata una ulteriore finestra per la selezione della (o delle) interfacce da utilizzare per la cattura, dopo di che si verrà portati nella finestra principale del programma, che si è riportata in fig. 1.7. La finestra è divisa in due parti, nella parte superiore vengono visualizzate le informazioni relative alle varie funzionalità del programma (bersagli, lista delle macchine, delle connessioni, ecc.) mentre nella parte inferiore vengono mostrati i messaggi per l'utente.

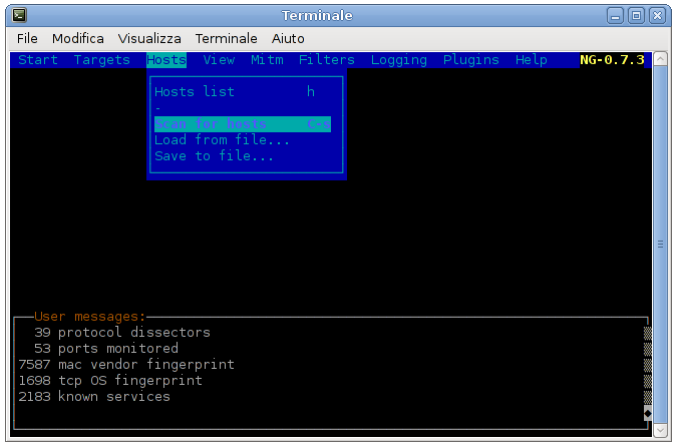


Figura 1.8: La schermata di `ettercap` con il menù per la ricerca delle macchine sulla rete.

Inoltre la prima riga della finestra contiene il menù delle operazioni su cui ci si può spostare con le frecce, e selezionare una voce posizionandosi e poi premendo l'invio. Si può passare da una finestra all'altra e ritornare sul menù utilizzando il testo di tabulazione. Si può far

partire e fermare la cattura dal menù Start, selezionare i bersagli dal menù Target, visualizzare le informazioni raccolte da View, attivare un attacco di *man-in-the-middle* da Mitm, ma in genere il primo passo è ottenere la lista delle macchine presenti sulla rete, cosa che normalmente si fa dal menù Hosts, illustrato in fig. 1.8.

E da questo menù che si può effettuare una scansione della rete per ottenere le macchine presenti, che viene eseguita inviando una richiesta di ARP per tutti gli IP della rete associata all'interfaccia su cui si sta eseguendo la cattura, vale a dire inviare una richiesta di *ARP reply* in *broadcast* per ogni IP della rete, che sarà pertanto ricevuta da tutte le macchine presenti. Questo metodo presenta però l'inconveniente di essere particolarmente "*rumoroso*" sulla rete (si parla infatti di *ARP storm*) e molti IDS sono in grado di accorgersi di questo comportamento anomalo. Per questo motivo il programma è in grado di leggere una lista delle macchine da un file o salvarla qualora la si sia ottenuta.

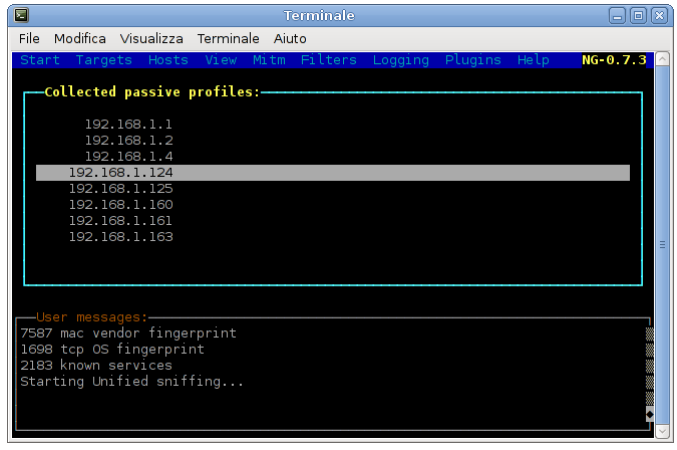


Figura 1.9: La schermata di `ettercap` che mostra l'elenco dei profili raccolti in maniera passiva.

Si tenga presente che fintanto che non si abilita la cattura dal menù Start (o con `C-w`) il programma non acquisirà nessun pacchetto, e che non è necessario selezionare i bersagli o individuare le macchine presenti per attivare la cattura; si può infatti anche far partire l'acquisizione senza altre operazioni ed il programma si limiterà ad ascoltare tutto il traffico mandando l'interfaccia in modalità promiscua, raccogliendo una serie di informazioni come IP e MAC address, sistema operativo,[20] ecc. che potranno essere visualizzate dal menù View nella voce Profiles (accessibile direttamente con il tasto "`0`"), il cui risultato è mostrato in fig. 1.9.

Da queste informazioni ottenute in modo passivo è possibile salvare una lista delle macchine rilevate premendo "`d`" (chiederà di indicare un nome di file), o convertirla direttamente per l'uso come lista di macchine premendo "`c`" (diventerà così disponibile nell'elenco della relativa pagina del menù Hosts, cui si accede direttamente anche premendo "`h`"). Si tenga presente che le informazioni raccolte comprendono tutti gli indirizzi IP rilevati nel traffico, compresi quelli remoti, in genere questi non servono nella lista da convertire come elenco di macchine, e si

[20] il programma è dotato anche un sistema di *passive fingerprinting* che usa le stesse tecniche di `nmap` per identificare i vari sistemi operativi sulla base del loro peculiare comportamento sulla rete.

possono rimuovere premendo "l", iniziale che sta per *local*, inoltre si può anche ridurre la lista ai soli indirizzi remoti premendo "r". Sempre da questa schermata si possono vedere i dettagli delle informazioni raccolte per ciascuna macchina spostandosi con le frecce per selezionarla e premendo invio.

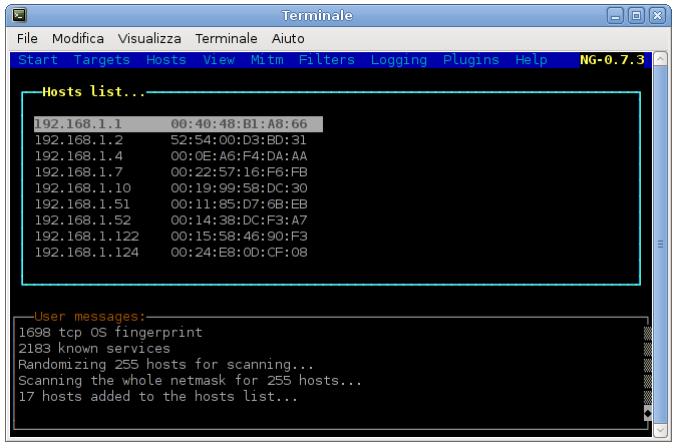


Figura 1.10: La schermata di `ettercap` la lista delle macchine sulla rete.

Una volta ottenuta la lista delle macchine, sia con una scansione diretta che convertendo le informazioni raccolte passivamente, si avrà un risultato come quello mostrato in fig. 1.10. Da questa schermata si potrà eseguire direttamente la selezione dei bersagli da attaccare spostandosi su uno degli IP elencati, e premendo i tasti "1" o "2" per selezionarlo come primo o secondo. Questa è anche la lista delle macchine che verranno prese in considerazione qualora si indichi un bersaglio generico, pertanto se si vuole evitare di attaccare una di queste macchine si può premere "d" per rimuoverla dalla lista.

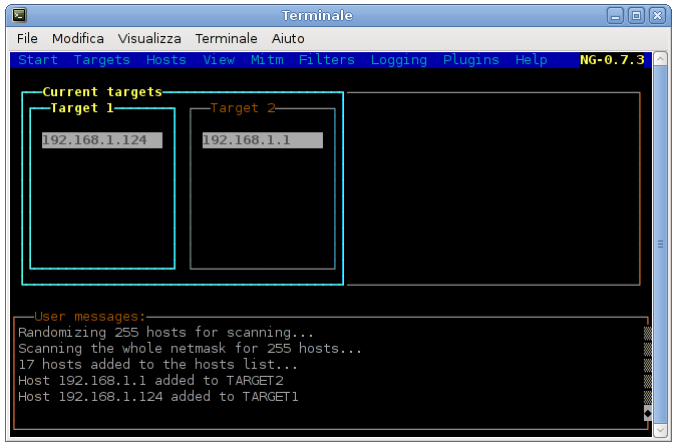


Figura 1.11: La schermata di `ettercap` con la selezione dei bersagli.

A questo punto si potrà verificare la relativa presenza dal menù `Target` (accessibile direttamente premendo "`T`"). Si otterrà così un risultato come quello mostrato in fig. 1.11. Si tenga presente che da detto menu si potranno anche impostare direttamente i bersagli con "`C-t`".

Il funzionamento di `ettercap` prevede che le varie metodologie di attacco vengano utilizzate per intercettare il traffico fra le macchine specificate come *Target 1* e quelle specificate come *Target 2*. Dal punto di vista della cattura dei pacchetti non esiste il concetto di sorgente e destinazione, dato che il traffico viene intercettato in maniera generica e quindi in entrambi i sensi, ma certi tipi di attacco consentono di interporsi solo su una direzione del traffico, in tal caso la direzione è definita e verrà intercettato il traffico da una macchina indicata come *Target 1* ad una specificata come *Target 2*.

Fintanto che non si attivano attacchi specifici `ettercap` si limiterà ad acquisire i pacchetti che vede passare sulla propria interfaccia di rete, questi possono essere anche tutti quelli che servono se si usa la modalità `bridged`, ma come detto questo implica la possibilità di una interposizione fisica.

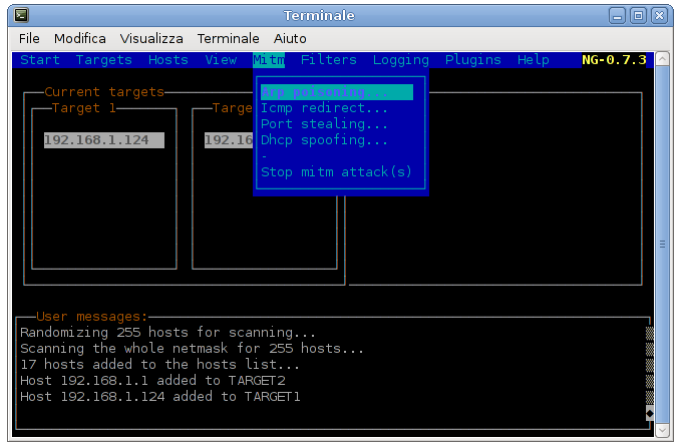


Figura 1.12: La schermata di `ettercap` con la selezione degli attacchi.

Per questo le funzionalità più interessanti di `ettercap` sono quelle che si attivano dal menù Mitm che consente di scegliere quali fra le varie tecniche per eseguire attacchi di *man-in-the-middle* disponibili deve essere usata in modo da ottenere che il traffico interessato possa arrivare al programma anche senza una interposizione fisica. Il menù è riportato in fig. 1.12, dove è stata selezionata la prima modalità di attacco; scegliendo le varie voci si possono attivare le altre.

La principale tecnica usata da `ettercap`, e la più efficace, è quella denominata *ARP poisoning* che consiste nell'inviare risposte alle richieste di ARP con il proprio MAC address, sovrascrivendo le risposte degli altri in modo da farsi inviare i pacchetti diretti a qualcun altro. Il problema risiede nell'insicurezza intrinseca del protocollo ARP, che non prevede nessun tipo di autenticazione, per cui gli indirizzi contenuti nei pacchetti ARP vengono automaticamente inseriti nella tabella di ARP del kernel.

Quello che accade normalmente cioè è che se si riceve una risposta di ARP, anche se non è stata fatta una richiesta, il kernel inserirà i relativi dati nella tabella di ARP con lo scopo di diminuire il traffico sulla rete. Allora basta mandare dei falsi pacchetti ARP con il proprio

MAC address alle due macchine di cui si vuole intercettare il traffico, indicando noi stessi come la macchina cui inviare il traffico relativo all'IP dell'altra (secondo lo schema in fig. 1.13).

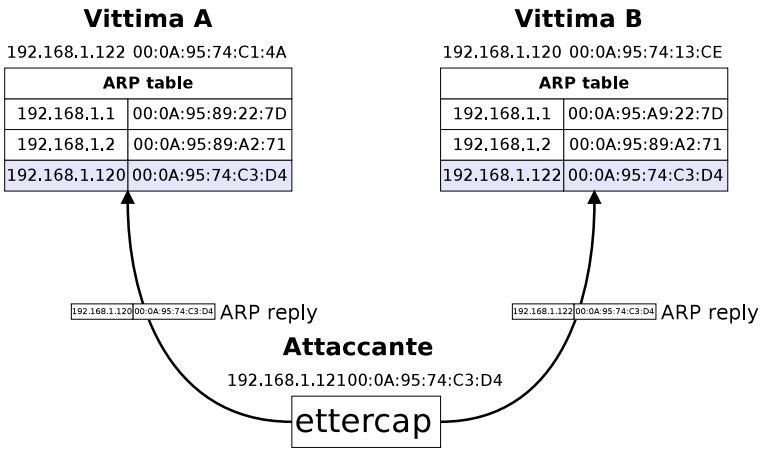


Figura 1.13: Schema di funzionamento dell'*ARP poisoning*.

Benché esistano dei tentativi di contromisura (ad esempio Linux non accetta risposte di ARP se non precedute da una richiesta fatta da lui), in realtà nessun kernel blocca le *richieste* di ARP, che quando ricevute prevedono di nuovo l'inserimento dell'indirizzo del richiedente nella propria tabella; pertanto su può tranquillamente *avvelenare* la cache di una macchina eseguendo una falsa richiesta di ARP in cui si associa il proprio *MAC address* all'IP di qualcun altro.

Questo consente a `ettercap` di *avvelenare* la cache usando semplicemente in modalità alternata falsi pacchetti ARP di risposta o di richiesta. Dato che il funzionamento di una rete prevede che tutti i pacchetti diretti ad un certo IP vengano inviati alla scheda con il corrispondente *MAC address* nella tabella di ARP, si potranno ricevere tutti i pacchetti destinati alla macchina con il suddetto IP, e se al contempo si è avvelenata pure la cache di quest'ultima, indicando di nuovo il proprio *MAC address* in corrispondenza dell'IP del vero mittente, si avrà un classico attacco di *man-in-the-middle* in cui tutti i pacchetti del traffico fra le due macchine viene intercettato da `ettercap`.

In questo modo il programma può analizzare tutto il traffico fra due macchine anche se la rete è basata su *switch* dato che i pacchetti vengono comunque mandati al *MAC address* dell'attaccante. Inoltre siccome il traffico passa dalla macchina dell'attaccante, `ettercap` è anche in grado di modificare il traffico inserendo dei dati, o cancellandoli o sostituendoli, ed il tutto avviene in maniera completamente trasparente rispetto ai protocolli del livello di rete, proprio in quanto l'operazione viene eseguita direttamente sui protocolli del livello di collegamento fisico.

Una volta effettuato l'*avvelenamento* il programma potrà (se si è attivata la cattura dei pacchetti) intercettare il traffico fra le macchine sottoposte all'attacco, ed osservare le connessioni di rete fra queste. Le connessioni riconosciute da `ettercap` vengono riportate in una tabella come quella riportata in fig. 1.14 cui si accede dalla voce Connections del menù View o premendo "`C`".

Selezionando (tramite le frecce) la connessione che si vuole intercettare e premendo invio si potrà accedere ai contenuti del traffico della stessa, presentato in due finestre affiancate per

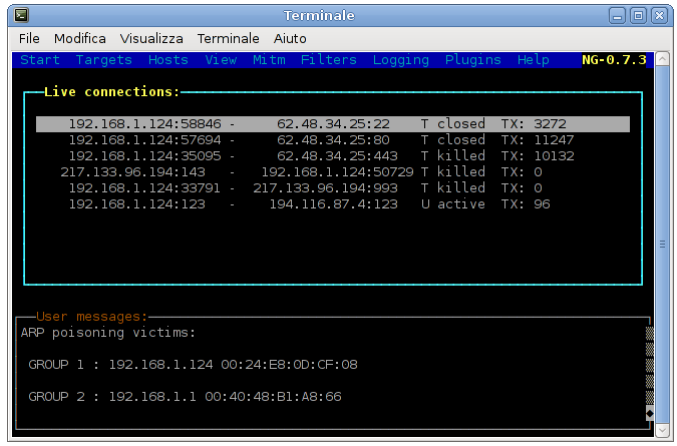


Figura 1.14: La schermata di `ettercap` con la lista delle connessioni rilevate nell'intercettazione del traffico fra due macchine.

indicare le due direzioni dello stesso (se disponibili), come mostrato in fig. 1.15. Il programma comunque è in grado di riconoscere il passaggio in chiaro delle password sui principali protocolli e quando rilevate le stampa direttamente sull'area di notifica in basso.

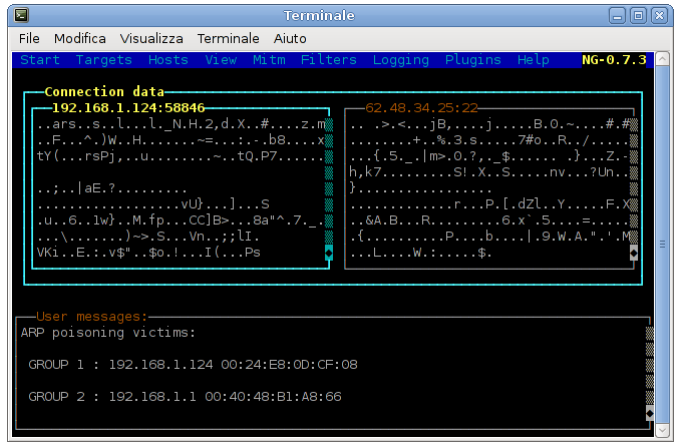


Figura 1.15: La schermata di `ettercap` con il traffico intercettato in una connessione.

Le altre tecniche di attacco sono meno efficaci, e pertanto meno utilizzate. Quella denominata *ICMP redirect* si basa sull'invio alla propria vittima dei menzionati di "*falsi*" pacchetti ICMP di questo tipo con l'IP del gateway, che indicano la propria macchina come migliore strada di uscita, in questo modo la vittima invierà il suo traffico in uscita verso `ettercap`,[21] che provvederà

[21] a meno che non sia stata configurata per non accettare questi pacchetti, questo su Linux può essere fatto scrivendo uno zero sul file `accept_redirects` dentro la sottodirectory di `/proc/sys/net/ipv4/conf/` relativa a ciascuna interfaccia.

a reinviarlo al gateway dopo averlo intercettato.

Questa tecnica consente di intercettare soltanto i pacchetti in uscita verso internet, non è possibile però ottenere le risposte, perché il gateway non accetterà mai un *ICMP redirect* per una rete a cui ha già accesso diretto, pertanto si avrà solo una metà della connessione.

Un attacco dagli effetti analoghi è quello denominato *DHCP spoofing* in cui `ettercap` si finge un server DHCP e cerca di sostituirsi a quello reale nel fornire una risposta alla vittima, in cui indica se stesso come gateway. In questo modo di nuovo la vittima invierà verso l'attaccante i pacchetti destinati alle connessioni la rete esterna, che provvederà a reinviarli al vero gateway dopo averli intercettati. Anche in questo caso si potrà osservare solo una metà del traffico (quello in uscita).

1.2.3 I *security scanner*

Un *security scanner* è un programma in grado analizzare una rete alla ricerca di servizi vulnerabili. In genere esso è pure in grado di compiere direttamente una serie di attacchi per verificare l'effettiva vulnerabilità dei servizi rilevati, con maggiore o minore profondità e dettaglio a seconda della implementazione.

Di nuovo si tratta di uno strumento che può essere usato sia da un attaccante che da chi deve provvedere alla sicurezza del sistema. In questo caso è di fondamentale importanza per tenere sotto controllo la propria rete, e come mezzo preventivo per rilevare potenziali problemi prima che lo faccia qualcuno con intenzioni peggiori delle vostre.

Il problema principale che si può avere con un *security scanner* è dovuto al fatto che quelli più evoluti eseguono la ricerca ed il rilevamento delle vulnerabilità attaccando effettivamente i servizi disponibili,[22] per cui il loro uso può causare malfunzionamenti dei servizi, la loro interruzione, ed addirittura, nel caso di sistemi operativi poco stabili, al crash della macchina.

Per questo motivo l'uso di uno *scanner* su macchine in produzione può essere estremamente pericoloso, e si deve sempre stare attenti a quello che si fa per non mettere a rischio i propri servizi. Inoltre si deve essere consapevoli che l'uso di uno *scanner* è del tutto equivalente ad un tentativo di intrusione, ed è necessario essere certi di avere anche il diritto legale di poter compiere una attività potenzialmente pericolosa per il sistema informatico che si vuole esaminare.

Un altro problema comune degli *scanner* è quello dell'affidabilità delle loro rilevazioni. Un primo problema è che è possibile che eventuali vulnerabilità presenti non vengano rilevate (i cosiddetti *falsi negativi*). Questo avviene perché una vulnerabilità, pur essendo stata scoperta, non è detto sia fra quelle verificate dal programma che si sta usando, per questo conviene sempre utilizzare uno *scanner* che fornisca un meccanismo di aggiornamento e comunque interessarsi a come detto aggiornamento viene effettuato.

Il secondo problema è quello della rilevazione di vulnerabilità inesistenti (i cosiddetti *falsi positivi*), in genere per l'utilizzo di test difettosi, o basati solo sul controllo della versione dei servizi. Questo è un caso molto comune e può facilmente causare dei falsi positivi in quanto è del tutto normale, come fa tra l'altro Debian per le sue distribuzioni stabili, correggere i problemi di sicurezza con degli opportuni patch, senza aggiornare tutto il programma, che così viene riportato, non essendo cambiata la versione, come vulnerabile.

[22] alcuni programmi o test si limitano ad una banale verifica della versione del servizio usato, da confrontare contro un elenco di versioni vulnerabili.

In entrambi i casi è comunque regola da seguire quella di prendere le segnalazioni di uno *scanner* per quello che sono: un'utile traccia per tenere sotto controllo la presenza di servizi inutili o obsoleti, da rimuovere o aggiornare, e non la certificazione della sicurezza o meno di una rete. Anche un risultato negativo non deve comunque lasciare tranquilli, gli *scanner* possono solo controllare le vulnerabilità loro note, e non può quindi sostituire una politica di controllo degli accessi basata su firewall o sugli altri meccanismi di protezione interni ai servizi.

Fino a qualche anno fa nel caso di GNU/Linux parlare di *scanner* di sicurezza significava sostanzialmente parlare di *Nessus*. Dalla versione 3 però il programma è diventato proprietario, ed è stato sostituito da un fork portato avanti da un progetto denominato *OpenVAS* (*Open Vulnerability Assessment System*), che ha ripreso il codice della ultima versione libera e lo ha sviluppato. Come per la versione da cui origina, anche *OpenVAS* è un programma di analisi modulare basato su plug-in ed una architettura client/server. La potenza del programma consiste esattamente nella combinazione in questi due aspetti, che lo rendono un sistema di analisi potente e flessibile.

Avendo a disposizione una architettura in cui il server è la parte che si occupa di eseguire i test di sicurezza, si possono dislocare diversi server su diverse macchine, in modo da avere diversi punti di vista sulla stessa rete. L'architettura a plug-in consente inoltre una grande flessibilità nella gestione dei test da eseguire, ciascuno di essi infatti è scritto in forma di plug-in, sia in C, che in un linguaggio di scripting dedicato, il NASL (*Nessus Attack Scripting Language*), progettato per scrivere con facilità nuovi test di sicurezza. É pertanto anche molto semplice avere estensioni e nuovi test, ed esiste un database on-line continuamente aggiornato dagli sviluppatori.

La struttura del programma si è molto evoluta, e nelle versioni più recenti le funzionalità sono state suddivise in diversi demoni separati e specializzati che comunicano via rete e che possono essere interrogati dai relativi client per raccogliere informazioni ed elaborare rapporti, mentre in precedenza prevedevano semplicemente un programma client che pilotava direttamente il server che eseguiva le scansioni.

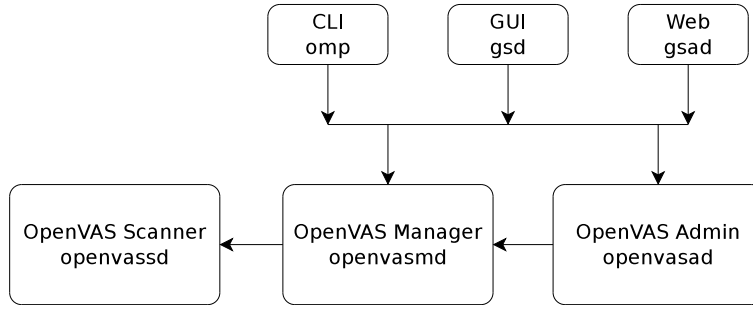


Figura 1.16: Architettura delle varie componenti di OpenVAS.

L'architettura di OpenVAS nelle versioni più recenti è schematicamente illustrata in fig. 1.16; il cuore del sistema è `openvassd`, l'*OpenVAS Scanner*, un demone che attinge ad un elenco di *Network Vulnerability Test* (NVT, nella nomenclatura del programma) ed esegue direttamente le scansioni che gli vengono richieste. I client però non interagiscono direttamente con lo scanner, ma si appoggiano ad al demone di gestione `openvasmd`, l'*OpenVAS Manager*, che cura l'interazione fra i client e lo *scanner* (per gestire richieste e risposte) e si appoggia al demone

di amministrazione `openvasad`, l'*OpenVAS Administrator*, che gestisce gli accessi e gli utenti di OpenVAS.

A queste tre componenti server si aggiungono i vari client che consentono agli utenti di richiedere le scansioni ed ottenere i relativi risultati, il programma ne prevede ben tre, con una programma a riga di comando, uno con interfaccia grafica, ed una applicazione web.

Fra le distribuzioni solo Debian forniva un pacchetto ufficiale che però è piuttosto arretrata rispetto allo sviluppo corrente[23] il progetto comunque distribuisce direttamente dal suo sito web (`http://www.openvas.org`) versioni in pacchetti delle varie versioni già pronte per le principali distribuzioni, con tanto di istruzioni, per cui non staremo a trattare i dettagli dell'installazione.

Una volta che si siano installate tutte le componenti il sistema prevede che tutte le comunicazioni siano cifrate con SSL ed è pertanto necessario generare sia i certificati che la relativa *Certification Authority*; questo può essere fatto direttamente in fase di installazione se previsto nei pacchetti, o manualmente in un secondo tempo, i vari file vengono installati sotto `/var/lib/openvas/CA/` per i certificati e sotto `/var/lib/openvas/private/CA/` per le chiavi.

Per generare manualmente i certificati usati dallo *scanner* e la relativa *Certification Authority* si può usare il comando `openvas-mkcert` che richiede l'inserimento delle solite informazioni illustrate in tab. **??**. Se i file esistono già il comando ne stampa il pathname rifiutandosi di proseguire fintanto che non si specifica l'opzione `-f` che forza la sovrascrittura; il comando può eseguire una generazione veloce con l'opzione `-q` in cui vengono inseriti valori predefiniti per le informazioni di tab. **??**, senza stare a richiederle interattivamente, si avrà pertanto un risultato del tipo:

```
# openvas-mkcert -q
/var/lib/openvas/private/CA created
/var/lib/openvas/CA created
```

Come illustrato in fig. 1.16 lo scanner viene realizzato dal demone `openvassd` che viene lanciato dallo script `/etc/init.d/openvas-scanner` (si tenga presente che almeno su Debian l'installazione non imposta l'avvio automatico di questi servizi, che devono essere attivati esplicitamente). Se eseguito a riga di comando il programma parte in modalità demone; per lanciarlo in maniera interattiva e farlo restare collegato sul terminale occorre specificare l'opzione `-f`. Di default il servizio si pone in ascolto solo su localhost, e sarà quindi utilizzabile solo localmente, se lo si vuole in ascolto su un indirizzo generico occorrerà utilizzare l'opzione `-a`.[24]

Le altre opzioni principali di `openvassd` sono riportate in tab. 1.8, l'elenco completo è nella pagina di manuale (accessibile con `man openvassd`). Nella stessa pagina di manuale sono indicate anche le direttive che possono essere usate nel file di configurazione del programma, `/etc/openvas/openvassd.conf`, che consentono di impostare le posizione dei vari file e directory, l'utente a cui inviare la posta, ed una serie di altre caratteristiche di funzionamento. Il file ha la forma di una direttiva assegnata al relativo valore; se il file non è presente il programma funziona ugualmente, assumendo che si vogliano usare i valori di default, che possono .

Il file di configurazione del server è `/etc/openvas/openvassd.conf` e contiene varie direttive di configurazione, fra cui le locazioni dei vari file e directory, l'utente a cui inviare la posta, e

[23] al momento della stesura di queste note la versione corrente è la 6.0, mentre su Debian Squeeze è disponibile il pacchetto della 2.0 e sulle versioni successive il pacchetto del programma non viene più fornito.

[24] su Debian sia l'indirizzo che la porta su cui `openvassd` si pone in ascolto possono essere impostati in `/etc/default/openvas-scanner`.

Opzione	Descrizione
`-c file`	usa il file di configurazione specificato al posto di `/etc/openvas/openvassd.conf`.
`-a addr`	specifica l'indirizzo IP su ascolta il demone.
`-p port`	specifica la porta su cui si pone in ascolto il server (il default è la 1241).
`-S ip1,...`	specifica la lista degli indirizzi IP da cui si accetteranno le connessioni.
`-f`	mantiene il server attivo sul terminale.
`-s`	stampa le opzioni di configurazione.

Tabella 1.8: Principali opzioni di `openvassd`.

l'impostazione delle caratteristiche di funzionamento, Debian installa una versione ben commentata, con dei valori di default ragionevoli, per la descrizione completa della sintassi si può fare riferimento alla pagina di manuale di *OpenVAS* accessibile con `man openvassd`.

Una volta lanciato lo *scanner* il secondo passo per utilizzare *OpenVAS* è attivare il *manager* con il demone `openvasmd`. Anche in questo caso la cosa deve essere fatta esplicitamente tramite lo script di avvio `/etc/init.d/openvas-manager`, ma perché questo sia possibile occorrono alcune operazioni preliminari, occorre infatti, perché questo possa comunicare con lo *scanner*, predisporre un opportuno certificato client. Per far questo si può usare il comando `openvas-mkcert-client`.

Il comando consente di creare un certificato client per la connessione allo scanner, per questo deve essere lanciato con l'opzione `-n` seguita dal nome dell'utente, che verrà registrato anche come utente per lo *scanner*, l'opzione `-i` installa il certificato per l'uso da parte del *manager*. Seguendo quanto indicato nelle istruzioni di installazione ed utilizzando `om` come nome utente, sarà cioè necessario eseguire il comando:

```
# openvas-mkcert-client -n om -i
Generating RSA private key, 1024 bit long modulus
..++++++
.....................................................................++++++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:State or Province Name (full name) [Some-State
]:Locality Name (eg, city) []:Organization Name (eg, company) [Internet Widgits
Pty Ltd]:Organizational Unit Name (eg, section) []:Common Name (eg, your name or
 your server's hostname) []:Email Address []:Using configuration from /tmp/openv
as-mkcert-client.13157/stdC.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
localityName          :PRINTABLE:'Berlin'
commonName            :PRINTABLE:'om'
Certificate is to be certified until Dec 26 16:38:54 2012 GMT (365 days)
```

```
Write out database with 1 new entries
Data Base Updated
User om added to OpenVAS.
```

Una volta predisposto il certificato si potrà avviare il *manager*, il comando prevede come opzioni principali `-p`, per impostare la porta su cui ascolta per le connessioni, e `-s` per impostare la porta su cui contattare lo *scanner*. Il comando inoltre prevede le due opzioni speciali `--rebuild` e `--update` per rispettivamente ricostruire e aggiornare il database delle informazioni ottenute dallo *scanner*. Le altre opzioni principali sono riportate in tab. 1.9.

Opzione	Descrizione
`-p port`	indica la porta su cui il servizio è in ascolto.
`-s port`	indica la porta su cui contattare lo *scanner*.
`-a addr`	specifica l'indirizzo IP su cui il servizio è in ascolto.
`-l addr`	specifica l'indirizzo IP su cui contattare lo *scanner*.
`--update`	aggiorna il database delle informazioni dallo *scanner*.
`--rebuild`	ricostruisce il database delle informazioni dallo *scanner*.

Tabella 1.9: Principali opzioni di `openvasmd`.

Come accennato una delle caratteristiche più interessanti di *OpenVAS* è quella di eseguire le scansioni tramite l'uso dei *Network Vulnerability Test*, che sono distribuiti in maniera indipendente dal programma. Per la gestione degli NVT viene fornito un apposito programma, `openvas-nvt-sync`; se si è in rete il programma si incarica di scaricare automaticamente le ultime versioni di tutti i test di sicurezza dal sito del progetto ed installarle nella relativa directory (il default è `/var/lib/openvas/plugins`). Una volta scaricati la prima volta il grosso degli NVT per ottenere gli eventuali aggiornamenti basterà rieseguire il comando. Un esempio di uso del comando è il seguente:

```
# openvas-nvt-sync
[i] This script synchronizes an NVT collection with the 'OpenVAS NVT Feed'.
[i] The 'OpenVAS NVT Feed' is provided by 'The OpenVAS Project'.
[i] Online information about this feed: 'http://www.openvas.org/openvas-nvt-feed.html'.
[i] NVT dir: /var/lib/openvas/plugins
[i] rsync is not recommended for the initial sync. Falling back on http.
[i] Will use wget
...
zyxel_pwd.nasl
zyxel_pwd.nasl.asc
[i] Download complete
[i] Checking dir: ok
[i] Checking MD5 checksum: ok
```

Una volta scaricati i test occorrerà reinizializzare il sistema, per questo sarà necessario fermare i servizi e riavviarli, in particolare sarà necessario riavviare manualmente lo *scanner* per fargli eseguire il caricamento iniziale degli NVT, che è una operazione piuttosto lunga in quanto questi sono alcune decine di migliaia. Pertanto il semplice riavvio del servizio risulterà piuttosto lento, lanciando invece `openvassd` su un terminale si otterrà qualcosa del tipo:

```
# openvassd
Loading the plugins... 10302 (out of 23988)
```

```
...
All plugins loaded
```

vedendo un progressivo crescere del numero fino al completamento del caricamento. Il passo successivo è riavviare anche il *manager* riaggiornando per la presenza dei nuovi test, per questo le istruzioni di installazione consigliano l'esecuzione preventiva dei comandi:

```
# openvasmd --migrate
# openvasmd --rebuild
```

che consentono di ricostruire (o creare quando eseguiti la prima volta) il database delle informazioni, senza il quale il programma non parte.

Una volta avviati lo *scanner* ed il *manager* il terzo servizio necessario per usare *OpenVAS* è l'*administrator*, realizzato dal programma `openvasad`.

Si tenga presente inoltre che i precenti passi riguardano solo l'autenticazione e cifratura del dialogo fra le sue componenti; oltre a questo *OpenVAS* supporta un suo database indipendente degli utenti, in modo da poter gestire chi si può collegare ai servizi e quali test può eseguire. Pertanto per poterlo utilizzare si deve prima creare almeno un utente con il comando `openvas-adduser`; questo è di uso immediato e si limita richiedere i dati dello stesso sul terminale con qualcosa del tipo:

```
# openvas-adduser
Using /var/tmp as a temporary file holder.

Add a new openvasd user
--------------------------------


Login : piccardi
Authentication (pass/cert) [pass] :
Login password :
Login password (again) :

User rules
---------------
openvasd has a rules system which allows you to restrict the hosts that piccardi has the right to test.
For instance, you may want him to be able to scan his own host only.

Please see the openvas-adduser(8) man page for the rules syntax.

Enter the rules for this user, and hit ctrl-D once you are done:
(the user can have an empty rules set)


Login             : piccardi
Password          : ***********

Rules             :


Is that ok? (y/n) [y] y
user added.
```

Il programma chiede un username e un metodo di autenticazione che per default (selezionato premendo invio) è tramite password, che dopo deve essere immessa per due volte (la seconda

per conferma) la password. Di seguito possono essere specificate delle regole di accesso. Le regole prendono le parole chiavi `accept` e `deny` (il cui significato è evidente) seguite da una rete specificata in notazione CIDR. È inoltre possibile utilizzare la parola chiave `default` per indicare una politica di accesso di default; un esempio, preso dalla pagina di manuale, è il seguente:

```
accept 192.168.1.0/24
accept 192.168.3.0/24
accept 172.22.0.0/16
default deny
```

queste devono essere immesse come testo una per riga e l'immissione deve essere conclusa con l'invio di un carattere di *end-of-file* (vale a dire `C-d`).

Completata l'immissione delle regole (che in genere si può lasciare vuota premendo subito `ctrl-D`) viene stampato un riassunto delle impostazioni di cui il programma chiede conferma. Sempre sulla pagina di manuale si trovano tutte le istruzioni dettagliate relative al funzionamento del comando. È altresì disponibile il comando `openvas-rmuser` che consente di rimuovere un utente con qualcosa del tipo:

```
# openvas-rmuser
Login to remove : piccardi
user removed.
```

Una volta eseguite le predecenti operazioni di preparazione (impostare un utente, aggiornare i plug-in, avviare il server) si potrà lanciare il client con il comando `openvas`. Questo è un programma ad interfaccia grafica, che partirà con la finestra principale posizionata sulla sezione di login, secondo quanto illustrato in fig. 1.17.

Come si può notare gran parte della finestra del programma è occupata dal selettore delle funzionalità, che si scelgono con le linguette in alto; la parte più bassa della finestra contiene tre bottoni che consentono rispettivamente di avviare la scansione, di caricare i risultati di una scansione precedente o di uscire dal programma. La sezione centrale contiene la parte di finestra relativa alle varie funzionalità, che nel caso del login è divisa in due parti: in quella superiore sono disposti i campi per scegliere il server a cui collegarsi, specificandone indirizzo e porta, mentre nella parte inferiore ci sono i campi per l'username e la password, da fornire per poter utilizzare il server.

Una volta immessi username e password e premuto il pulsante per il login, quando ci si collega la prima volta si presenterà una finestra di accettazione del certificato SSL, passata la quale il programma si sposterà da solo nella sezione di selezione dei plugin, illustrata in fig. 1.18, avvisando al contempo, con una ulteriore finestra in pop-up, che i plug-in potenzialmente pericolosi sono stati disabilitati.

Nella sezione di selezione dei plugin la parte alta della finestra mostra le classi di attacchi possibili, elencate una per riga, con una checkbox che mostra se la classe è abilitata o meno. Selezionando una riga con il mouse vengono mostrati tutti i plugin contenuti nella relativa classe nella parte bassa della finestra, uno per riga, contenente, oltre la solita checkbox, pure una eventuale icona che identifica il plugin come *pericoloso*. Selezionando uno di questi col mouse viene visualizzata una finestra con una breve descrizione della vulnerabilità e del comportamento del plugin.

Nella parte centrale della finestra una serie di bottoni consentono di impostare i plugin da utilizzare, consentendo ad esempio di abilitare solo quelli non classificati *pericolosi*. Questi

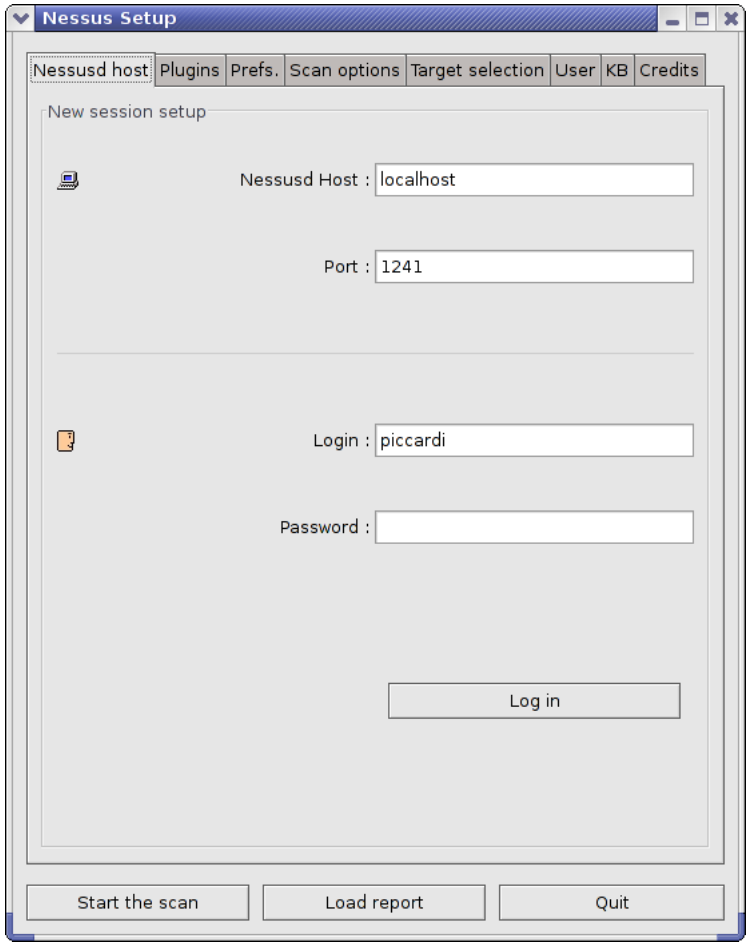


Figura 1.17: Finestra di login di `openvas`.

ultimi sono quelli che, qualora attivati, tentano una verifica diretta delle vulnerabilità e possono pertanto causare il crash dei rispettivi servizi (e con Windows in vari casi anche del sistema operativo). Si tenga presente comunque che la classificazione è basata sulla esperienza dell'autore del plugin, e non c'è nessuna assicurazione sul fatto che un plugin non classificato pericoloso non lo sia davvero. Pertanto si eviti di usare `openvas` nei confronti di macchine in produzione, a meno di non essere disposti a correre il rischio di renderle non più tali.

Una volta scelti i plugin che si vogliono utilizzare, prima di poter iniziare una scansione, occorre selezionare il bersaglio della stessa. Questo si fa dalla sezione Target selection della finestra principale, come illustrato in fig. 1.19. Il bersaglio può essere immesso nel relativo campo (o letto anche da un file). Basta inserire un indirizzo di una macchina o quello di una rete in notazione CIDR. Se si usa un indirizzo simbolico relativo ad un dominio la relativa checkbox permette di ottenere dal DNS un elenco delle macchine presenti nello stesso attraverso

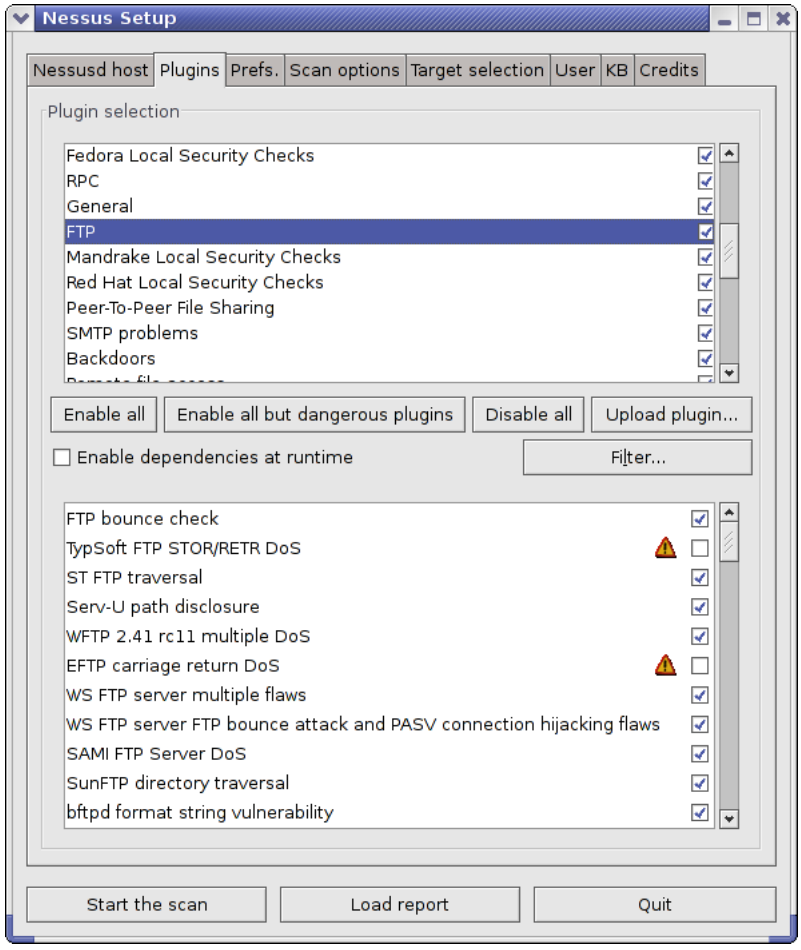


Figura 1.18: Finestra di selezione dei plugin di `openvas`.

un trasferimento di zona. La parte bassa della finestra contiene i risultati delle precedenti sessioni di scansione, che vengono salvate se si attiva la relativa checkbox.

Le restanti sezioni permettono di configurare altre funzionalità; le principali sono quella per le opzioni relative alle singole scansioni (Scan option), in cui si può stabilire l'intervallo di porte da esaminare, le modalità di uso del portscanner ed altre opzioni relative alla esecuzione della scansione, e quella delle preferenze (Prefs.) che permette di specificare tutta una lunga serie di impostazioni relative alle modalità con cui `openvas` esegue i vari programmi a cui si appoggia.[25]

Completate le impostazioni si potrà avviare la scansione, ed il server inizierà le sue operazioni (si tenga presente che la cosa può richiedere molto tempo) durante le quali il client mostrerà

[25] ad esempio Openvas utilizza `nmap` per eseguire un portscan preliminare per determinare su quali porte sono presenti servizi, e può invocare ulteriori programmi esterni come Hydra per cercare di eseguire un attacco a forza bruta per il login su servizi che richiedono autenticazione.

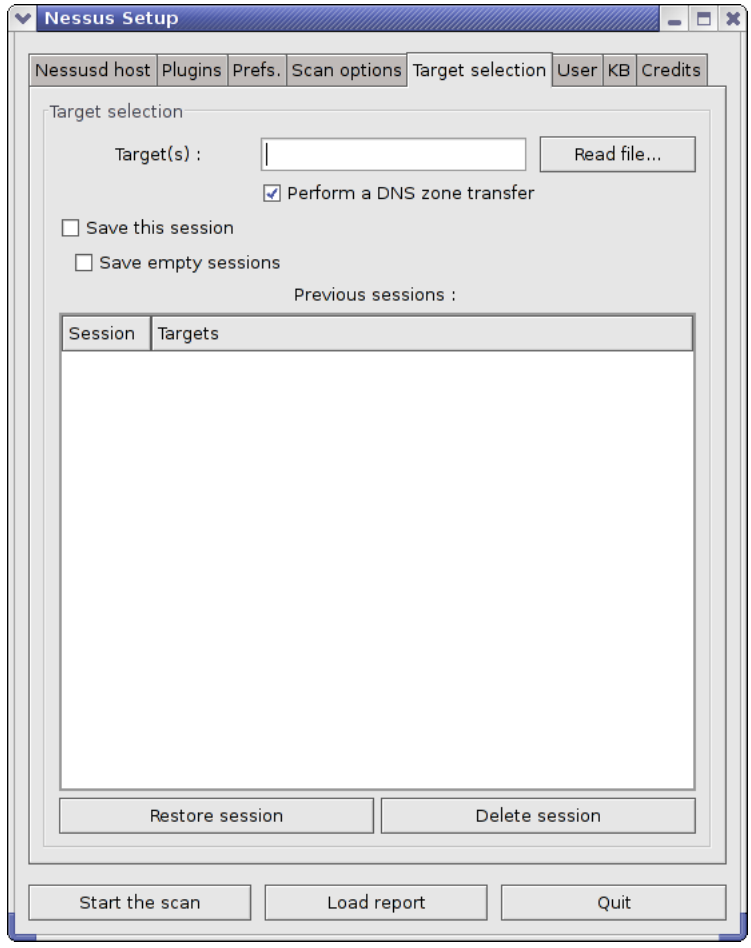


Figura 1.19: Finestra dei bersagli della scansione di `openvas`.

una finestra di progresso del tipo di quella riportata in fig. 1.20. Nel nostro caso si è eseguita una scansione su una sola macchina, che è l'unica riportata, ma `openvas` consente di analizzare anche intere reti, nel qual caso la finestra riporterà lo stato di tutte le scansioni in corso. La finestra consente, coi relativi pulsanti, di interrompere le singole scansioni o una intera sessione.

Una volta completata la scansione `openvas` genererà un rapporto dei risultati ottenuti in una finestra apposita, come quella riportata in fig. 1.21. Questa riassume sulla sinistra in alto la lista reti analizzate, selezionando una rete comparirà in basso la lista delle macchine ivi presenti per le quali si è eseguita la scansione. Una volta selezionata una macchina compariranno nella parte alta a destra due ulteriori sotto-finestre, la prima contenente la lista dei servizi che sono stati rilevati, affiancati da una icona che identifica la gravità di eventuali problemi (assente se non ve ne sono) affiancata a destra da un elenco dei problemi rilevati. Selezionando uno dei problemi per i quali esiste un rapporto quest'ultimo verrà mostrato nella parte inferiore della finestra.

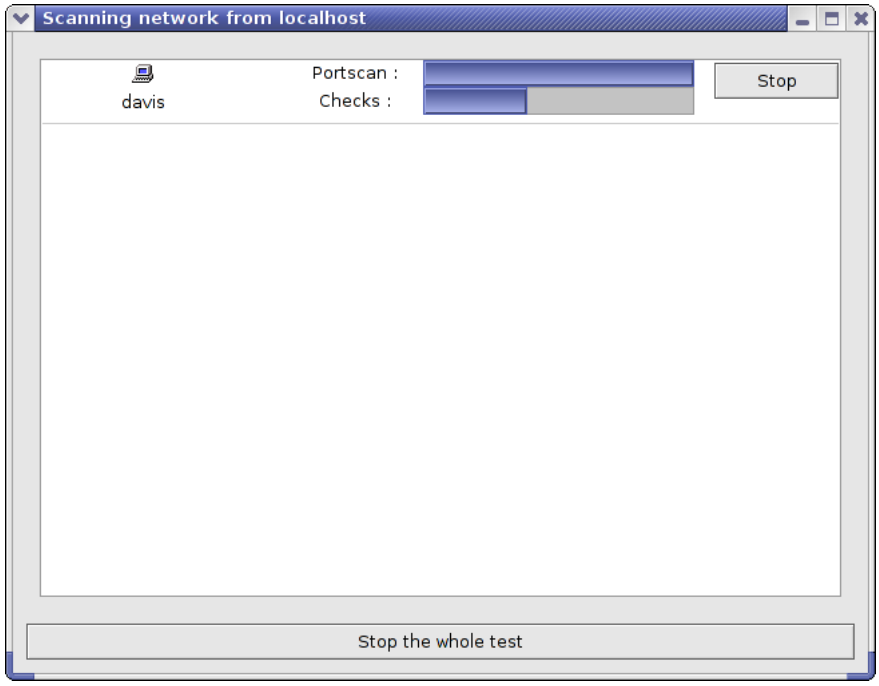


Figura 1.20: Finestra del progresso delle operazioni di `openvas`.

Un apposito pulsante consente di salvare l'intero rapporto su disco in diversi formati, compreso quello di una directory contenente grafici e dati in formato HTML pronti per essere pubblicati su web.

1.2.4 I monitor di rete

Benché non si tratti strettamente di strumenti di ricerca di vulnerabilità, vale la pena parlare di una serie di programmi che servono per fare delle sommarie analisi del traffico, che possono essere di aiuto ad identificare traffico sospetto. La differenza principale fra questi programmi e un NIDS è che essu è che questi in genere servono per scopi diversi (più che altro monitoraggio) dalla rilevazione di intrusioni, anche se poi tornano utili per le informazioni che sono in grado di fornire.

Il primo di questi programmi è `iptraf`, una specie di incrocio fra uno *sniffer* vero e un programma per l'analisi del traffico, `iptraf` utilizza una interfaccia utente semplice, che lavora comunque in modalità testuale, basata sulle librerie *ncurses*.

L'uso del programma è immediato in quanto basato su menù e maschere, sia pure testuali; una volta lanciato ci presenterà la schermata di avvio, e basterà premere un tasto per ottenere il menù principale delle opzioni, riportato in fig. 1.22.

Questo presenta molteplici opzioni, e per entrare nel dettaglio di tutte le possibilità offerte dal programma conviene leggere la documentazione allegata, che nel caso di Debian viene installata

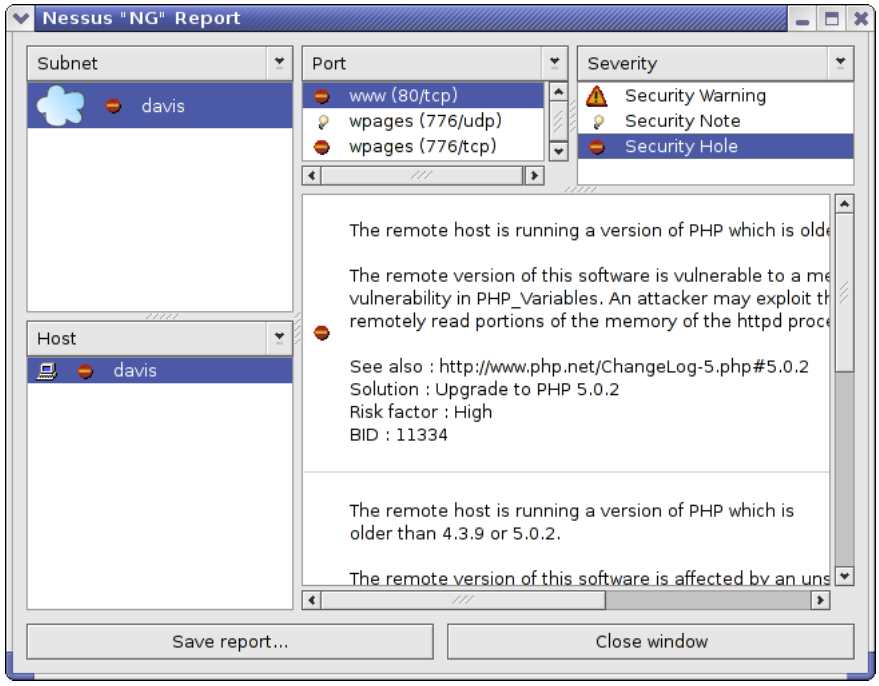


Figura 1.21: Finestra dei rapporti dei risultati di `openvas`.

in `/usr/share/doc/iptraf`; in particolare può essere di aiuto il manuale d'uso presente nella sottodirectory `html`, che illustra le molteplici funzionalità del programma.

Le funzionalità di analisi `iptraf` sono elencate nella parte superiore del menù di fig. 1.22, e possono essere selezionate spostandosi con i tasti di freccia e premendo invio una volta posizionati su quella prescelta. Si possono inoltre impostare le configurazioni tramite l'apposita voce del menù, così come si possono creare dei filtri (utilizzando una serie di maschere accessibili una volta selezionato la relativa voce). Non entreremo nei dettagli di queste configurazioni ulteriori lasciando l'esplorazione delle stesse come esercizio.

La prima voce del menù, `IP traffic monitor`, consente una analisi dettagliata del traffico IP. Una volta selezionata verrà mostrato un ulteriore menù di scelta (mostrato in fig. 1.23) che permette di selezionare l'interfaccia di rete su cui porsi in ascolto (o di ascoltare su tutte quante); si potrà selezionare una delle voci del menù spostandosi con i tasti di freccia e poi attivarla premendo invio. Selezionata l'interfaccia verrà attivata la finestra di osservazione del traffico, riportata in fig. 1.24, in cui vengono mostrati i risultati della cattura dei pacchetti in tempo reale.

Come si può notare in fig. 1.24 la finestra di osservazione è suddivisa in due parti; la prima e più ampia, posta in alto, contiene il traffico TCP, classificato automaticamente per connessioni. Ciascuna connessione è indicizzata in prima colonna tramite la coppia `indirizzo:porta` dei due estremi della connessione che sono riportate su due righe adiacenti collegate. Le colonne successive indicano il numero di pacchetti emessi nelle due direzioni, i flag attivi (presi dall'ultimo

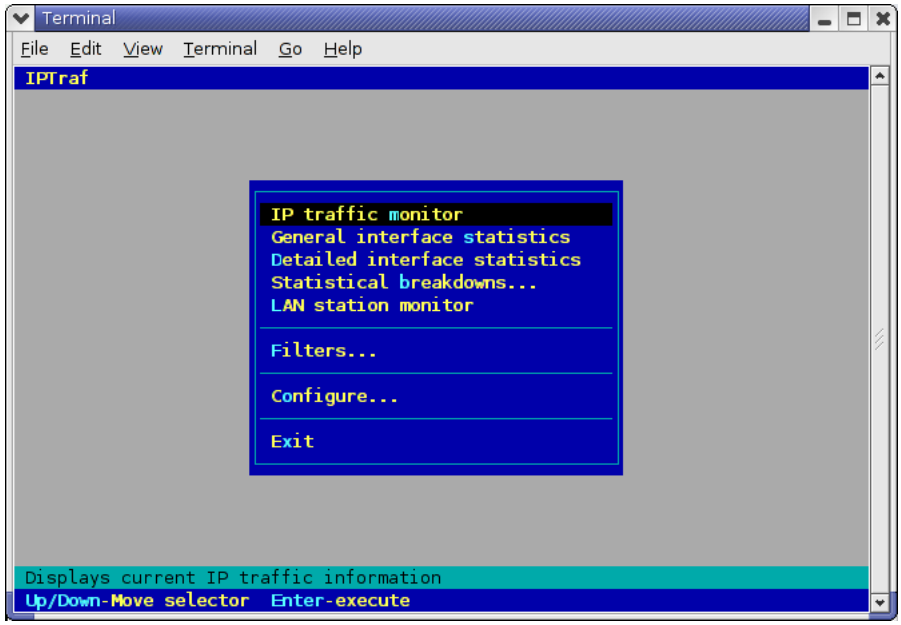


Figura 1.22: Menù principale di `iptraf`.

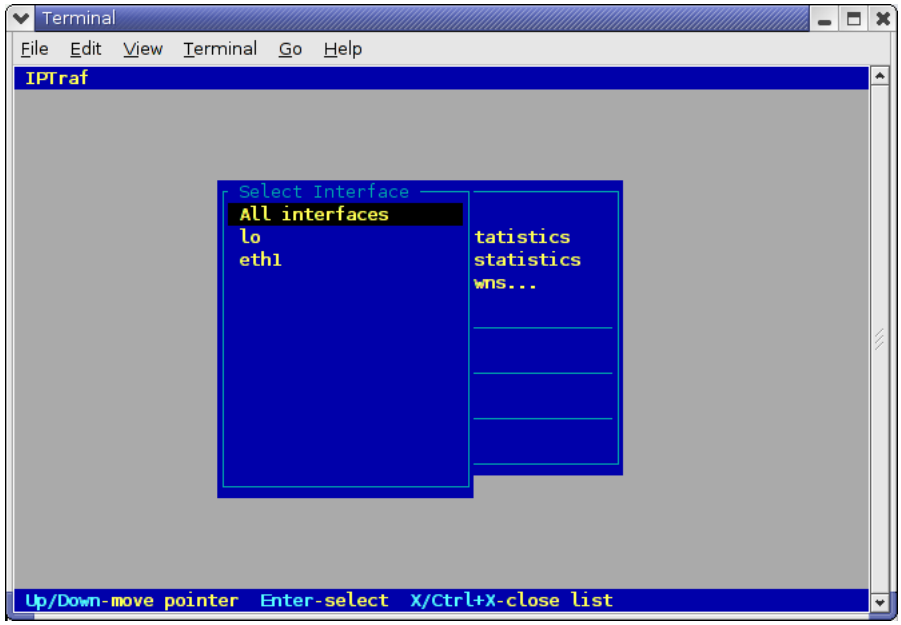


Figura 1.23: Menù di selezione delle interfacce di `iptraf`.

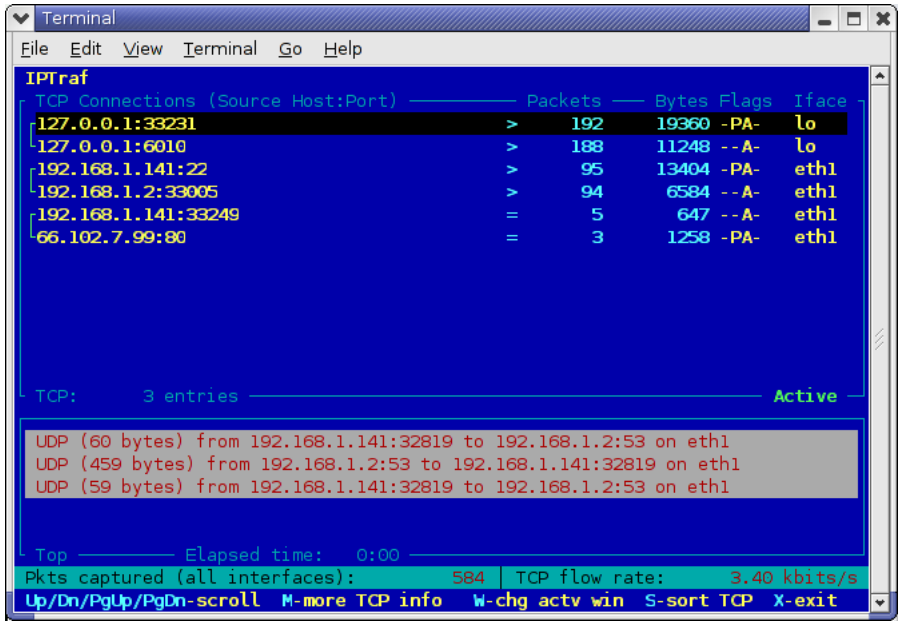


Figura 1.24: Finestra del monitor del traffico di `iptraf`.

pacchetto rilevato ed aggiornati in tempo reale) ed infine l'interfaccia di provenienza degli stessi.

La seconda sezione della finestra, di dimensioni più ridotte e posta in basso contiene invece i dati relativi al traffico dovuto a pacchetti non TCP, riportando in prima colonna il nome del relativo protocollo. Nel caso illustrato in fig. 1.24 è facile identificare il traffico come relativo a delle richieste UDP effettuate sul DNS.

In questa seconda finestra vengono riassunte brevemente le informazioni essenziali trovate nelle intestazioni dei relativi protocolli. Nel caso di UDP viene mostrata prima la dimensione di ciascun pacchetto ed a seguire indirizzi e porta sorgente e destinazione, per finire con l'interfaccia. Nel caso di ICMP viene mostrato il tipo di pacchetto, seguito dalla dimensione e dagli indirizzi sorgente e destinazione; al solito chiude l'interfaccia su cui è stato catturato il pacchetto.

Benché non si possa apprezzare dalle figure, entrambe le finestre di osservazione sono *navigabili*. Si può passare dall'una all'altra con il tasto di tabulazione, ed inoltre le finestre scorrono quando riempite, e una volta posizionatisi su di esse ci si può muovere al loro interno con i soliti tasti di freccia per rivedere il traffico precedente, ed effettuare uno scroll con i tasti di pagina su e pagina giù. Chiude la finestra di osservazione una riga riassuntiva sul numero di pacchetti catturati ed il flusso di dati su TCP ed una di aiuto che riassume i tasti utilizzabili.

Oltre alla osservazione dettagliata del flusso dei pacchetti, `iptraf` consente anche la raccolta di statistiche relative all'uso della rete. Dalla seconda voce del menù principale si ottiene una finestra di statistiche generali (mostrata in fig. 1.25), che raccoglie i dati relativi a tutte le interfacce presenti sul sistema, e che riporta un sommario delle informazioni essenziali relative a ciascuna interfaccia.

In questo caso viene riservata una riga per ogni interfaccia presente, il cui nome viene indicato

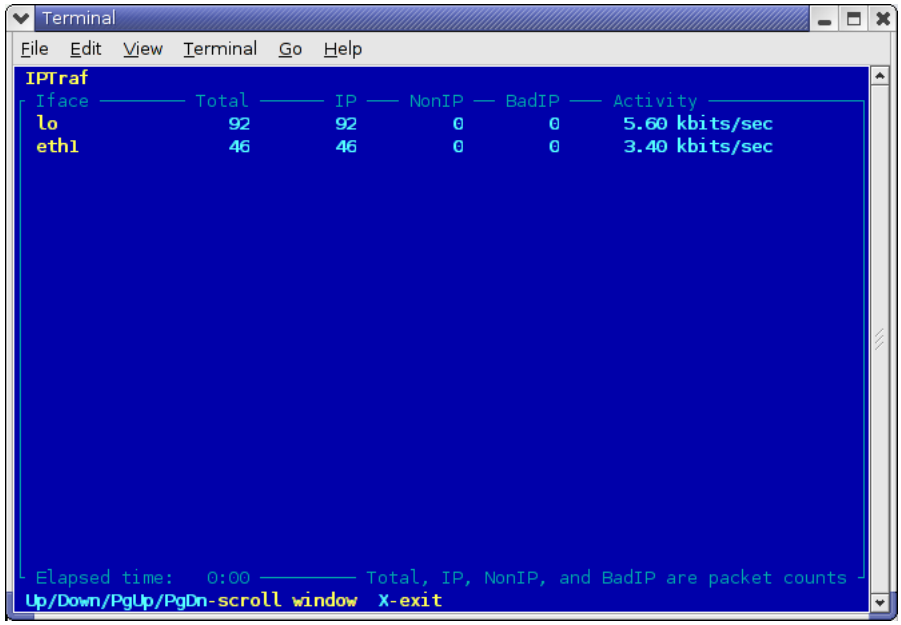


Figura 1.25: Finestra delle statistiche generali di `iptraf`.

nella prima colonna. Seguono una serie di colonne che riportano il totale dei pacchetti catturati ed una loro suddivisione elementare in pacchetti IP, non IP, pacchetti IP rovinati, più una stima del traffico sostenuto nella colonna finale, che comparirà solo quando è stata raccolta una statistica sufficiente. Come per la finestra di osservazione i dati vengono aggiornati in tempo reale.

A partire dalla terza voce del menù si può ottenere una finestra con delle statistiche più dettagliate per ciascuna interfaccia di rete. In questo caso prima di arrivare alla finestra illustrata in fig. 1.26 si dovrà nuovamente scegliere l'interfaccia da un menù, analogo a quello di fig. 1.23, dopo di che saranno mostrate le relative statistiche che mostrano non solo il totale dei pacchetti ma anche la loro suddivisione per protocolli, l'ammontare del traffico in byte, e l'andamento del traffico in ingresso ed in uscita.

Come visto l'uso di un programma come `iptraf` consente sia di elencare le connessioni presenti, che di ottenere delle statistiche generali sul traffico, cosa che può essere di aiuto nella rilevazione di traffico anomalo, oltre che per la semplice osservazione dello stato della propria rete.

Un secondo programma che consente di monitorare il traffico di rete è `iftop`, il cui scopo è analogo a quello di `iptraf`. Il programma in questo caso più semplice e non presenta una interfaccia semigrafica tentando di porsi come applicazione del concetto del comando `top` alle connessioni di rete. Il suo principale vantaggio rispetto a `iptraf` è di essere in grado di segnalare, oltre alle connessioni esistenti, la quantità di traffico su ciascuna di esse in tempo reale ed in forma semigrafica, rendendo molto evidente eventuali anomalie.

Il comando non richiede nessun argomento o opzione obbligatoria, e se lanciato senza spe-

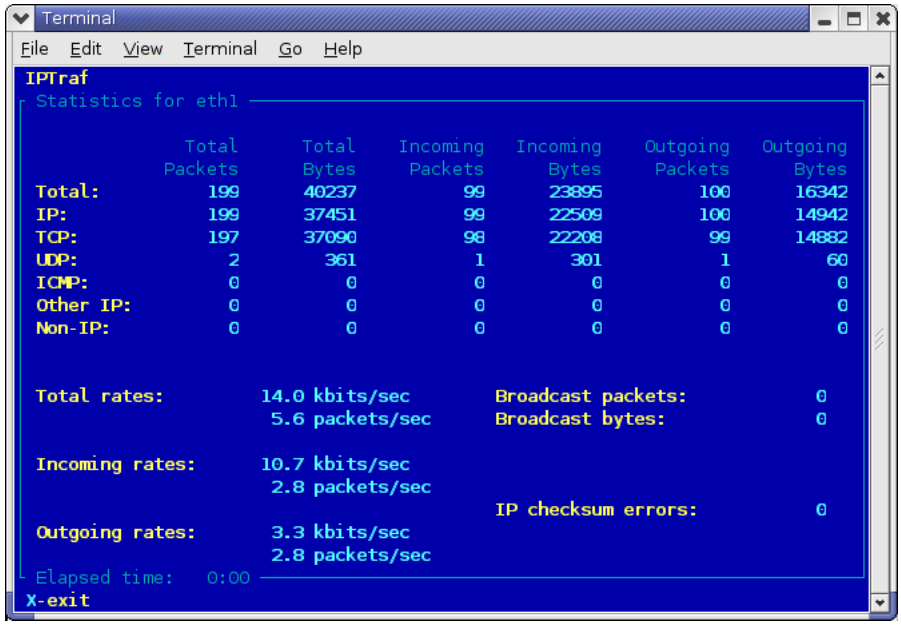


Figura 1.26: Finestra delle statistiche dettagliate per interfaccia di `iptraf`.

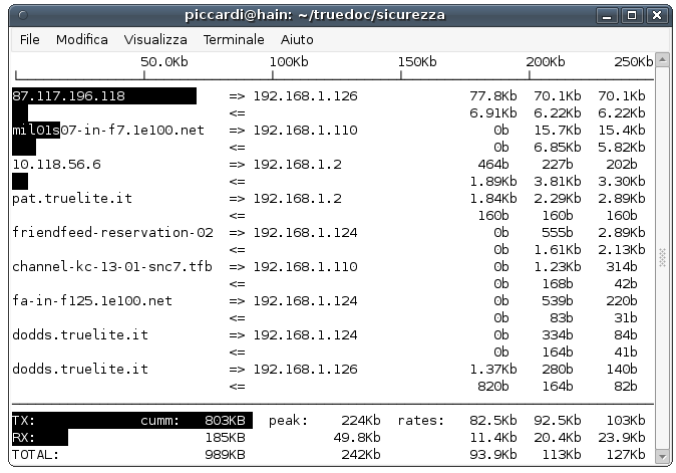


Figura 1.27: Finestra di `iftop`.

cificare niente si pone in ascolto sulla prima interfaccia ethernet attiva, stampando brevemente nome dell'interfaccia e relativo MAC address prima di occupare l'intera area del terminale e mostrare ogni due secondi le connessioni presenti, ordinate per quantità di traffico, come illustrato in fig. 1.27.

La schermata del comando riporta in alto la scala usata per illustrare la quantità di traffico

illustrata in forma di istogramma nelle righe seguenti; ciascuna connessione occupa due righe (una per direzione del traffico) con l'indicazione delle macchine fra cui avviene il traffico ed un riassunto del rate rilevato (in bits/secondo) negli ultimi 2, 10 e 40 secondi. Lo stesso valore viene visualizzato in forma grafica invertendo il colore della riga corrispondente in modo da ottenere un istogramma del traffico in forma orizzontale; per facilitare la visualizzazione viene usata una rappresentazione logaritmica.

Di fronte alle possibilità di intercettazione del traffico che un programma come `ettercap` fornisce, l'unica contromisura possibile è l'uso di `arpwatch`, un programma nato per tenere sotto controllo una rete ethernet e segnalare la presenza di nuovi indirizzi hardware e tutti gli eventuali cambiamenti nelle corrispondenze fra MAC address e numeri IP. Il programma si mette in ascolto sulla rete e segnala via e-mail tutti i cambiamenti rilevati; in questo modo dopo un periodo iniziale di stabilizzazione in cui vengono identificate tutte le macchine presenti si otterrà un avviso tutte le volte che un nuovo MAC address farà la sua comparsa sulla rete.

Un programma di questo tipo è di grande aiuto per rilevare il traffico sospetto generato localmente da utenti che cercano di evitare eventuali controlli eseguiti a livello di IP andando ad operare direttamente su ethernet, per rilevare i tentativi di *ARP poisoning* e per verificare l'ingresso di nuove macchine in una rete.

Il funzionamento normale di `arpwatch` prevede che il programma venga eseguito in background come demone analizzando tutto il traffico ARP sull'interfaccia che gli viene specificata a riga di comando con l'opzione `-i`, registrando le corrispondenze IP/MAC su un file che di default è `arp.dat`. Tutte le anomalie rilevate saranno inviate all'indirizzo specificato con l'opzione `-m` tramite il comando specificato con l'opzione `-s`. Le altre principali opzioni sono riportate in tab. 1.10, l'elenco completo è nella pagina di manuale.

Opzione	Descrizione
`-d`	abilita il debug ed esegue il programma interattivamente.
`-f`	specifica un file da cui leggere le corrispondenze fra indirizzi IP e MAC, se diverso da `arp.dat`.
`-i`	specifica l'interfaccia su cui osservare il traffico (il default è la prima).
`-m`	specifica l'indirizzo di e-mail cui inviare gli avvisi.
`-p`	disabilita il *modo promiscuo* per l'interfaccia.
`-r`	legge i dati da un file (passato come parametro) invece che dall'interfaccia.
`-s`	specifica il programma da usare per inviare le e-mail di avviso.

Tabella 1.10: Le principali opzioni di `arpwatch`.

Il programma manda i suoi errori sul syslog, e necessita di un file `arp.dat` (la directory usata di default è `/var/lib/arpwatch`) da cui leggere le coppie di corrispondenza fra indirizzi IP/MAC rilevate in precedenza; se il file non esiste deve essere creato vuoto. Inoltre nel caso di Debian viene fornito uno script di avvio che legge il contenuto del file `/etc/arpwatch.conf` e si mette in ascolto sulle interfacce ivi specificate, inviando gli avvisi agli indirizzi ivi specificati; il file ha un formato molto semplice, un estratto è:

arpwatch.conf

```
eth0    -N -p -m root+eth0@example.com
eth1    -N -p -m root+eth1@example.com
```

con il semplice formato di:

```
<interfaccia> <opzioni ...>
```

Qualora vengano rilevate delle anomalie `arpwatch` genera dei messaggi di allarme, che poi possono essere inviati via e-mail ad un amministratore, o registrati sul *syslog* (in genere entrambi). Un elenco di questi messaggi, insieme al relativo significato, è riportato in tab. 1.11.

Messaggio	Descrizione
`new activity`	rileva un nuovo abbinamento fra IP e MAC address.
`new station`	rileva un MAC address mai visto prima.
`flip flop`	l'indirizzo è passato da quello visto per ultimo a quello visto come penultimo.
`changed ethernet address`	L'indirizzo IP è passato ad un nuovo MAC address.
`ethernet broadcast`	si è osservato un indirizzo di broadcast in un MAC address o come sorgente in un pacchetto ARP.
`ip broadcast`	si è osservato un indirizzo di broadcast sull'IP di una macchina.
`bogon`	si è rilevato un indirizzo IP non corrispondente alla sottorete su cui si affaccia l'interfaccia.
`ethernet mismatch`	il MAC address non corrisponde a quello indicato come sorgente nel pacchetto ARP.
`reused old ethernet address`	analogo a `flip flop` ma il passaggio è riferito ad un indirizzo precedente il penultimo.

Tabella 1.11: I messaggi di allarme di `arpwatch`.

Un possibile esempio dell'utilizzo di `arpwatch` è il seguente. Appena installato ed avviato il programma per la prima volta otterremo una serie di e-mail del tipo:

arpwatch-new-station-mail

```
X-Original-To: root+eth0@monk.truelite.it
From: Arpwatch <arpwatch@monk.truelite.it>
To: root+eth0@monk.truelite.it
Subject: new station (parker.truelite.it)

            hostname: parker.truelite.it
          ip address: 192.168.1.1
    ethernet address: 0:48:54:62:18:6f
     ethernet vendor: Digital Semi Conductor 21143/2 based 10/100
           timestamp: Monday, June 7, 2004 17:47:49 +0200
```

per tutte le macchine presenti sulla rete. Se poi però proviamo ad utilizzare `ettercap` per avvelenare la cache di `zorn.truelite.it` otterremo anche:

arpwatch-changed-mail

```
X-Original-To: root+eth0@monk.truelite.it
From: Arpwatch <arpwatch@monk.truelite.it>
To: root+eth0@monk.truelite.it
Subject: changed ethernet address (parker.truelite.it)

            hostname: parker.truelite.it
          ip address: 192.168.1.1
    ethernet address: 0:10:dc:c0:6c:b2
```

```
     ethernet vendor: Micro-Star International Co., Ltd.
old ethernet address: 0:48:54:62:18:6f
 old ethernet vendor: Digital Semi Conductor 21143/2 based 10/100
           timestamp: Monday, June 7, 2004 17:53:39 +0200
  previous timestamp: Monday, June 7, 2004 17:52:49 +0200
               delta: 50 seconds
```

il che ci mostra chiaramente come ci sia stato un tentativo di modificare la scheda di rete a cui vengono inviati i pacchetti destinati all'indirizzo `192.168.1.1`.

1.3 I *sistemi antintrusione locali*

Come accennato in sez. 1.1 esistono varie tipologie di IDS, in questa sezione ci occuperemo di quelli che si possono chiamare *sistemi antintrusione locali*, cioè di quei programmi che permettono di rilevare in tempo reale o verificare a posteriori eventuali tentativi di intrusione su una singola macchina.

1.3.1 Programmi di verifica di sicurezza.

Un programma di verifica di sicurezza è tutto sommato molto simile a quello che potrebbe essere un *antivirus* su un sistema non unix-like; infatti benché con GNU/Linux per la struttura del sistema i virus siano sostanzialmente inefficaci[26] l'installazione di un *rootkit* o di un *trojan* da parte di un eventuale intruso che ha sfruttato una vulnerabilità del sistema ha sostanzialmente lo stesso effetto,[27] con la modifica di alcuni programmi di sistema e l'installazione di altri che consentano un accesso da remoto.

Il principio di un programma di scansione di sicurezza è allora quello di eseguire una serie di controlli per rilevare la presenza di eventuali *rootkit* o *trojan* sulla base del comportamento degli stessi, (la cosiddetta *signature*) che si suppone nota in precedenza. In sostanza, come per un antivirus, si ha a disposizione un database dei *rootkit* noti, e si analizza il sistema alla ricerca delle modifiche che questi possono avere effettuato agli altri programmi, all'inserimento di file e directory per eventuali *backdoor* e alla presenza di file di log (in cui in genere vengono memorizzate le informazioni raccolte *sniffando* sia la rete che l'attività sul sistema).

Il problema che si pone nell'uso di questo tipo di programmi è che essi sono in grado di rilevare solo la presenza di *rootkit*, *trojan* o *backdoor* che gli siano noti, qualora venga usato un *rootkit* sconosciuto al programma o modificato in maniera da non essere riconosciuto, la loro efficacia cesserà immediatamente. Per questo motivo sono stati sviluppati sistemi alternativi in grado di superare questo problema.

Nel caso di GNU/Linux uno dei programmi più utilizzati per questo compito è `chkrootkit`, nato, come il nome stesso suggerisce, per verificare la presenza di eventuali *rootkit* installati su una macchina. In realtà `chkrootkit` non si limita al controllo dei soli *rootkit* e effettua anche

[26] nell'uso da parte di un utente normale un virus non ha nessuna possibilità di modificare i programmi di sistema, danneggiare quelli di altri utenti, o effettuare una qualunque operazione privilegiata, per cui risulta di scarsa efficacia.

[27] in realtà questo vale solo per i cosiddetti *worm*, *rootkit* e *trojan* hanno il solo scopo di consentire all'attaccante un successivo accesso alla macchina, anche se questa ha corretto la vulnerabilità, e non hanno normalmente nessun effetto *infettivo* nei confronti di altre macchine.

una serie di ulteriori verifiche per identificare eventuali *trojan* o *backdoor* o altri programmi indesiderati.

L'uso del programma è tutto sommato elementare, basta eseguirlo (ovviamente con i privilegi di amministratore) da un terminale e questo stamperà a video l'elenco dei suoi risultati, con un qualcosa del tipo:

```
# chkrootkit
ROOTDIR is '/'
Checking 'amd'... not found
Checking 'basename'... not infected
Checking 'biff'... not infected
...
Checking 'write'... not infected
Checking 'aliens'... no suspect files
Searching for sniffer's logs, it may take a while... nothing found
Searching for HiDrootkit's default dir... nothing found
Searching for t0rn's default files and dirs... nothing found
Searching for t0rn's v8 defaults... nothing found
Searching for Lion Worm default files and dirs... nothing found
Searching for RSHA's default files and dir... nothing found
Searching for RH-Sharpe's default files... nothing found
...
Searching for ShKit rootkit default files and dirs... nothing found
Searching for anomalies in shell history files... nothing found
Checking 'asp'... not infected
Checking 'bindshell'... not infected
Checking 'lkm'... nothing detected
Checking 'rexedcs'... not found
Checking 'sniffer'...
lo: not promisc and no packet sniffer sockets
Checking 'w55808'... not infected
Checking 'wted'... nothing deleted
Checking 'scalper'... not infected
Checking 'slapper'... not infected
Checking 'z2'... nothing deleted
```

ovviamente in questo caso il problema resta quello che se è stato installato un *rootkit* non noto, esso non sarà rilevato.

1.3.2 Programmi per la verifica di integrità

Come accennato in precedenza per `chkrootkit`, un qualunque sistema di rilevazione di *rootkit* basato su controlli effettuati in base alla conoscenza di cosa viene modificato o inserito nel sistema, è destinato a fallire non appena l'attaccante utilizzi una metodologia diversa.

Esistono però delle caratteristiche comuni di qualunque *rootkit*, ed è pertanto possibile superare il precedente problema sfruttando questo fatto. Ad esempio un *rootkit*, nel momento in cui cerca di nascondere le tracce del suo operato, dovrà necessariamente modificare il comportamento di programmi come `ps`, `netstat`, `lsof`, ecc. Per questo motivo una strategia efficace è semplicemente quella di mettere sotto osservazione tutti i file critici di un sistema, per poterne verificare l'integrità in qualunque momento successivo.[28]

[28] esiste una eccezione a tutto ciò, che consiste nel realizzare il *rootkit* direttamente nel kernel, in modo da far

Il problema che si pone nell'attuare questa strategia è che una eventuale compromissione del sistema può comportare anche quella del sistema di verifica, pertanto il meccanismo per essere veramente efficace prevede una modalità di operazione che comporti la registrazione dello stato originario del sistema (e dello stesso programma di verifica) in un supporto che non possa essere compromesso (cioè su un mezzo che sia fisicamente accessibile in sola lettura) dal quale poi effettuare il controllo.

L'*Advanced Intrusion Detection Environment*, in breve `aide`, è un programma che permette di costruire un database di una serie di proprietà, come permessi, numero di inode, dimensioni, tempi di accesso ed una serie di hash crittografici, relativo all'insieme dei file che si vogliono tenere sotto controllo, per poter verificare in un momento successivo tutti gli eventuali cambiamenti delle stesse.

Tipicamente l'uso del programma comporta che l'amministratore esegua la creazione del database (da eseguire con il comando `aide --init`, anche se Debian mette a disposizione uno script apposito) una volta completata l'installazione della macchina, prima che questa venga posta in rete e sia esposta a possibili intrusioni. Si dovrà poi provvedere a salvare il database, il programma di controllo ed il relativo file di configurazione su un mezzo appropriato che sia impossibile da modificare da parte un eventuale attaccante; in genere si tratta di un floppy o di un CDROM o di una chiave USB: comunque di un mezzo per il quale sia possibile impostare l'accesso in sola lettura a livello fisico.

Nel caso di Debian l'installazione è elementare, essendo disponibile direttamente il pacchetto `aide` che installa tutto il necessario, compresi gli script eseguiti quotidianamente da *cron* per controllare lo stato dei file. La prima schermata di installazione, mostrata in fig. 1.28, notifica la scelta di default per l'utente a cui inviare la posta dei rapporti periodici, suggerendo al contempo come modificare in un momento successivo l'impostazione nel file di configurazione di `aide` che in Debian è tenuto sotto `/etc/aide/aide.conf`.

Una volta letta la schermata e premuto invio, la configurazione ne propone una seconda, riportata in fig. 1.29, che chiede se creare il database. Nel caso specifico abbiamo scelto di farlo subito, in caso contrario sarebbe stata mostrata un'altra schermata per ricordare che per creare il database in un secondo momento è disponibile un apposito script, `aideinit`.

Una volta creato il database (di default Debian lo crea in `/var/lib/aide/aide.db`) è necessario salvare su un supporto apposito non solo il quest'ultimo ma anche file di configurazione e programma, in quanto un eventuale compromissione permetterebbe all'attaccante di modificare la versione presente sul sistema. Per questo il controllo deve essere sempre fatto con l'immagine del programma salvata sul mezzo in sola lettura, per essere sicuri di usare un programma non compromesso. Nel caso di Debian `aide` viene installato linkato in maniera statica, in modo da impedire all'attaccante di comprometterne il funzionamento sostituendo delle librerie.

Una volta creato il database si potrà controllare l'integrità dei file messi sotto controllo con il comando `aide --check`. In generale però, dato che l'operazione non comporta sostanzialmente un carico aggiuntivo significativo, è preferibile utilizzare il comando nella forma `aide --update`, che oltre al rapporto sulle differenze trovate, permette di creare una nuova versione del database, salvato nel nuovo file `/var/lib/aide/aide.db.new`. Questo è anche quello che viene fatto dal

sparire alla radice tutte le tracce; un esempio di questo è `adore`, realizzato come modulo del kernel che effettua la cancellazione di tutta una serie di tracce; in tal caso si dovrà utilizzare un equivalente sistema di controllo realizzato allo stesso livello.

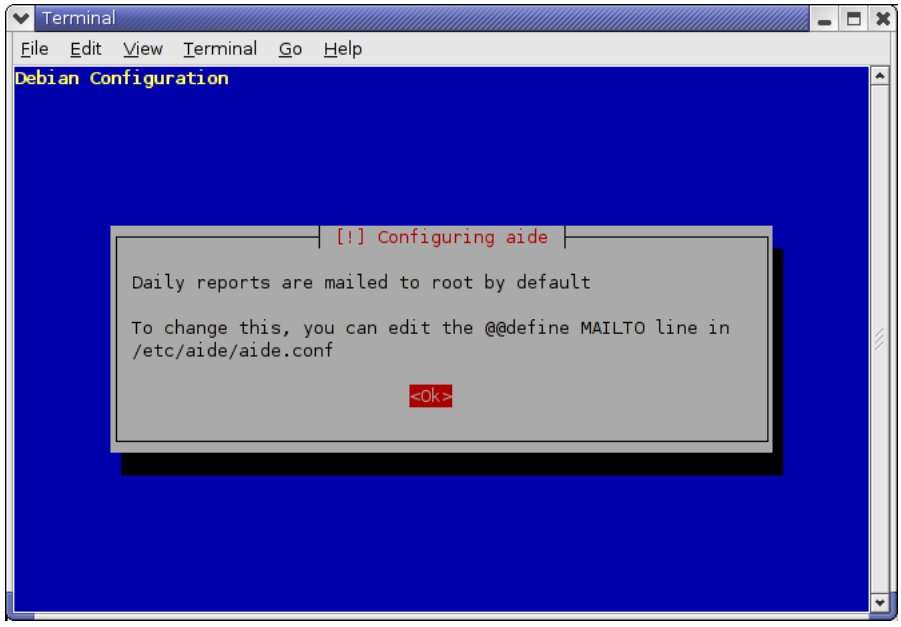


Figura 1.28: Prima finestra di configurazione dell'installazione di `aide`.

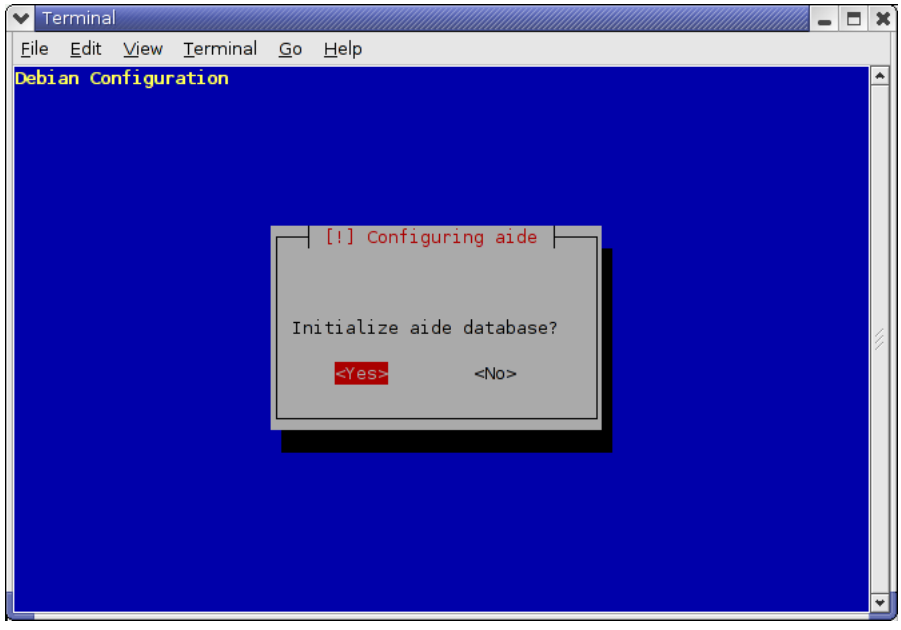


Figura 1.29: Seconda finestra di configurazione nell'installazione di `aide`.

cron job installato dal pacchetto che gira tutte le notti, e invia una e-mail con il rapporto dei cambiamenti all'amministratore (a meno di non aver modificato l'impostazione di default).

In generale avere dei cambiamenti non è un sinonimo di intrusione, il caso più comune è che questi sono del tutto legittimi: ad esempio possono essere state installate delle nuove versioni di alcuni programmi, o modificate alcune delle configurazioni. Pertanto il rapporto deve essere verificato costantemente, e qualora le modifiche trovate siano legittime, si potrà evitare di vedercele riproposte tutte le volte salvando la nuova versione del database al posto della precedente. Quando si è sicuri del nuovo database si può provvedere a ripetere l'operazione utilizzando la versione sul supporto in sola lettura, provvedendo a salvare anche il nuovo database che può prendere a tutti gli effetti il posto del precedente.

Occorre ovviamente effettuare il controllo in condizioni di sicurezza, ad esempio disconnettendo la macchina durante il controllo con il database precedente (da fare sempre dal supporto in sola lettura) e la produzione ed il salvataggio del nuovo file, per evitare eventuali modifiche *al volo*[29] da parte di un eventuale intruso. Una buona politica può essere quella di eseguire l'aggiornamento in maniera periodica, verificando cosa è cambiato dalla volta precedente, in modo da avere sempre un file di controllo sufficientemente recente.

La parte più critica nell'utilizzo di `aide` (come di ogni programma simile) è proprio la scelta di quali file tenere sotto controllo. Questo è controllato dal contenuto del file di configurazione del programma, `aide.conf`. Il file prevede tre tipi diversi di direttive; le prime sono le righe di configurazione, nella forma `parametro=valore`, esempi possibili sono le linee:

```
database=file:/var/lib/aide/aide.db
```

che specifica dove è mantenuto il database di `aide`, o:

```
database_out=file:/var/lib/aide/aide.db.new
```

che specifica il file in cui viene creato il nuovo database quando si esegue `aide --update`. L'elenco completo dei parametri predefiniti e dei possibili valori è documentato, insieme a tutte le altre caratteristiche del file, nella relativa pagina di manuale, accessibile con `man aide.conf`.

Se il parametro non corrisponde a nessuno dei valori predefiniti l'espressione viene allora considerata come la definizione di un nuovo *gruppo*; i gruppi sono la modalità con cui `aide` identifica le proprietà di un file da tenere sotto controllo, alcuni di questi *gruppi* sono predefiniti ed identificati dalle espressioni riportate in tab. 1.12, un nuovo gruppo può essere specificato con una espressione del tipo:

```
# Custom rules
Binlib = p+i+n+u+g+s+b+m+c+md5+sha1
ConfFiles = p+i+n+u+g+s+b+m+c+md5+sha1
```

in cui si sommano gruppi predefiniti o altri gruppi definiti in precedenti (ma è supportata anche l'operazione di sottrazione per gruppi già definiti).

Il secondo tipo di righe presenti nel file di configurazione sono le righe di selezione dei file da tenere sotto controllo; queste sono di tre tipi, quelle normali che iniziano sempre per "`/`", quelle di uguaglianza che iniziano con "`=`" che servono ad aggiungere file al database e quelle di negazione che iniziano con "`!`" che permettono di deselezionare file selezionati dalle precedenti.

[29] poco probabili, ma nelle questioni di sicurezza la paranoia è una virtù.

Gruppo	Descrizione
p	permessi del file
i	numero di inode
n	numero di link
u	utente proprietario
g	gruppo proprietario
s	dimensione
b	numero di blocchi
m	tempo di ultima modifica
a	tempo di ultimo accesso
c	tempo di ultimo cambiamento dell'inode
S	dimensione crescente
md5	md5 checksum
sha1	sha1 checksum
rmd160	rmd160 checksum
tiger	tiger checksum
R	p + i + n + u + g + s + m + c + md5
L	p + i + n + u + g
E	gruppo vuoto
>	file di log (p + u + g + i + n + S)
crc32	crc32 checksum
haval	haval checksum
gost	gost checksum

Tabella 1.12: Gruppi predefiniti per le proprietà dei file in `aide.conf`.

Ciascuna di queste è una espressione regolare da usare come corrispondenza con il pathname assoluto di un file, per cui qualcosa del tipo `=/etc` corrisponde con qualunque file dentro `/etc`.

Il terzo ed ultimo tipo di direttive sono le righe che esprimono dei *macro*-comandi, (è previsto un linguaggio elementare che permette di definite variabili e eseguire blocchi condizionali) inizianti sempre per `@@`, queste sono quelle che definiscono le variabili come `MAILTO` che sono usate dallo script del *cron job*. Di nuovo per una lista completa e relativa spiegazioni si può fare riferimento alla pagine di manuale.

Nel caso di Debian viene usato un file con valori di default ragionevoli ed una serie di regole predefinite, ma in ogni caso la scelta dei file da tenere sotto controllo dipende sempre dal compito delle varie macchine, e un po' di lavoro di personalizzazione è sempre necessario. In questo il ciclo di revisione del database dopo i controlli, con l'adattamento della configurazione per inserire nel controllo solo i file (o le proprietà) la cui variazione è significativa, è di fondamentale importanza.

1.4 I *NIDS* per GNU/Linux

In questa sezione prenderemo in considerazione le problematiche relative all'uso dei *Network Intrusion Detection System* disponibili per GNU/Linux, cioè a quei sistemi antintrusione specificamente destinati al controllo del traffico di rete, per riconoscervi segnali o tracce di tentativi di intrusione o di intrusioni vere e proprie.

1.4.1 La dislocazione di un NIDS

Prima di analizzare il funzionamento di un NIDS specifico come `snort` occorre fare alcune considerazioni di carattere generale. Come accennato in sez. 1.2.2 a proposito degli *sniffer* la presenza di reti switchate comporta un problema qualora si voglia analizzare il traffico di una rete, in quanto solo il traffico a lei diretto potrà arrivare alla macchina sulla quale è posto il sistema antintrusione.

Questo pone allora una serie di problemi, il primo, di difficile soluzione, è quello dell'osservazione di tutto il traffico. Questo può essere risolto solo per quegli switch che sono sufficientemente sofisticati da essere in grado di deviare il traffico che passa attraverso di loro, usando il cosiddetto *mirroring*, su una certa porta.

In questo caso basta piazzare la macchina che fa da NIDS sulla porta di *mirror* però si pone il problema che su una porta si hanno al massimo 100Mbit, mentre il traffico sullo switch può essere molto superiore, nel qual caso si perdono pacchetti. Inoltre per alcuni switch l'abilitazione della modalità di *mirroring* degrada notevolmente le prestazioni.

Negli switch più semplici questa funzionalità non esiste, per cui l'unica alternativa è usare un hub, su cui attaccare le macchine del tratto di rete di cui si vuole osservare il traffico (il router e il firewall) e quella con il sistema di *intrusion detection*. Questo non consente di osservare tutto il traffico, ma in genere questo non è necessario, l'inconveniente è che per flussi di traffico molto elevati si possono avere collisioni saturando l'*hub*, inoltre con l'inserimento nella rete dell'*hub* si ha un ulteriore *single point of failure*.[30]

Questo ci lascia con la necessità di scegliere in quale punto della propria rete inserire il sistema di NIDS. I due punti critici sono fra il router ed il firewall, in diretto contatto con il traffico esterno, e sulla rete interna. Nel primo caso si possono osservare tutti gli attacchi diretti verso la propria rete, nel secondo solo quelli che passano il filtro del firewall.

In generale porre il sistema di rilevazione davanti al firewall ha il vantaggio di mostrare lo stato effettivo degli attacchi che si possono subire, ma quando il firewall fa anche da NAT si perde la capacità di tracciare la sorgente di eventuali pacchetti sospetti che provengono dall'interno. Il fatto che in questo caso si generino grosse quantità di allarmi, rischia di far abituare chi li controlla ad un esame sommario, rendendo più difficile l'individuazione di quelli che hanno successo.

Porre il sistema dietro il firewall ha il vantaggio di rilevare solo gli attacchi che passano quest'ultimo e generano pertanto meno carico di lavoro non tanto sul sistema (nel caso di `snort` questo è trascurabile) quanto su chi deve eseguire i controlli. Inoltre in questo modo si possono individuare in maniera diretta le macchine che inviano traffico sospetto, ed accorgersi di attacchi provenienti dall'interno, spesso i più pericolosi. Però in questo modo si perde contatto con quelli che sono i pericoli reali che possono provenire dalla rete e si può ingenerare un falso senso di sicurezza e perdere la consapevolezza della provenienza degli attacchi, in quanto questi vengono bloccati dal firewall.

Su quale sia la scelta migliore la discussione è accesa, esistono buone ragioni per entrambe le scelte, anche se personalmente, dovendone scegliere una sola, propenderei per quella interna. Se le risorse lo consentono, implementarle entrambe può essere la soluzione ideale. La regola d'oro comunque resta di piazzare un sistema di rilevazione laddove si è sicuri che poi si sia in

[30] si chiama così, nella struttura di un sistema generico, un componente che, con il suo malfunzionamento, compromette l'intero funzionamento del sistema.

grado di esaminarne i dati, la risorsa dell'amministratore che verifica gli allarmi infatti è quella più preziosa e difficile da ottenere.

1.4.2 Il programma `snort`

Il principale e più usato pacchetto di *network intrusion detection* per GNU/Linux è `snort`, un programma in grado di analizzare in tempo reale il traffico di rete, selezionare e registrare i pacchetti, eseguire ricerche e corrispondenze sul contenuto, in modo da riconoscere una gran varietà di attacchi e tentativi di scansione (dai *portscan* ai tentativi di riconoscimento del sistema operativo, dai *buffer overflow* agli *shell-code*).

Il programma può operare in tre modalità, come semplice *sniffer*, in modalità analoga a `tcpdump`, come sistema per la registrazione dei pacchetti, e come vero e proprio sistema di rilevamento delle intrusioni. La forza del programma è la presenza di un linguaggio che permette di creare complesse regole di selezione per i pacchetti da analizzare, vari meccanismi di registrazione (dai file di testo, alla registrazione su database) ed un motore di analisi modulare che consente di inserire nuovi meccanismi di rilevamento. Inoltre il programma è dotato di un meccanismo per la gestione degli allarmi, in grado anche esso di inviarli a diversi sistemi di registrazione.

L'architettura modulare consente di estendere con facilità sia le capacità di rilevamento che quelle di registrazione e rendicontazione. I plugin attualmente disponibili consentono la registrazione su database e in formato XML, la rilevazione dei pacchetti frammentati artificialmente, la rilevazione dei *portscan*, la capacità di riassemblare le connessioni TCP, ed un sistema di rilevazione di anomalie statistiche.

Per usare `snort` come sniffer basta lanciarlo con l'opzione `-v`, nel qual caso stamperà a video le intestazioni dei pacchetti in maniera simile a `tcpdump`, con l'opzione `-d` si può richiedere anche la stampa del contenuto dei pacchetti, mentre con `-e` si richiede anche la stampa delle intestazioni dei pacchetti a livello di collegamento fisico.

Il comando prende come argomento una espressione di filtraggio con la stessa sintassi del *Berkley Packet Filter* già illustrata in sez. 1.2.2. Inoltre con l'opzione `-i` si può specificare su quale interfaccia ascoltare, con `-n` specificare un numero massimo di pacchetti da leggere e con `-P` limitare l'acquisizione di un singolo pacchetto ad una certa dimensione. Un esempio di uso di `snort` in questa modalità è il seguente:

```
# snort -dev
Running in packet dump mode
Log directory = /var/log/snort

Initializing Network Interface tun0

        --== Initializing Snort ==--
Initializing Output Plugins!
Decoding 'ANY' on interface tun0

        --== Initialization Complete ==--

-*> Snort! <*-
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
05/19-00:23:26.420194 > l/l len: 0 l/l type: 0x200 B2:C8:0:0:0:0
pkt type:0x4 proto: 0x800 len:0x55
10.1.0.31:38233 -> 192.168.1.2:631 TCP TTL:64 TOS:0x0 ID:49168 IpLen:20 DgmLen:69 DF
```

```
***AP*** Seq: 0x667A1D32  Ack: 0xCA6E3122  Win: 0xF944  TcpLen: 32
TCP Options (3) => NOP NOP TS: 22411008 556682494
50 4F 53 54 20 2F 20 48 54 54 50 2F 31 2E 31 0D  POST / HTTP/1.1.
0A                                               .

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/19-00:23:26.420216 > l/l len: 0 l/l type: 0x200 B2:C8:0:0:0:0
pkt type:0x4 proto: 0x800 len:0x59
10.1.0.31:38233 -> 192.168.1.2:631 TCP TTL:64 TOS:0x0 ID:49169 IpLen:20 DgmLen:73 DF
***AP*** Seq: 0x667A1D43  Ack: 0xCA6E3122  Win: 0xF944  TcpLen: 32
TCP Options (3) => NOP NOP TS: 22411008 556682494
43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20  Content-Length:
32 37 30 0D 0A                                   270..

...
...

===============================================================================
Snort analyzed 84 out of 84 packets, dropping 0(0.000%) packets

Breakdown by protocol:                Action Stats:
    TCP: 84         (100.000%)         ALERTS: 0
    UDP: 0          (0.000%)          LOGGED: 0
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
DISCARD: 0          (0.000%)
===============================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================================
Fragmentation Stats:
Fragmented IP Packets: 0          (0.000%)
    Fragment Trackers: 0
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
  Frag2 memory faults: 0
===============================================================================
TCP Stream Reassembly Stats:
        TCP Packets Used: 0          (0.000%)
         Stream Trackers: 0
          Stream flushes: 0
           Segments used: 0
   Stream4 Memory Faults: 0
===============================================================================
Snort exiting
```

Se si vuole utilizzare `snort` in modalità di *packet logger* per archiviare il traffico occorrerà

specificare una directory dove verranno registrati i pacchetti con l'opzione `-l`. In tal caso tutti i pacchetti (ed in modalità NIDS pure gli alert) verranno decodificati in ASCII come per la stampa a video e salvati in questa directory in una gerarchia di sottodirectory ordinate secondo gli IP di destinazione degli stessi, questo comporta che compariranno anche gli IP delle macchine sulla propria rete locale, per evitarlo, e classificare tutti gli IP in relazione a questa si può usare l'opzione `-h` per indicare qual'è la nostra rete locale, in questo modo i pacchetti IP di risposta diretti alla nostra rete locale saranno comunque classificati sulla base degli IP remoti.

Utilizzare questa modalità comporta delle forti penalità in termini di spazio utilizzato e velocità di memorizzazione, in quanto tutto deve essere convertito in caratteri ASCII, per questo si può usare l'opzione `-b`[31] che registrerà il traffico in un unico file binario (nello stesso formato di `tcpdump`) con un nome del tipo `snort.log` dove `XXX` indica il tempo in cui il programma è stato lanciato; si può specificare un nome diverso con l'opzione `-L`. Si possono anche rileggere i pacchetti da un file invece che dalla rete con l'opzione `-r`.

Modo	Descrizione
`fast`	scrive una riga di allarme sul file di default.
`full`	scrive un allarme completo dell'intestazione del pacchetto che lo ha causato.
`none`	disabilita gli allarmi.
`unsock`	invia tutta l'informazione su un socket.

Tabella 1.13: Le quattro modalità di allarme specificabili con l'opzione `-A` di `snort`.

Infine l'utilizzo principale di `snort` è come NIDS; in tal caso di solito si usa l'opzione `-D` per fare eseguire il programma in background in modalità demone; questa modalità viene attivata con l'opzione `-c` che specifica un file di configurazione da cui leggere le regole di individuazione del traffico sospetto, i plug-in da usare e le modalità di registrazione degli allarmi. Delle sei modalità di allarme usate da `snort` le quattro che coinvolgono la registrazione su un file sono controllate dall'opzione `-A` che specifica uno dei quattro modi riportati in tab. 1.13. In questo caso gli allarmi vengono registrati nella directory specificata con `-l` in una apposito file separato (`alert.log`).

Le altre due modalità prevedono l'invio al sistema del *syslog* con l'opzione `-s` che userà la *facility* `auth` e la priorità `alert`. Alternativamente (avendo una versione compilata con il relativo supporto) si può attivare una finestra di *pop-up* su una macchina Windows usando l'opzione `-M`. Vedremo inoltre che con gli opportuni plug-in sono possibili modalità di registrazione ancora più sofisticate.

Il comando supporta inoltre numerose altre opzioni (ad esempio per controllare l'utente ed il gruppo per conto del quale viene eseguito dopo l'inizializzazione). Si sono riportate le principali in tab. 1.14, per l'elenco completo si faccia al solito riferimento alla pagina di manuale.

1.4.3 La configurazione di `snort` come NIDS

Come accennato in sez. 1.4.2 per utilizzare `snort` in modalità NIDS occorre usare l'opzione `-c` specificando un opportuno file di configurazione. Questa è la modalità in cui viene usualmente

[31] con questa opzione `snort` è tranquillamente in grado di acquisire i dati su una interfaccia a 100Mb a pieno traffico.

Opzione	Descrizione
`-A mode`	specifica la modalità di registrazione degli allarmi.
`-b`	richiede la registrazione dei pacchetti in formato binario compatibile con `tcpdump`.
`-c file`	specifica il file di configurazione che contiene le regole usate in modalità NIDS.
`-d`	acquisisce anche il contenuto dei pacchetti oltre le intestazioni (in modalità sniffer e logging).
`-D`	esegue il comando in background in modalità demone.
`-e`	visualizza/registra anche l'intestazione del pacchetto a livello di collegamento fisico.
`-F file`	legge le regole di selezione dei pacchetti da un file invece che da riga di comando.
`-g group`	gira per conto del gruppo specificato una volta completata l'inizializzazione.
`-h net`	specifica quale è la rete locale su cui si trova la macchina.
`-i`	specifica l'interfaccia da cui acquisire i pacchetti.
`-l dir`	specifica la directory dove registrare i pacchetti e gli allarmi.
`-n`	indica un massimo di pacchetti da acquisire.
`-N`	disabilita la registrazione dei pacchetti (mantenendo quella degli allarmi).
`-P len`	specifica la lunghezza massima dei dati acquisiti all'interno di un pacchetto.
`-r file`	legge i dati da un file invece che dalla rete.
`-s`	manda i messaggi di allarme al *syslog*.
`-S val=x`	imposta una variabile interna al valore specificato.
`-t dir`	esegue un *chroot* alla directory specificata dopo l'inizializzazione (tutti i file saranno relativi a quest'ultima).
`-u user`	gira per conto dell'utente specificato una volta completata l'inizializzazione.
`-v`	stampa le intestazione dei pacchetti sulla console.
`-o`	modifica l'ordine di scansione delle regole.

Tabella 1.14: Principali opzioni di `snort`.

avviato come servizio dagli script di avvio, e nel caso di Debian il file di configurazione usato è `/etc/snort/snort.conf`. Assieme a questo in `/etc/snort/` vengono mantenuti anche una serie di ulteriori file, ed in particolare le regole per la generazione degli allarmi (nella sottodirectory `rules`) ed i file usati dai vari *plug-in* con cui si possono estendere le funzionalità di `snort`.

L'uso di un file di configurazione permette di utilizzare quest'ultimo per specificare una serie di opzioni e valori in maniera più organica rispetto alla riga di comando. Inoltre solo attraverso il file di configurazione si possono impostare le regole di allarme, che costituiscono il cuore delle funzionalità di NIDS di `snort`. Il file ha la solita struttura di una direttiva per riga, con righe vuote o inizianti per "`#`" ignorate, mentre direttive troppo lunghe possono essere scritte su più righe terminate con una "\" che indica la prosecuzione sulla riga successiva.

Ciascuna direttiva è introdotta da una parola chiave, seguita dalla relativa serie di parametri. Le direttive sono di vari tipi e controllano tutti gli aspetti della esecuzione di `snort`, ad esempio la direttiva `config` permette di impostare nel file di configurazione le stesse proprietà che si specificano a riga di comando (ed anche altre che non hanno una corrispondente opzione), con un qualcosa nella forma:

```
# sintassi
# config <opzione>: valore
config set_gid: snort
config disable_decode_alerts
```

le principali opzioni sono riportate in tab. 1.15 l'elenco completo è riportato nello *Snort User Manual* disponibile su `http://www.snort.com/`.

Opzione	Descrizione
`disable_decode_alerts`	disattiva gli allarmi derivati dalla decodifica dei pacchetti, è una delle varie opzioni nella forma `disable_xxx_alerts` che permettono di disabilitare una certa serie di allarmi, i cui esempi si trovano nel file di configurazione di esempio distribuito con `snort` e presente nella maggior parte delle distribuzioni.
`dump_payload`	acquisisce e decodifica il contenuto dei pacchetti, è equivalente all'opzione `-d`.
`decode_data_link`	decodifica il contenuto anche le intestazioni dei pacchetti a livello di collegamento fisico, è equivalente all'opzione `-e`.
`bpf_file`	imposta un file da cui leggere le regole di selezione dei pacchetti, è equivalente all'opzione `-F`.
`daemon`	esegue il programma in modalità demone, è equivalente all'opzione `-D`.
`set_gid`	imposta il gruppo per conto del quale viene eseguito il programma, è equivalente all'opzione `-g`.
`set_uid`	imposta l'utente per conto del quale viene eseguito il programma, è equivalente all'opzione `-u`.
`stateful`	imposta .
`verbose`	stampa l'intestazione dei pacchetti in console, è equivalente all'opzione `-v`.
`pkt_count`	imposta in numero massimo di pacchetti da acquisire, è equivalente all'opzione `-n`.
`nolog`	blocca la registrazione dei pacchetti, è equivalente all'opzione `-N`.
`logdir`	imposta la directory dove registrare i file coi pacchetti acquisiti, è equivalente all'opzione `-v`.
`interface`	imposta l'interfaccia di rete, è equivalente all'opzione `-i`.
`reference_net`	imposta l'indirizzo della rete locale, è equivalente all'opzione `-h`.
`detection`	imposta il motore di rilevamento.
`order`	modica l'ordine di scansione delle regole (vedi sez. 1.4.4), è equivalente all'opzione `-o`.

Tabella 1.15: Principali opzioni della direttiva `config` nel file di configurazione di `snort`.

Una seconda direttiva è `var`, che permette di definire delle variabili interne da riutilizzare all'interno del file di configurazione, utile specialmente per parametrizzare le regole di allarme; un possibile esempio di dichiarazioni di questo tipo (riprese dal file di configurazione installato da Debian) è la seguente:

```
# sintassi
# var VARIABILE valore
#var HOME_NET $eth0_ADDRESS
var HOME_NET [192.168.1.0/24,172.16.0.0/16]
```

```
var EXTERNAL_NET !$HOME_NET
var DNS_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var HTTP_PORTS 80
```

Le variabili possono avere nomi che seguono la stessa sintassi delle variabili di shell, e di norma si segue anche la stessa convenzione di scriverle in lettere maiuscole. I valori possono essere specificati sia facendo riferimento ad altre variabili, anche qui anteponendo un "`$`" come nella shell, ma oltre a stringhe e numeri, si possono specificare indirizzi di rete in notazione CIDR, e liste di valori fra parentesi quadre, inoltre quando si specificano indirizzi e porte si può usare il carattere "`!`" per invertire la selezione.

Come detto più volte la potenza di `snort` consiste anche nella sua capacità di utilizzare una serie di estensioni, queste vengono realizzate nella forma di plugin che permettono di processare i pacchetti una volta che questi sono stati decodificati, ma prima dell'esecuzione del motore di analisi e rilevazione delle intrusioni. Per abilitare queste estensioni si utilizza la direttiva `preprocessor` seguita dal nome del relativo plugin, più tutte le opzioni che ciascuno di questi supporta. Di nuovo un esempio di queste direttive, preso dal file di configurazione di default, è il seguente:

```
# sintassi
# preprocessor <plugin>: opzione
preprocessor flow: stats_interval 0 hash 2
preprocessor frag2
preprocessor stream4: disable_evasion_alerts detect_scans
```

In questo caso ciascun plugin comporta una serie di funzionalità e diverse opzioni ne controllano il comportamento; in particolare molte delle capacità di rilevamento derivano direttamente dagli stessi plugin. Un elenco dei principali è il seguente:

`flow` questo plugin ha sostanzialmente uno scopo di servizio e serve a classificare i flussi di dati, viene poi utilizzato da altri plugin come base per successivi analisi. Al momento viene usato soltanto da `flow-portscan`, ma è previsto che in futuro su di esso si vadano a basare anche gli altri plugin per la classificazione degli stati dei pacchetti. Il plugin supporta quattro opzioni diverse che controllano parametri interni del funzionamento, descritte nel file `README.flow` allegato alla documentazione (su Debian in `/usr/share/doc/snort-doc`). Il plugin non genera nessun tipo di allarme.

`frag2` Questo plugin esegue la deframmentazione preventiva dei pacchetti IP, rilevando al contempo attacchi basati su questo tipo di pacchetti (in genere si tratta di *Denial of Service*). Le opzioni permettono di controllare parametri interni del funzionamento e di solito si lasciano al valore di default. Il loro significato si può trovare nei commenti all'interno del file di configurazione di default distribuito con il pacchetto. Il pacchetto genera degli allarmi se nel riassemblaggio rileva pacchetti di dimensione eccessiva (> 64k), o frammenti troppo piccoli (quelli utilizzati nel cosiddetto *teardrop attack*).

stream4 è il plugin che riassembla i flussi di dati TCP e rileva tutti i pacchetti anomali, identificando vari tipi di portscan, tentativi di *OS fingerprinting*,[32] ed altre anomalie varie legate a possibili attacchi. Supporta numerose opzioni che ne controllano il comportamento, le principali sono `detect_scans` che attiva il rilevamento e la generazione di allarmi in caso di portscan, e `disable_evasion_alerts` che disabilita una possibile sorgente di numerosi allarmi, che di norma vengono attivate. La descrizione delle altre opzioni si può trovare di nuovo nei commenti all'interno del file di configurazione di default distribuito con il pacchetto.

stream4_reassemble questo preprocessore riassembla i flussi di dati delle connessioni TCP, identificando con questo tutti i pacchetti estranei, in questo modo poi si otterrà, per l'uso successivo, un flusso di dati già pulito, del quale esaminare solo il contenuto. In genere viene sempre abilitato insieme al precedente `stream4`.

flow-portscan questo è un motore di rilevazione di portscan che si base sul precedente motore di analisi `flow` che deve sempre essere attivato; questo soprassiede due altri processori usati in precedenza (`portscan1` e `portscan2`) e pur essendo più complesso è in grado di mitigare la frequenza di falsi positivi, e prevede una lunga lista di opzioni di configurazione che permettono di controllarne tutti i parametri interni; questi sono descritti in dettaglio nel file `README.flow-portscan` allegato alla documentazione.

http_inspect questo preprocessore consente di decodificare e normalizzare il traffico HTTP, in modo da rilevare tutti gli attacchi basati sull'uso di diverse mappe di caratteri e codifiche, riportando tutto ad una codifica standard. Viene usato insieme al successivo `http_inspect_server`, entrambi sono descritti in dettaglio nel file `README.http_inspect` della documentazione allegata.

Il comando supporta una serie di altri preprocessori, in genere per effettuare decodifiche di particolari protocolli o analisi specifiche, la relativa documentazione, scarsa purtroppo, si trova insieme alla documentazione generale del pacchetto `snort` o nei commenti del file di configurazione.

Un'altra direttiva fondamentale per l'uso di `snort` è `output`, che governa i molteplici plugin per la registrazione dei dati (sia pacchetti che allarmi) rilevati dal programma. Un esempio di queste direttive, ricavato da quelli contenuti nel file di configurazione di default, è il seguente:

```
# sintassi
# output <plugin>: opzione
output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT
output log_tcpdump: snort.log
output database: log, mysql, user=root password=test dbname=db host=localhost
```

[32] chiama così la tecnica, usata anche da `nmap` come accennato in sez. 1.2.1, per rilevare il tipo di sistema operativo sulla base delle risposte fornite da quest'ultimo; in questo caso `snort` rileva ovviamente solo le tecniche di *fingerprinting* attivo, in cui si inviano pacchetti di test, non può ovviamente accorgersi di tecniche passive come quelle usate da `ettercap` cui si è accennato in sez. 1.2.4.

Come per i preprocessori ciascun plugin di uscita supporta una serie di funzionalità e la relativa serie di opzioni di configurazione; i principali plugin sono riportati di seguito, un elenco più completo si trova nello *Snort User Manual* già citato in precedenza:

alert_syslog invia gli allarmi sul sistema del *syslog*, nella sua forma più generale richiede come opzione un indirizzo ed una porta da specificare con l'opzione `host` nella forma classica mostrata nell'esempio, seguito eventualmente dalla *facility* e dalla priorità cui piazzare gli allarmi.

log_tcpdump permette di indicare un file, come unica opzione, su cui registrare i pacchetti nel solito formato usato da `tcpdump`.

database è probabilmente il plugin più evoluto, dato che consente di registrare gli eventi di allarmi su un database, eventualmente situato anche su una macchina remota.[33]

La funzionalità più interessante di `snort`, quella che lo rende uno degli IDS più interessanti fra quelli disponibili (compresi pure prodotti proprietari molto costosi), è quella di possedere un motore di analisi per il rilevamento di traffico sospetto estremamente potente, che è possibile controllare tramite delle opportune *regole*, con le quali si possono generare degli allarmi, eseguire altre azioni, registrare i pacchetti, sulla base di una enorme lista di proprietà sia dei pacchetti stessi che dei flussi di dati che le varie funzionalità permettono di identificare, che possono venire espresse tramite un opportuno linguaggio di scripting elementare, che permette di creare condizioni e controlli anche molto complessi.

Data l'ampiezza dell'argomento lo tratteremo a parte in sez. 1.4.4, qui vale la pena solo di ricordare un'ultima direttiva `include` che serve appunto per inserire all'interno del file di configurazione altri file. Questo viene usato principalmente proprio per includere i file che contengono appunto le cosiddette *regole* di `snort`. In genere infatti sono già state prodotte un vasto insieme di regole pronte, in grado di identificare le più varie forme di attacchi possibili, che normalmente vengono fornite su una serie di file a parte, caricati appunto con questa direttiva, che ha la forma:

```
# sintassi
# include pathname
include classification.config
include reference.config
include $RULE_PATH/local.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
...
```

1.4.4 Le regole di `snort`

Come accennato la vera potenza di `snort` sta nel suo motore di rilevamento, che viene controllato da una serie di regole che vengono processate dal programma. Questo le applica ai dati che riceve,

[33] è tramite questo plugin che si possono usare sistemi di integrazione di allarmi da più fonti come ACID, che poi dal database legge i dati e genera automaticamente dei rapporti e delle pagina con tutti gli allarmi e le statistiche.

e sulla base di quanto in esse specificato è in grado di compiere una serie di azione come generare allarmi, registrare pacchetti, attivare altre regole.

Le regole vengono espresse tramite alcune direttive speciali, riportate in tab. 1.16, chiamate *azioni* (o *rule action*), ciascuna regola è introdotta da una di queste azioni, seguita dal corpo della regola; quest'ultimo è lo stesso per ciascuna di esse, seguendo una sintassi comune.

Tipo	Descrizione
`alert`	genera un allarme secondo quanto specificato nella regola e poi registra il pacchetto.
`log`	registra il pacchetto.
`pass`	ignora il pacchetto.
`activate`	genera un allarme ed attiva una regola dinamica.
`dynamic`	la regola è disattiva fin quando non viene attivata e diventa una regola di registrazione.

Tabella 1.16: Le direttive predefinite per la definizione delle regole di `snort`.

Le regole hanno un ordine preciso di scansione, prima vengono utilizzate le `alert` per generare gli allarmi, poi le `pass` per ignorare i pacchetti che non si vuole siano ulteriormente processati, e poi le `log` per registrarli. Quindi l'uso di `pass` non evita che siano generati eventuali allarmi; questo è per molti controintuitivo per cui `snort` mette a disposizione sia l'opzione `-o` a riga di comando che il parametro `order` per la direttiva `config` per cambiare le modalità di questa scansione, processando per prima le regole di tipo `pass`.

A ciascuna azione seguono una serie di campi per la selezione dei pacchetti cui la regola si applica; il primo di questi indica il protocollo con il nome utilizzato in `/etc/protocols` (al momento `snort` analizza solo TCP, UDP, ICMP, IP, ma in futuro potranno essercene altri). I campi successivi servono per specificare indirizzi e porte dei pacchetti e sono sempre nella forma:

```
ip-sorgente porta-sorgente -> ip-destinazione porta-destinazione
```

dove si può usare per ciascun termine la parola chiave `any` per indicare un indirizzo qualunque o indicare gli indirizzi sia singoli che come reti in notazione CIDR, invertire le selezioni apponendo un "`!`", o indicare liste di questi elementi separate da virgole e poste fra parentesi quadre. Per le porte invece si possono indicare degli intervalli separandoli con il carattere "`:`", un esempio allora potrebbe essere una regola del tipo:

```
alert tcp ![192.168.1.0/24,10.0.0.0/8] any -> 192.168.1.0/24 111 \
    (content: "|00 01 86 A5|"; msg: "mountd access";)
```

inoltre al posto dei valori possono essere usate delle variabili, definite secondo quanto visto in sez. 1.4.3. Infine in un criterio di selezione si può adoperare l'operatore `<>`, che indica le coppie indirizzo porta in entrambe le direzioni (consentendo così di classificare immediatamente tutto il traffico di una connessione).

Come l'esempio mostra una volta selezionato il pacchetto in base alle informazioni essenziali (le porte si devono omettere in caso di pacchetti ICMP) il resto della regola sia espresso in un ultimo capo, compreso fra parentesi tonde in cui si inseriscono una serie di *opzioni* nella forma `opzione: valore`, separate da dei "`;`" che indicano la fine dei parametri passati all'opzione. Gran parte della potenza di `snort` deriva dalle capacità di queste opzioni che sono le più varie, come

quella di analizzare il contenuto dei pacchetti selezionati, e di produrre messaggi di conseguenza, come nell'esempio.

In particolare, prima di passare all'elenco di quelle principali, sono da segnalare due opzioni, che sono usate per la particolare funzionalità indicate nell'uso delle azioni `activate` e `dynamic`. L'idea alla base di queste azioni è quella di poter creare una regola che viene attivata dinamicamente (da questo i nomi) in corrispondenza di alcuni eventi, in genere appunto per registrare tutto il traffico ad essi collegato anche quando l'evento che ha generato l'allarme è passato.

Per far questo si deve sempre specificare una coppia di regole, la prima di tipo `activate` che identifica l'evento critico (e genera pertanto anche un allarme), una tale regola ha una opzione obbligatoria, `activates`, che deve indicare un numero che identifica la regola `dynamic` da attivare. Quest'ultima a sua volta ha una opzione obbligatoria `activated_by` che indica il numero da usare per attivarla. Un esempio di una coppia di tali regole potrebbe essere il seguente:

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA; \
      content: "|E8C0FFFFFF|/bin"; activates: 1; \
      msg: "IMAP buffer overflow!\";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by: 1; count: 50;)
```

Come già detto le opzioni delle regole sono il cuore delle funzionalità di `snort`, quello che segue è un elenco delle principali con relativa spiegazione del significato e dei parametri che le controllano, e degli effetti che sortiscono nel comportamento del programma, un elenco molto più dettagliato si può trovare nello *Snort User Manual* già citato più volte:

`msg` specifica una stringa da riportare come messaggio quando utilizzata da regole di allarme o di registrazione dei pacchetti; un possibile esempio è:

```
msg: "testo del messaggio";
```

`ttl` imposta un criterio di corrispondenza sul valore del campo TTL del pacchetto, rispetto all'espressione passata come parametro. Questo permette di selezionare pacchetti in base a questo valore e può essere utilizzato per identificare l'esecuzione di un `traceroute`; un possibile esempio è:

```
ttl: 0;
ttl: >220;
```

`dsize` imposta un criterio di corrispondenza sulle dimensioni del carico dati dei pacchetti, che è molto più efficiente della analisi del contenuto per eseguire rilevazioni di eventuali tentativi di *buffer overflow*. Supporta varie combinazioni degli operatori `>` e `<`, anche se in genere si usa solo il primo; possibili esempi sono:

```
dsize: >1000;
dsize: 300<>400;
```

`fragbits` imposta un criterio di corrispondenza sulla presenza dei bit riservati dell'intestazione del protocollo IP.[34] Questi sono tre e sono usati principalmente per la

[34] la struttura dell'intestazione di IP è riportata in fig. **??**.

gestione dei pacchetti frammentati: il *Reserved Bit* (RB), normalmente non utilizzato, è selezionabile con `R`, il *Don't Fragment* bit (DF), usato per impedire la frammentazione e richiedere l'emissione di un ICMP *fragmentation needed*, è selezionabile con `D` ed il *More Fragments* bit, usato per indicare la presenza di ulteriori frammenti, è selezionabile con `M`. Specificando uno o più di questi caratteri si richiede la presenza di tutti e soli i bit corrispondenti; usando anche il carattere `+` si richiede solo tutti quelli indicati siano presenti, usando il carattere `*` si richiede che almeno uno di quelli indicati sia presente, mentre usando il carattere `!` si richiede che il flag indicato non sia presente; possibili esempi sono:

```
fragbits: M+;
```

content imposta un criterio di corrispondenza sul contenuto del carico dati del pacchetto. Il criterio comporta ovviamente una ricerca su tutto il traffico è piuttosto pesante dal punto di vista del carico sulla macchina, ma è anche la funzionalità che permette di eseguire la rilevazione di gran parte degli attacchi che in genere vengono compiuto nei confronti di server vulnerabili. Il vantaggio di questo criterio è che può essere ripetuto più volte nella stessa opzione per specificare diversi contenuti e ridurre l'incidenza di falsi positivi (una corrispondenza casuale è sempre possibile) e permette l'uso di una serie ulteriori di opzioni che permettono di estenderne il comportamento.

L'opzione prende come parametro una espressione che permette di indicare sia un contenuto binario che testuale, il contenuto deve essere passato come parametro racchiuso fra virgolette, un contenuto binario deve essere specificato come bytecode esadecimale eventualmente separato da spazi e racchiuso fra delle barre verticali, mentre un contenuto testuale può essere scritto direttamente, con l'unica accortezza di proteggere con il carattere "\" i caratteri riservati "`;`", "`"`", "`:`" e "\"; un possibile esempio è:

```
content:"|E8 C0FF FFFF|/bin/sh";
```

flags imposta un criterio di corrispondenza sul valore dei flag presenti nell'intestazione di un pacchetto TCP, utilizzando i caratteri illustrati in tab. 1.17. Come per i flag di IP indicando un gruppo di questi caratteri si richiede che siano presenti tutti e soli i flag corrispondenti, valgono le stesse estensioni fornite dai caratteri "`*`", "`+`" e "`!`" illustrate in precedenza. Inoltre in questo caso si può indicare una maschera opzionale, in cui si indichi quali degli altri flag non considerare nella corrispondenza, così da creare selezioni in cui si richiede la presenza di solo certi flag, l'assenza di altri e l'irrilevanza dei restanti. Un possibile esempio è:

```
flags:SF,12;
flags: A+;
```

ack imposta un criterio di corrispondenza sul valore del campo di *acknoweledge* di un pacchetto TCP, utilizzato in genere per riconoscere il *TCP ping* di `nmap` illustrato in sez, 1.2.1, che è contraddistinto da un valore nullo di tale campo. L'opzione richiede un valore numerico; un esempio è il seguente:

Flag	Descrizione
F	Flag FIN.
S	Flag SYN.
R	Flag RST.
P	Flag PUSH.
A	Flag ACK.
U	Flag URG.
2	Secondo bit riservato.
1	Primo bit riservato.
0	Nessun bit attivo.

Tabella 1.17: Identificatori dei flag TCP nell'opzione `flag`.

```
ack: 0;
```

itype imposta un criterio di corrispondenza sul valore del campo *type* di un pacchetto ICMP,[35] permettendo la selezione dei vari tipi di pacchetto; un possibile esempio è il seguente:

```
itype: 0;
```

icode imposta un criterio di corrispondenza sul valore del campo *code* di un pacchetto ICMP, utilizzato per gli stessi scopi del precedente; un possibile esempio è il seguente:

```
icode: 0;
```

resp permette di indicare una azione di risposta da dare in caso di traffico che corrisponde alla regola in cui è usata. L'idea è di utilizzare questa opzione per bloccare automaticamente il traffico dalle connessioni da cui si identifica un attacco. Come già sottolineato questo è molto pericoloso sia perché è facile chiudersi fuori in caso di errore, sia perché questa funzionalità può essere usata contro di noi per un DoS eseguendo apposta un attacco simulato. L'opzione prende come parametro una lista delle azioni elencate in tab. 1.18, separate da virgole; un possibile esempio è il seguente:

```
resp: rst_all
```

reference imposta una (o più se utilizzata più volte) referenza per il particolare attacco identificato dalla regola in questione, così che questa possa passare nel messaggio di allarme ed essere utilizzata per identificare la natura di quest'ultimo. L'opzione richiede due parametri separati da virgola, il nome di un ente classificatore (espresso in forma della URL del relativo sito, anche se alcuni di essi sono identificati con stringhe predefinite) ed l'identificativo usato da questo per lo specifico attacco; un possibile esempio è il seguente:

[35] una spiegazione sul significato di questi pacchetti è stata affrontata in sez.**??**, ed in tab.**??** e tab.**??** sono riportati i valori numerici dei vari campi del protocollo.

Azione	Significato
rst_snd	invia un reset TCP alla sorgente.
rst_rcv	invia un reset TCP alla destinazione.
rst_all	invia un reset TCP ad entrambi i capi.
icmp_net	invia un *ICMP Net Unreachable* al mittente.
icmp_host	invia un *ICMP Host Unreachable* al mittente.
icmp_port	invia un *ICMP Port Unreachable* al mittente.
icmp_all	invia tutti i precedenti pacchetti ICMP al mittente.

Tabella 1.18: Significato delle azioni dell'opzione `resp`.

```
reference: bugtraq,5093;
reference: cve,CAN-2002-1235;
```

sid definisce un identificatore univoco per la regola in questione, in modo che i plugin di uscita possano trattarla velocemente. I valori inferiori a 100 sono riservati, quelli da 100 a 1000000 sono utilizzati per le regole distribuite insieme a `snort`, e quelli sopra 1000000 per l'uso locale; un possibile esempio è il seguente:

```
sid:1810;
```

classtype definisce in quale classe fra tutte quelle in cui sono categorizzati i vari tipi di attacco ricade quello identificato dalla regola corrente. Una serie di valori sono predefiniti, altro possono essere definiti con la direttiva `config classification` (per una tabella completa si faccia riferimento allo *Snort User Manual*); un possibile esempio è il seguente:

```
classtype:successful-admin;
```

tag permette di sostituire le regole di tipo `dynamic` consentendo la definizione di un insieme di pacchetti da registrare oltre quelli relativi all'allarme. Con questa opzione tutto il traffico proveniente dalla sorgente che ha innescato l'allarme può essere *etichettato* e registrato. L'opzione prevede almeno tre parametri: il primo indica il tipo di traffico e può essere `host` per indicare tutto il traffico proveniente dall'IP sorgente o `session` per indicare il traffico nella stessa connessione; il secondo indica un numero di unità da registrare ed il terzo l'unità stessa (che può essere `packets` o `seconds`); un possibile esempio è il seguente:

```
tag: session,5,packets;
```

ip_proto imposta un criterio di corrispondenza sul campo di protocollo di un pacchetto IP, permettendo di selezionare pacchetti anche degli altri protocolli. L'opzione prende come parametro il valore del suddetto campo, e si può usare il carattere "`!`" per negare una selezione; un possibile esempio è il seguente:

```
ip_proto: 50
```

Appendice A

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this

License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History

section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License or any later version applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Indice analitico