La gestione della sicurezza con GNU/Linux

Simone Piccardi
piccardi@truelite.it
Revisione: 1368

Truelite Srl http://www.truelite.it info@truelite.it

Sicurezza con GNU/Linux – 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:

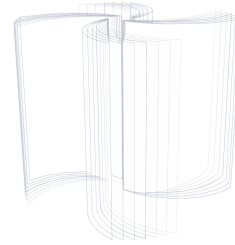
`http://svn.truelite.it/truedoc`

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

Le basi della sicurezza informatica

1.1 Introduzione ai concetti di base

In questa prima sezione faremo una breve introduzione teorica sulla sicurezza informatica, illustrando i concetti di base su cui essa è fondata, descrivendo gli attacchi e le fonti di insicurezza, ed i servizi e le funzionalità necessari alla sua salvaguardia. Infine esamineremo alcuni criteri di base necessari alla creazione di una politica di sicurezza.

1.1.1 Gli obiettivi fondamentali della sicurezza informatica

Per poter parlare di sicurezza di un sistema informatico occorre anzitutto definire cosa si intende per sicurezza, perché è solo in base a questo che si possono identificare le minacce ed i pericoli che possono metterla in discussione e poi discutere delle politiche che si intendono applicare per realizzarla.

Nel campo dell'informatica si sono spesso adottate le più varie definizioni del termine sicurezza che ne caratterizzano il significato secondo vari aspetti. Nel nostro caso utilizzeremo una definizione operativa che si basa sul soddisfacimento di tre obiettivi fondamentali, e che è ampiamente adottata in buona parte della letteratura che tratta delle problematiche di sicurezza informatica.[1]

Si dice allora che quando un sistema informatico è in grado di realizzare questi obiettivi potremo considerarlo sicuro. Questi obiettivi di base, che costituiscono quelli che considereremo come criteri essenziali per la sicurezza informatica, sono i seguenti:

confidenzialità la garanzia che solo le persone autorizzate possono accedere al sistema ed ai dati, ed operare tramite di esso e su di essi.

[1] un ottimo testo che usa questa classificazione, il cui contenuto è però molto teorico, è [CompSec].

integrità la garanzia che solo le persone autorizzate possano modificare il sistema e i dati in esso mantenute, e possano farlo solo all'interno dei limiti e delle autorizzazioni loro assegnate.

disponibilità la garanzia che il sistema è in grado di fornire i servizi previsti, e sia in grado di farlo in tempi e modalità "*ragionevoli*".

In realtà le definizioni appena riportate sono estremamente generiche ed astratte, ed il loro significato viene spesso inteso in maniera diversa a seconda degli ambienti in cui si opera, a seconda delle esigenze degli utenti, (si pensi ad esempio al significato di *ragionevoli* nella disponibilità dei servizi) o delle consuetudini locali.

La *confidenzialità* riguarda la necessità di mantenere segreti, o meglio accessibili solo dalle persone autorizzate, informazioni, risorse e dati presenti in un sistema informatico. A prima vista si potrebbe pensare che questo è di interesse solo in casi particolari, come per l'uso in campi particolarmente sensibili come quelli di applicazioni militari, amministrative o industriali o per la protezione di dati ed informazioni particolarmente sensibili.

In realtà benché questo sia un aspetto di cui è senz'altro necessario tenere conto, anche le più semplici esigenze di privacy (come quelle imposte dal Testo Unico del decreto legislativo 196/03) necessitano di meccanismi di riservatezza. Inoltre le stesse credenziali di accesso che permettono di definire gli utenti di un sistema devono essere adeguatamente protette.

La confidenzialità in sostanza esprime la necessità che l'accesso alle opportune risorse (dati ed informazioni) sia garantito solo a chi ha il diritto di utilizzarle, nelle modalità stabilite dalla propria politica di sicurezza. A livello di realizzazione tecnica questo comporta un supporto diretto sia da parte del sistema operativo che delle applicazioni.

Infine una aspetto talvolta trascurato, ma che in certi casi è molto importante, è quello della riservatezza riguardo l'esistenza stessa di certe informazioni (ad esempio quanto potrebbe rivelare ad attaccanti contromisure prese a difesa di una rete o per il rilevamento di attacchi), cioè quello che usualmente viene classificato sotto il nome generico di *information hiding*.

In genere la modalità più utilizzata per garantire la confidenzialità dei dati è quella di usare delle opportune tecnologie crittografiche (vedi sez. 1.2) in modo da renderli accessibili solo da coloro che detengono le opportune chiavi di accesso. Nel caso però di *information hiding* si suole parlare invece di *steganografia*, per indicare un insieme di tecniche con cui le informazioni oltre che ad essere opportunamente cifrate, vengono pure nascoste (in genere all'interno di altri dati apparentemente "*innocui*").

Si tenga presente comunque che non sempre è possibile cifrare tutte le informazioni (ad esempio i dati mantenuti in un database, o da esso ricavati) per esigenze di natura tecnica o realizzativa, nel qual caso si dovranno prendere provvedimenti di altra natura per garantire i requisiti di confidenzialità necessari.

L'*integrità* esprime la necessità che dati e risorse siano mantenuti ed acceduti in maniera affidabile, così che ne sia sempre garantita la validità. In particolare si deve poter essere certi sia del loro contenuto che della loro origine. In genere si interpreta questo requisito in maniera semplificata identificandolo con la capacità di prevenire operazioni di scrittura non autorizzata,[2] o richiedendo che le informazioni siano non modificabili, consistenti, o modificabili solo tramite i procedimenti appropriati, o più genericamente *accurate*.

[2] come nel rapporto [AIPAsec] dell'AIPA sulla sicurezza informatica.

In questo caso cioè si intende per integrità che i dati sono nello stato in cui sono stati ha lasciati dall'ultima operazione correttamente eseguita su di essi, e che non hanno subito altre manipolazioni, modifiche o cancellazioni non autorizzate. Normalmente quando si parla di integrità in un sistema sicuro si fa riferimento solo al poter fare affidamento che dati e informazioni siano mantenuti correttamente al riparo da distruzione o falsificazioni.

Il termine *accuratezza* precedentemente citato però ha una valenza che non è completamente esaurita dagli aspetti finora elencati, e sottintende anche una ulteriore caratteristica, che spesso viene trascurata, e cioè che i dati devono affidabili fin dall'inizio. L'integrità quindi deve anche rispondere alla esigenza di garantire la correttezza dell'origine dei dati e delle modalità con cui non solo vengono trattati, ma sono anche ottenuti.

Per questo rispetto a quanto è necessario per la confidenzialità, l'integrità richiede delle misure completamente diverse; nel primo caso l'obiettivo è impedire l'accesso ai dati, in questo caso invece deve essere garantita anche la correttezza e la loro validità ed in particolare quest'ultimo punto è estremamente critico, in quanto si basa sull'assunzione di validità che si pone all'origine dei dati stessi.

Questo rende spesso la valutazione dell'integrità dei dati estremamente difficile, in quanto deve basarsi su delle assunzioni riguardanti la loro origine e sulla fiducia che si può porre nella relativa fonte, due aspetti che sono molto spesso sono erroneamente dati per acquisiti acriticamente nelle politiche di sicurezza.

Anche per la gestione dell'integrità dei dati sono di fondamentale importanza le tecnologie crittografiche, anche se in questo caso vengono applicate in maniera sostanzialmente diversa dalla confidenzialità, e servono non tanto per cifrare e decifrare dati, quanto per calcolare una *impronta* di un certo file in un certo istante, memorizzando la quale è poi possibile eseguire una verifica ad un tempo successivo. Tratteremo gli aspetti generici di queste tecniche in sez. 1.2.4, mentre vedremo un esempio del loro utilizzo in sez. 5.3.2.

Ovviamente tecniche come quella appena descritta sono efficaci solo per informazioni che variano poco nel tempo; nel caso di basi di dati con informazioni che variano continuamente questo approccio non è adeguato e si può garantire un buon livello di integrità solo con una corretta politica di identificazione ed autenticazione degli utenti, controllo degli accessi e tracciabilità delle operazioni eseguite.

La *disponibilità* esprime la necessità di poter utilizzare i dati, le informazioni e le risorse messe a disposizione da un sistema informatico. Questo può apparire poco attinente alle questioni relative alla sicurezza, ma è ovvio che un sistema senza backup è un sistema poco sicuro, in quanto può essere messa a rischio l'esistenza stessa dei dati che gestisce.

In genere comunque nel settore specifico della sicurezza si intende per disponibilità di una risorsa o di un'applicazione la capacità di poter far si che la stessa sia accessibile e utilizzabile a ogni richiesta inoltrata da un utente autorizzato.

In genere gli attacchi rivolti a questo aspetto della sicurezza vengono chiamati *Denial of Service*, (o più brevemente DoS) e benché possano sembrare meno critici rispetto ad attacchi che compromettano l'integrità o la confidenzialità dei dati, ciò non di meno sono in grado di rendere inutilizzabili le risorse, e con questo bloccare completamente il funzionamento del sistema.

Inoltre spesso questo tipo di attacco serve come ausilio per effettuare attacchi più pericolosi, ad esempio se si è in grado di bloccare con un attacco di *Denial of Service* un sistema chiave per la propria sicurezza (ad esempio quello che garantisce un accesso cifrato) gli utenti finiranno probabilmente con l'usare un accesso non cifrato non adeguatamente protetto, renden-

do così possibile, all'attaccante che lo ha predisposto, accedere a risorse critiche altrimenti non disponibili.

1.1.2 I servizi per la sicurezza

Per poter assicurare la realizzazione di un adeguato livello di sicurezza secondo i criteri illustrati in sez. 1.1.1 un sistema informatico deve fornire una serie di servizi che gli permettano di assicurare *confidenzialità*, *integrità* e *disponibilità*.

In generale quando si parla di servizi di sicurezza la *disponibilità* non viene presa in considerazione esplicitamente, in quanto attiene più alla robustezza infrastrutturale del sistema che a dei servizi specifici, anche se ci possono essere eventuali indisponibilità dovute alle conseguenze di un accesso abusivo. Non tratteremo pertanto questi aspetti in questa sezione, ma si tenga presente che anche i servizi di sicurezza più sofisticati non ci salveranno dall'uso un sistema operativo o di programmi instabili o di scarsa qualità, o dalla mancanza di una adeguata politica di backup.

Una prima classificazione elementare dei servizi di sicurezza può essere la seguente, in cui essi sono stati suddivisi in tre classi funzionali diverse, riportate in tab. 1.1. Queste prime tre classi sono fondamentali e normalmente sono realizzate a livello di sistema operativo, anche se molte applicazioni implementano a loro volta al loro interno questi stessi meccanismi. In ogni caso questi servizi riguardano aspetti infrastrutturali, che devono essere messi a disposizione dal sistema informatico che si usa (applicazione o sistema operativo che sia).

Classe	**Descrizione**
identificazione	Risponde alla domanda *chi sei?*
autenticazione	Risponde alla domanda *come mi accerto che sei tu?*
autorizzazione	Risponde alla domanda *cosa puoi fare?*

Tabella 1.1: Le tre classi funzionali dei servizi di sicurezza.

La prima classe di servizi di sicurezza è quella dell'*identificazione*; lo scopo di questi servizi è appunto quello di identificare l'utente all'interno del sistema: questo da una parte è in genere la condizione per l'accesso al sistema, dall'altra permette di associare all'utente privilegi e capacità operative specifiche ad uso dei servizi di controllo dell'accesso.

Questi servizi possono essere realizzati in diverse modalità, la più classica è quella di associare un nome a ciascun utente (il classico *username*) che lo identificherà,[3] ma possono esistere anche sistemi alternativi, come l'uso di smart-card o di dati biometrici,[4] che permettono di identificare l'utente in base al possesso di un opportuno oggetto o alle sue caratteristiche fisiche.

La seconda classe di servizi è quella dell'*autenticazione*, il cui scopo è quello di verificare l'identità di chi intende utilizzare il sistema e che questa corrisponda ad un utente autorizzato. Si deve cioè determinare in una qualche maniera la validità dell'identificazione, tanto che spesso si tende ad considerare l'autenticazione come parte dei servizi di identificazione.

[3] nel caso di Unix questo è fatto associando ad un utente un numero identificativo detto *User ID*, che a sua volta è associato all'*username*; altri sistemi usano meccanismi più complessi, che prevedono identificativi comprendenti informazioni ulteriori, come la macchina su cui l'utente opera.

[4] in questi casi di spesso il servizio di identificazione viene unificato a quello di autenticazione.

Il meccanismo classico in questo caso è quello della password, dove l'autenticazione è eseguita attraverso una parola chiave segreta, dimostrando la conoscenza della quale si dimostra anche la correttezza della propria identità. Anche in questo caso sono disponibili meccanismi alternativi, più o meno sofisticati come le smart-card o i sistemi biometrici, o altre tecnologie che consentano comunque di poter determinare con ragionevole certezza l'identità di un utente; l'ambito del "*ragionevole*" definisce appunto quella zona grigia sui contorni della quale si concentrano tanti attacchi alla sicurezza.

In generale per i servizi di autenticazione si fa un uso significativo di tecnologie crittografiche, le password ad esempio vengono controllate tramite hash crittografici, mentre altri meccanismi come quelli usati da SSL e SSH comportano l'uso di chiavi asimmetriche (vedi sez. 1.2.3).

La terza classe di servizi è infine quella dell'*autorizzazione*, cioè quella che mira a stabilire cosa l'utente può fare o non fare all'interno del sistema, e ad cui si fa spesso riferimento anche come *controllo degli accessi*. Questa è in genere la parte più corposa e complessa dei servizi di sicurezza, in quanto ci sono diversi modelli di realizzazione che riflettono anche diverse concezioni delle metodologie applicate.

Lo schema classico di controllo degli accessi è quello chiamato *Discrectionary Access Control* o DAC; in questo caso viene definito il proprietario di un dato ed è lui a decidere quale tipo di accesso gli altri utenti possono avere dello stesso. In questo caso sono i singoli utenti che governano gli accessi ai propri dati e decidono quali, in un elenco di operazioni definite possono essere consentite agli altri utenti.

Il caso classico di questa tipologia è quello dei dati mantenuti in filesystem, e dei permessi usualmente associati ai file. In genere in uno schema DAC si articola in una serie di permessi che il proprietario può assegnare ad altri utenti e gruppi, lo schema classico di Unix è limitato ad una assegnazione generalizzata dei permessi a gruppi o a tutti gli utenti, ma si possono creare anche schemi più complessi con l'uso delle *Access List*, che permettono di assegnare permessi specifici ai singoli utenti e gruppi. Anche in questo caso comunque è sempre il proprietario del file che decide le modalità di accesso.

Il limite dello schema del *Discrectionary Access Control* è che non si possono imporre limiti alla disponibilità del dato al proprietario dello stesso; esistono infatti situazioni in cui si vuole poter definire delle politiche di accesso che limitino anche il proprietario di un dato nelle operazioni consentite. Un esempio è quello in cui si vuole evitare che anche in caso di compromissione del sistema in cui un attaccante ha avuto accesso allo stesso, questo possa compiere delle operazioni che si sono stabilite essere comunque illegittime.

Un caso classico in cui ci si scontra con questo problema è quello in cui si vuole poter limitare le capacità dell'amministratore del sistema di compiere certe operazioni. Questo comporta la necessità non solo di associare dei permessi e dei controlli ai singoli dati, come nel DAC, ma anche alle varie operazioni su di essi eseguibili.

Per rispondere a questi problemi è stato elaborato il cosiddetto *Mandatory Access Control* o MAC, in cui si estende il concetto dei privilegi di accesso non solo al contenuto dei singoli dati, ma alle varie operazioni che possono essere eseguite nel sistema. Questo comporta ad esempio la possibilità di usare o meno un certo programma per accedere a certi specifici dati o per compiere certe specifiche operazioni.

Inoltre esiste il cosiddetto *Role-Based Access Control* o RBAC, che rappresenta un approccio nuovo ed alternativo rispetto sia al *Discrectionary Access Control* che al *Mandatory Access Control*. In questo caso l'idea è quella della creazione di *ruoli* all'interno del sistema, a cui

sono assegnate certe capacità specifiche. Solo gli utenti inseriti all'interno di un ruolo potranno eseguire le relative operazioni.

Questo approccio, eliminando la necessità di assegnare ad ogni utente permessi e privilegi, semplifica notevolmente la gestione della sicurezza, in quanto basterà definire gli opportuni ruoli e poi assegnare gli utenti ad uno di essi. La semplificazione è però apparente, in quanto la difficoltà viene comunque spostata nel compito di creare e definire adeguatamente le capacità di accesso dei singoli ruoli.

Sia MAC che RBAC richiedono comunque, rispetto al caso tradizionale in cui i permessi erano direttamente associati ai dati, un supporto molto più complesso da parte del sistema operativo, che deve consentire di inserire controlli e verifiche non solo a livello di accesso ai dati, ma anche all'interno delle operazioni da eseguire su di essi. Questo permette in controllo molto più raffinato, ma ovviamente aumenta anche la complessità di gestione e con questo la probabilità di errori, ed in genere diminuisce anche le prestazioni.

1.1.3 La classificazione dei rischi e delle minacce

Rispetto ai tre obiettivi fondamentali della sicurezza illustrati in sez. 1.1.1, si tende a fornire una classificazione teorica dei rischi e delle minacce ad essi portate suddividendo questi ultimi nelle quattro classi generiche illustrate in tab. 1.2.

Classe	Descrizione
disclosure	la possibilità di un accesso non autorizzato ad informazioni e dati riservati di un sistema.
deception	la possibilità di inserire o far accettare dati o informazioni false o errate in un sistema.
disruption	la possibilità di interrompere o compromettere il corretto funzionamento delle operazioni di un sistema.
usurpation	la possibilità di prendere il controllo di un sistema (o di una parte dello stesso) senza averne il diritto.

Tabella 1.2: Le quattro classi astratte delle minacce alla sicurezza.

Come si può notare quella di tab. 1.2 è una classificazione molto astratta, relativa a caratteristiche completamente generiche, che si è ordinata per gravità crescente della minaccia. In generale però quando si vanno a classificare le caratteristiche funzionali dei vari tipi di attacco questi ricadono in più di una delle classi precedenti; per questo c'è chi tende ad usare una categorizzazione delle minacce diversa, secondo il seguente elenco:

intercettazione detta in gergo *snooping*, questa categoria indica sia i rischi relativi all'intercettazione non autorizzata di informazioni che l'accesso non autorizzato a dati e documenti.

Questo tipo di minacce ricade nella classe della *disclosure*, e può essere implementato con varie metodologie: dalla intercettazione eseguita con mezzi fisici (il cosiddetto *wiretapping*), all'uso di software specifici come gli *sniffer* sulla rete (vedi sez. 5.2.2), a sistemi di monitoraggio delle operazioni degli utenti: in generale ci si basa su tecniche di tipo passivo.

Qualora si intercettino informazioni essenziali come le credenziali degli utenti (ad esempio le loro password) questo tipo di minaccia può anche portare a rischi ulteriori, fino alla *usurpation*. In generale si risponde a questo tipo di minacce con contromisure basate sulla crittografia che consentano di cifrare il traffico o i dati, vale a dire attraverso servizi dedicati al mantenimento della confidenzialità.

Alterazione si coprono in questa categoria le minacce relative un vasto insieme di attacchi miranti a modificare in maniera non autorizzata dati ed informazioni. Questo di nuovo porterebbe questa categoria nella classe della *deception*, ma qualora si tratti di dati usati nel controllo del sistema di nuovo si potrebbero avere conseguenze in termini sia di *disruption* che di *usurpation*.

Di norma gli attacchi usati in questa categoria di minacce si basano su tecniche attive, il caso esemplare è quello dell'intercettazione attiva, in cui si modifica il contenuto dei dati. I cosiddetti attacchi di "*man-in-the-middle*", in cui l'attaccante si interpone fra due interlocutori per presentarsi come l'altro a ciascuno di essi, rientrano in questa tipologia. In generale si risponde a questo tipo di minacce con contromisure volte a verificare l'integrità dei dati.

Mascheramento detto in gergo *spoofing*, è la categoria di minacce basata sul presentarsi con una diversa identità, e pertanto attiene principalmente alla classe della *deception*, ma qualora si riesca ad impersonare un amministratore o un utente privilegiato può sconfinare nella *usurpation*. Appartiene a questa categoria il *phishing*, cioè il redirigere gli utenti su siti web che sembrano quelli di altre entità per ottenere informazioni di valore, mentre con *spoofing* si tende ad indicare in maniera più specifica la falsificazione degli indirizzi dei pacchetti di rete.

Sebbene in teoria possano essere passivi, attendendo semplicemente che l'utente si colleghi ad un servizio falsificato, per lo più gli attacchi in questa categoria sono di natura attiva, con scenari in cui l'attaccante si presenta sotto false credenziali o si spaccia per altri. Di nuovo si risponde a questo tipo di minacce con contromisure volte a verificare l'integrità, che in questo contesto sono essenzialmente servizi di autenticazione che mirino a confermare l'identità degli attori in gioco.

Blocco o ritardo appartengono a questa categoria gli attacchi che mirano a ritardare l'uso dei servizi, a o bloccarli completamente. La si può considerare una forma particolare di *usurpation*, in quanto tende a manipolare le strutture informatiche per ottenere i suoi risultati, ma spesso gli attacchi di questa categoria vengono usati come supporto per attacchi di *deception* in cui si tenta di sostituirsi al sistema che si è bloccato per ingannare l'utente.

Gli attacchi in questa categoria sono sempre di natura attiva e sfruttano in generale delle debolezze dell'infrastruttura bersaglio; ma si tenga pre-

sente che in blocco dei servizi può derivare anche da debolezze intrinseche o difetti strutturali del sistema che ne causino il malfunzionamento. Si risponde a questa categoria di minacce con servizi mirati a garantire la disponibilità del sistema.

In generale un sistema informatico deve essere in grado di garantire protezioni da tutte queste minacce; per questo tutti i sistemi operativi moderni e molte applicazioni forniscono degli opportuni servizi di sicurezza (vedi sez. 1.1.2) il cui scopo è esattamente quello di proteggersi contro questi rischi.

Se il mondo forse perfetto i servizi di sicurezza sarebbero infallibili, i programmi non avrebbero errori e vulnerabilità e gli utenti si comporterebbero sempre in maniera corretta e non malevola. Il mondo però non è perfetto, i programmi hanno errori, i sistemi presentano malfunzionamenti e gli utenti possono essere scorretti, per cui alle minacce illustrate in precedenza corrispondono sempre una serie di attacchi possibili che possono realizzarle concretamente.

Per capire meglio allora quali sono i rischi che si vanno a correre è opportuno fornire una ulteriore classificazione, questa volta andando a esaminare i vari tipi di attacchi che vengono usati nella pratica per violare la sicurezza di un sistema informatico; un breve elenco dei principali è il seguente, che ci servirà anche come glossario:

Exploit è una espressione gergale che copre tutti quegli attacchi che sfruttano delle vulnerabilità presenti nei programmi dovute ad errori di programmazione, anche se talvolta la si usa per indicare semplicemente il successo nell'intrusione abusiva in un sistema attraverso una di queste vulnerabilità. In genere quello che succede è che inviando al programma dei dati in ingresso in una forma che non era stata presa in considerazione dal programmatore, si ha un errore.

Questo può portare semplicemente al crash del programma, nel qual caso si ha un *denial of service*, ma spesso è possibile, usando dei dati opportunamente forgiati, fare eseguire al programma del codice malevolo (il codice dell'*exploit*,[5] appunto) con risultati che possono andare dalla sovrascrittura di un file al completo controllo di una macchina.

Gli errori più comuni definiscono a loro volta ulteriori sottocategorie di attacchi. I più noti sono i cosiddetti *buffer overflow*, in cui inviando più dati di quanti previsti, in assenza degli opportuni controlli, si eccede la dimensione di un buffer precedentemente allocato consentendo l'esecuzione dell'*exploit*; questo attacco è descritto con estrema chiarezza in un articolo classico [StS] di Phrack fin dal 1996, e nonostante ciò si continuano ad avere problemi di questo tipo. Altri tipi di *exploit* sono gli *integer overflow* dove lo stesso effetto è ottenuto per mancati controlli sulle dimensioni di un intero,[6] gli *string format* in cui si abusa della formattazione in uscita di un dato.

[5] spesso questo è semplicemente un cosiddetto *shellcode*, cioè un programma elementare il cui unico scopo è eseguire una shell per dare accesso al sistema compromesso.

[6] in questo caso si usa un intero che viene usato come indice per poter eseguire sovrascritture di dati in zone di memoria non previste.

In generale si tende a dividere gli *exploit* in locali e remoti, a seconda che per poterli realizzare sia necessario avere un accesso sulla macchina sul quale eseguire il programma vulnerabile o sia sufficiente eseguire l'attacco su una connessione via rete. Ovviamente i rischi maggiori sono dovuti a questi ultimi.

In generale si tende ad inserire in questa categoria anche alcuni dei più comuni problemi della sicurezza informatica come i *virus* ed i *worm*. In sostanza si tratta della stessa tipologia di attacco per cui da una vulnerabilità di un programma si ottiene la possibilità di eseguire del codice il cui scopo in questo caso è quello di propagarsi verso altri computer. La differenza principale fra le due forme è che un *virus* in genere cerca di inserirsi all'interno di altri programmi e si attiva con l'esecuzione di questi ed agisce localmente, mentre un *worm* agisce principalmente via rete.

Backdoor si chiama così un attacco che consente di oltrepassare delle misure di protezione attraverso appunto una *porta sul retro*. In realtà non si tratta di un attacco, se per questo si intende una azione diretta volta a effettuare una violazione di sicurezza, quanto piuttosto di un meccanismo per mantenere insicuro un sistema. In genere ha due significati diversi a seconda del contesto. Quando riferito ad un algoritmo crittografico si intende la possibilità di decifrare i messaggi anche senza conoscere la chiave con cui sono stati cifrati;[7] quando riferito ad un sistema informatico si intende del codice che viene inserito al suo interno per permettere un accesso scavalcando i normali criteri di autenticazione.

Il secondo è il caso più comune, ed è il metodo che viene usato molto spesso da chi si introduce abusivamente in un sistema dopo un *exploit* per garantirsi un accesso, in genere da remoto, anche quando la via originaria che ne aveva consentito l'ingresso non è più disponibile, ad esempio perché il programma vulnerabile ad un *exploit* è stato aggiornato.

Nel caso di sistemi Unix questi vengono in genere chiamati *rootkit* perché servono a garantire un accesso da amministratore, ed oltre ad una *backdoor* prevedono anche ulteriori meccanismi volti a mascherare la presenza di quest'ultima. Nel caso Windows esistono programmi analoghi; quando la *backdoor* viene inserita all'interno di un altro programma, che si cerca di far eseguire all'utente, in genere si parla di *trojan*.[8]

Una *backdoor* comunque non è detto debba derivare direttamente dall'installazione abusiva di un programma apposito da parte di un intruso. Molti programmi infatti contengono delle *backdoor* inserite dagli stessi programmatori che li hanno realizzati, messe lì per consentire loro un accesso diretto. Nel caso di software libero sono stati scoperti tentativi di

[7] è il caso del famoso *clipper chip*, un processore crittografico che negli Stati Uniti era stato proposto per cifrare tutte le comunicazioni, e che prevedeva la possibilità di avere una chiave speciale che potesse decifrarle sempre, ad uso dell'FBI, che, anche per questo motivo, alla fine non è mai stato dispiegato.

[8] il nome deriva ovviamente dall'idea del cavallo di Troia.

inserimento di *backdoor* (in particolare ne è stata scoperta una all'interno di un database proprietario di cui sono stati rilasciati i sorgenti, presente presumibilmente da molti anni), ma nei vari casi in cui sono state trovate sono state anche immediatamente rimosse; per i programmi proprietari, per i quali non si ha l'accesso al codice, non si potrà mai essere certi della loro assenza, mentre sono noti molti esempi concreti di una loro presenza.

Inoltre si tenga presente che è possibile creare delle *backdoor* anche senza modificare direttamente il codice di un programma. L'esempio classico è quello creato da Ken Thompson, uno dei creatori di Unix, considerato fra i più grandi scienziati nel campo dell'informatica, che aveva inserito all'interno del compilatore del C[9] un meccanismo per cui veniva inserita automaticamente una *backdoor* tutte le volte che questo veniva usato per compilare il programma di accesso al sistema. Era anche stata creata una versione dello stesso compilatore in grado di replicare questo comportamento nei compilatori creati tramite il compilatore originale.

In questo modo nel codice sorgente, a parte quello del compilatore originale da cui si era iniziato il processo, non vi sarebbe stata alcuna traccia della *backdoor*, rilevabile solo ad una analisi, molto meno probabile e comunque molto più complessa da fare, del codice eseguibile. Questo esempio illustra in maniera molto chiara quanto accennato in sez. 1.1.1, per cui è essenziale, per valutare l'integrità di un sistema, il grado di fiducia che si può riporre nell'origine di dati ed informazioni e negli strumenti per il loro utilizzo.

Intercettazione corrispondono alla omonima categoria di minacce e sono gli attacchi che mirano a ottenere informazioni, in genere relative alle credenziali di accesso o a dati riservati, intercettando le comunicazioni. Benché usualmente in questo caso si tenda a pensare a problematiche non strettamente attinenti all'ambito informatico e più vicine al mondo dello spionaggio, come microfoni, microspie, intercettazioni telefoniche, telecamere nascoste, ecc. occorre comunque valutare questo tipo di attacchi; le più accurate misure di sicurezza informatica servono a poco infatti se è possibile ad esempio intercettare il segnale di una telecamera che riprende la tastiera di chi inserisce una password.

Inoltre esistono tecnologie specifiche, realizzate proprio per intercettare i dati all'interno di sistemi informatici, come le reti e gli stessi computer. Un esempio sono i *keylogger*, apparecchi[10] in grado di registrare tutte le pressioni dei tasti eseguite su una tastiera, o il sistema *Tempest* che è in grado di intercettare le emissioni elettromagnetiche dei monitor e replicare a distanza quello che appare sullo schermo.

[9] il C è il linguaggio con cui è stato creato Unix, e con cui sono realizzati la gran parte dei sistemi operativi e dei programmi di sistema.

[10] per analogia si chiamano così anche i programmi che forniscono la stessa funzionalità via software, spesso presenti nei *rootkit*.

Non è inoltre detto che si debba arrivare all'utilizzo di apparecchiature di intercettazione specializzate, esistono infatti anche programmi dedicati all'intercettazione che possono essere eseguiti su un qualunque computer, come gli *sniffer* (di cui parleremo in sez. 5.2.2) che permettono di intercettare il traffico che passa su di una rete locale,[11] o i *keylogger* o altri programmi dedicati che possono intercettare tutte le operazioni compiute su un desktop.[12]

Benché molti dei problemi relative all'intercettazione possano essere risolti con l'uso della crittografia per rendere inutilizzabili le informazioni intercettate, occorre comunque essere consapevoli che questa è una soluzione parziale. Per quanto sia sicuro un algoritmo crittografico esso è normalmente soggetto ai cosiddetti attacchi di *man-in-the-middle* in cui si intercettano tutte le informazioni scambiate fra due capi di una comunicazione presentandosi a ciascuno dei due come l'altro. Esistono modalità per ridurre questo rischio, ma in generale non è possibile cancellarlo completamente.

Ingegneria sociale per quanto raffinato possa essere un sistema di controllo, per quanto accuratamente possa essere stato scritto il software, gli essere umani possono possono sempre sbagliare, ma soprattutto essere ingannati, usando quella che in gergo viene chiamata *social engineering*. Per cui si può utilizzare la più stretta delle politiche di accesso ed il controllo più stretto sui firewall e l'antivirus più potente, ma se si convince la segretaria del capo a far girare un programma spacciandosi per il nuovo sistemista della sede secondaria che manda un aggiornamento, e questo in realtà è un programma che permette di accedere alla rete interna, tutte le precauzioni prese saranno inutili.

Gli attacchi di *social engineering* sono senz'altro quelli più difficili da contrastare, proprio perché non sono categorizzabili e non hanno a che fare con l'organizzazione di un sistema informatico, ma con quella delle persone. Le uniche contromisure possibili pertanto sono l'educazione e la formazione, ma anche queste non possono coprire tutte le possibilità. Inoltre rendere estremamente rigide le politiche di sicurezza rischia anche di essere controproducente, in quanto diventa molto più facile che queste, per la loro scomodità, vengano ignorate dalle persone, lasciando così più spazi di quanto non ci sarebbero con politiche più *sostenibili*.

1.1.4 I criteri per una politica di sicurezza

Probabilmente il criterio fondamentale che deve essere compreso per poter affrontare il tema della sicurezza è quello che Bruce Schneier, crittografo e matematico, architetto dell'algoritmo

[11] il rischio è ancora più forte se la rete è wireless e per inserirsi non è neanche necessaria la presenza fisica all'interno dei locali.

[12] ad esempio il programma VNC, spesso usato per l'amministrazione remota, può essere usato anche a questo scopo.

Blowfish, uno dei maggiori esperti mondiali in ambito di crittografia e sicurezza informatica, ha espresso con la frase: "*la sicurezza è un processo, non un prodotto*". Non esiste cioè un prodotto che possa risolvere una volta per tutte le problematiche di sicurezza, ma queste devono essere affrontate all'interno di un processo che preveda una loro valutazione, la valutazione dei relativi rischi, dei costi delle contromisure, ed una continua verifica e ridispiegamento e affinamento delle misure adottate.

Si tratta cioè di realizzare che il primo passo da fare per affrontare il tema è quello di definire una *politica di sicurezza*, cioè definire le proprie priorità, le linee guida per realizzarle e le modalità per verificarne l'effettività. Un elenco di concetti da tenere ben presenti quando si vuole impostare una politica di sicurezza è il seguente:

- *la sicurezza assoluta non esiste*; se un attaccante è sufficientemente determinato e ha risorse sufficienti potrà superare qualunque misura di sicurezza abbiate realizzato, ad esempio puntandovi una pistola alla testa... Pertanto chiunque propagandi la nuova rivoluzionaria soluzione a tutti i problemi della sicurezza vi sta vendendo "*olio di serpente*".[13]

- *il miglioramento della sicurezza è asintotico*; questo significa che si possono ottenere buoni risultati di sicurezza anche con investimenti modesti, ma più si vuole elevare il livello di sicurezza più i costi aumentano. Per questo una valutazione da fare prima di realizzare una qualunque misura di sicurezza è quella dell'analisi dei rischi che si vogliono coprire e dei costi che si intendono affrontare. Spesso per ottenere un piccolo miglioramento della sicurezza si devono affrontare dei costi maggiori, sia in termini di infrastrutture che di organizzazione, di quelli sostenuti per creare l'infrastruttura esistente.

- *la sicurezza e la facilità d'uso sono esigenze contrastanti*; poter lasciare la porta aperta è molto più comodo che portare tre mazzi di chiavi e ricordarsi dieci combinazioni diverse. Perché una politica di sicurezza sia efficace occorre che le sue misure siano praticabili e ragionevoli. Potete imporre password composte di lettere casuali lunghe 20 caratteri da cambiare ogni mese, ma rischiate che la gente per ricordarsele se le scriva su un foglietto sotto la tastiera.

Per poter avere una indicazione dei passi da fare nell'implementazione di una politica di sicurezza faremo riferimento allo standard ISO/BS-17799 relativo all'adozione di un sistema di gestione della sicurezza delle informazioni. Il processo previsto dallo standard è piuttosto complesso dal punto di vista organizzativo e burocratico, simile come concetti a quelli della certificazione di qualità ISO-9000. Quello che interessa però non sono tanto i dettagli specifici quanto i concetti generali che possono servire come linee guida per la creazione di una politica di sicurezza.

Il punto fondamentale chiarito dallo standard è che la gestione della sicurezza deve essere vista come parte integrante del funzionamento dei propri processi interni; ci deve cioè essere una decisione strategica in cui si sceglie di occuparsi delle problematiche relative alla sicurezza e si fa sì che queste vengano considerate all'interno di tutte le procedure organizzative.

Un secondo punto significativo è che lo standard adotta il modello di sicurezza come processo, formalizzato un modello ciclico detto PDCA (dall'inglese *Plan-Do-Check-Act*), i cui concetti fondamentali sono riassumibili nei quattro punti:

[13] traduzione improvvisata dell'espressione gergale "*snake oil*", che origina dalle medicine miracolose dei *western* che guarivano qualunque cosa.

Pianifica è fondamentale capire quali sono i requisiti di sicurezza sulla base delle proprie attività, e deve essere chiara la necessità di stabilire politiche e obiettivi; occorre cioè una analisi dei rischi ed una pianificazione della gestione della sicurezza.

Fai occorre implementare le opportune misure di sicurezza nel contesto delle normali procedure operative; devono cioè essere integrate nei propri processi organizzativi e funzionali.

Controlla occorre tenere sotto controllo la congruità delle politiche di sicurezza, il funzionamento sul campo delle misure adottate ed infine verificarne continuamente l'efficacia e le prestazioni.

Agisci sulla base dei controlli effettuati occorre migliorare continuamente processi e misure di sicurezza, ripetendo da capo il ciclo di pianificazione, implementazione, verifica, aggiornamento.

Il concetto fondamentale sottolineato in questo schema è che non basta pianificare ed implementare una politica di sicurezza; questi sono solo i primi due passi, dopo i quali essa deve anche essere tenuta sotto controllo, valutata e continuamente aggiornata.

In genere il primo passo nella pianificazione di una politica di sicurezza è quello di eseguire una analisi dei rischi. La valutazione del rischio è probabilmente una delle parti più complesse dell'operazione: devono essere identificati i beni che si vogliono proteggere, in questo caso dati ed informazioni gestiti attraverso i sistemi informatici ed essere identificate e valutate le minacce.

Alla fine la valutazione consiste nel prodotto fra il valore dei beni soggetti a rischio e la probabilità che esso possa concretizzarsi, e per rischi e minacce occorre anche considerare non solo gli attacchi deliberati ma anche i possibili incidenti e le eventuali catastrofi naturali. La difficoltà della procedura sta appunto nell'identificare tutte queste variabili. Non trattando qui delle metodologie per l'analisi del rischio, non entreremo ulteriormente in detto argomento.

Una volta individuati i beni ed i rischi, occorrerà valutare le misure, organizzative e funzionali, che si possono porre in atto per eliminare o ridurre gli stessi ad un valore ritenuto ragionevole. Non è detto che per un qualunque rischio si debba prevedere una misura di sicurezza, fra le decisioni possibili infatti c'è sempre quella di accettare il rischio (ad esempio quando le misure dovessero avere un costo superiore al bene da proteggere), così come quella di trasferire ad altri il rischio (ad esempio facendo una assicurazione).

Il secondo passo della pianificazione è quello della individuazione delle procedure funzionali relative alle misure di sicurezza che si vogliono mettere in atto; queste in genere vengono suddivise in tre categorie generali:

Prevenzione spesso ci si limita a pensare a questo aspetto, quello che cioè mira a prevenire una violazione di sicurezza come procedimenti di autorizzazione, firewall, antivirus, ecc. Ovviamente questa è la parte principale di un qualunque insieme di misure di sicurezza, ma dato che la sicurezza assoluta non esiste, per una politica di sicurezza sostenibile sono necessarie anche misure nelle due categorie successive.

Rilevamento per ridurre al minimo i danni causati da una eventuale violazione di sicurezza è cruciale intervenire il più tempestivamente possibile; per questo uno degli

aspetti cruciali di una qualunque politica di sicurezza resta quello delle misure prese per rilevare eventuali violazioni (IDS, analisi dei log, scanner) in modo da minimizzare i tempi di intervento.

Recupero infine una volta che si è subita una violazione di sicurezza è indispensabile poter disporre di una strategia che permetta di recuperare l'integrità e l'affidabilità dei sistemi e delle informazioni; un aspetto che spesso non viene adeguatamente pianificato. Rientrano in questa categoria le procedure per il backup e il *disaster recovery*.

Infine nella creazione di una politica di sicurezza occorrerà anche definire le modalità con cui si gestiscono di dati ed informazioni ed i relativi criteri di accesso, cioè la parte che attiene alla identificazione dei processi di gestione e dei ruoli di coloro che sono incaricati di eseguirli. In questo caso si fa riferimento ad alcuni criteri generici che sono chiamati i *pilastri* della sicurezza, intesa dal punto di vista organizzativo:

Classification risponde alla necessità di identificare dove è richiesta una maggiore sicurezza. Si applica in genere ai requisiti di confidenzialità e prevede una classificazione verticale di informazioni e sistemi a seconda della loro criticità e di utenti e processi in base al loro livello di affidabilità. Un incaricato o un processo deve poter accedere ad un dato di un certo livello di sensibilità solo se ha un corrispondente livello di affidabilità.

Need to know risponde alla necessità di definire quali sono i dati e le informazioni a cui un utente o un processo deve poter accedere per potere eseguire i suoi compiti. In questo caso la classificazione è di tipo orizzontale e attiene alle divisioni operative (comparti, funzioni, ecc). In questo caso si tratta di identificare quale è lo stretto necessario allo svolgimento dei compiti assegnati e fornire accesso solo a quello, evitando così possibili abusi di privilegi non necessari.

Separation of duties risponde alla necessità di suddividere organizzativamente alcuni compiti, in modo da minimizzare gli effetti di possibili errori o frodi. In sostanza si tratta di separare alcuni compiti (il caso classico è chi sta alla cassa e chi conta i soldi a fine giornata) in modo che la suddivisione di responsabilità renda molto più difficile un atto fraudolento.

1.2 Crittografia e sicurezza

Daremo in questa sezione una introduzione molto generica alle basi della crittografia, descrivendone i concetti generali e illustrando le funzionalità e le caratteristiche dei vari algoritmi e delle varie tecniche crittografiche che vengono utilizzate per applicazioni relative alla sicurezza.

1.2.1 Introduzione

La crittografia[14] è una disciplina antichissima, e riguarda l'arte di trasmettere messaggi il cui contenuto sia stato opportunamente "*cifrato*" in modo che a chi esegue una eventuale intercettazione essi risultino inintelligibili, mentre restino comprensibili per il legittimo destinatario che sia in grado di *decifrarne* il contenuto.

Uno degli esempi più antichi è quello usato dai condottieri cretesi nelle campagne militari, che prevedeva messaggi scritti su strisce di stoffa avvolte su un cilindro di legno di un diametro fissato, il cui contenuto diveniva leggibile solo quando riavvolte sul cilindro del destinatario.

In generale si pensa alla crittografia come ad una tecnica che attiene al mantenimento della confidenzialità delle informazioni, oggi però si tende a richiedere anche qualcosa in più; per questo una definizione utilizzata comunemente per la crittografia moderna è quella data da Bruce Schneier in [ApplCrypt]:

> la crittografia è "*la disciplina che studia la trasformazione di dati allo scopo di nascondere il loro contenuto semantico, impedire il loro utilizzo non autorizzato, o impedire qualsiasi loro modifica non rilevabile*"

In generale qualunque tecnica crittografica si avvale di un *algoritmo crittografico* che attraverso una serie di trasformazioni consente il passaggio (la *cifratura*) da un *testo in chiaro*, contenente il messaggio che si vuole spedire, ad un *testo cifrato* da trasmettere effettivamente su un canale su cui può essere intercettato. Chi riceverà il testo cifrato dovrà poi fare il passaggio inverso (cioè *decifrare*) per ricavare il testo in chiaro.

Tradizionalmente la crittografia si basava sulla segretezza del metodo utilizzato per cifrare e decifrare i messaggi, secondo lo schema illustrato in fig. 1.1; uno degli esempi classici è quello del *cifrario di Cesare* che consiste nello scrivere un messaggio sostituendo a ciascuna lettera quella che la segue di tre posti nell'ordine alfabetico.[15] Una volta noto l'algoritmo decifrare un messaggio è banale.

Al giorno d'oggi nessuno si basa più su un sistema crittografico fondato sulla segretezza di un algoritmo; questo perché ci sono degli evidenti svantaggi dal punto di vista pratico nel sostituire un algoritmo in caso di scoperta, o nell'usarne uno diverso per ogni canale di comunicazione che si vuole mantenere separato. Inoltre la verifica della effettiva validità dell'algoritmo, cioè della sua capacità di rendere davvero inintelligibile il testo cifrato, non può che essere fatta in ambiti molto ristretti, visto che non ne può essere divulgato il funzionamento senza comprometterlo.

Per questo fin dal secolo scorso è incominciato a diventare chiaro che era necessario puntare su una diversa concezione che prevedesse algoritmi di cifratura pubblici le cui capacità di rendere illeggibile un messaggio si basassero solo sulla segretezza di una *chiave* di cifratura. In questo caso il testo cifrato viene prodotto dal testo in chiaro attraverso delle manipolazioni che prevedono la presenza di una "*chiave*",[16] nota la quale è possibile riottenere il testo in chiaro. Senza la chiave anche con la conoscenza dell'algoritmo di cifratura deve essere praticamente impossibile riottenere il testo in chiaro (vedremo più avanti cosa si intende per praticamente).

[14] la parola deriva dal greco e significa "*scrittura segreta*".

[15] che è ancora in uso, come semplice tecnica per oscurare un testo, nel cosiddetto *ROT-13* in cui la rotazione è fatta su tredici posti invece che su tre.

[16] esistevano già algoritmi di cifratura di questo tipo, in fondo anche nel caso dell'antico metodo dei condottieri cretesi c'era una chiave, che era il bastone cilindrico.

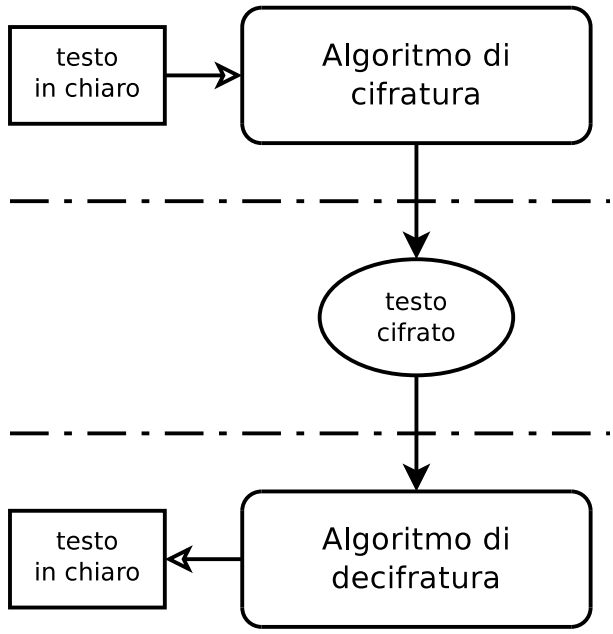


Figura 1.1: Schema di funzionamento di un algoritmo crittografico tradizionale.

Un'analisi dettagliata degli algoritmi di crittografia va ben al di là degli scopi di queste dispense, ma una breve introduzione generica è necessaria per approfondire la comprensione delle varie applicazioni che di essa si fanno nel campo della sicurezza. In particolare è utile comprendere quali sono i criteri generali che permettono di stabilire la validità di un algoritmo crittografico.

Fra questi, come appena accennato, ce n'è uno da scartare in partenza: la segretezza. Periodicamente c'è chi proclama di disporre di un nuovo segretissimo algoritmo crittografico supersicuro. Dato che è segretissimo però nessun crittografo esperto potrà mai dirvi se è sicuro davvero. I veri algoritmi sicuri sono quelli che sono risultati tali dopo essere stati analizzati in tutti i loro dettagli, e lo sono per la loro robustezza intrinseca e non per il non sapere come sono realizzati.

La disciplina che studia la robustezza degli algoritmi crittografici si chiama *crittanalisi*; i criteri di base per cui si considera un algoritmo sicuro sono quelli che gli permettono di resistere alle diverse tipologie di attacco da parte dei *crittanalisti*. Questi attacchi vengono classificati nelle quattro categorie seguenti:

- *attacchi a testo in cifra noto*: è l'attacco più semplice, in cui si analizzano un certo numero di messaggi cifrati per ricavarne le informazioni necessarie a realizzare una delle violazioni che illustreremo a breve. In genere si tratta di attacchi su basi statistiche, che cercano di rivelare informazioni sul testo o sulla chiave utilizzando il fatto che nel testo in chiaro non tutti i caratteri compaiono con la stessa frequenza. In questo caso basta intercettare i messaggi cifrati, ed in genere più se ne intercetta più è facile eseguire questo tipo di

attacco.

- *attacchi a testo in chiaro noto*: in questo caso oltre ai messaggi cifrati si ha a disposizione anche il testo in chiaro che li ha generati. Ovviamente questo tipo di attacco comporta nella pratica una esecuzione più complessa, dovendosi ottenere, oltre ai messaggi cifrati anche la relativa conoscenza del testo in chiaro.

- *attacchi a scelta di testo in chiaro*: rispetto al caso precedente in questo tipo di attacchi si è in grado non solo di conoscere alcuni messaggi passati sul canale, ma di scegliere anche il loro contenuto. Sebbene questo tipo di attacco possa sembrare impossibile da realizzare in pratica sono noti molti casi in cui lo si è eseguito con relativa facilità, ad esempio fornendo informazioni note che si sapeva sarebbero passate sul canale cifrato.

- *attacchi a scelta adattativa di testo in chiaro*: oltre a quanto previsto nel caso precedente qui si richiede anche che sia possibile scegliere il testo dei messaggi da cifrare in base ai messaggi cifrati ottenuti per quelli scelti in precedenza. È ovviamente il più complesso da realizzare in quanto suppone la capacità di poter inserire arbitrariamente del testo sul canale cifrato.

A fronte di ciascuno di questi attacchi sono possibili vari livelli di successo, quello che in gergo si chiama una *rottura* dell'algoritmo. Anche questi casi sono classificati, e vengono utilizzate le seguenti quattro categorie generiche:

- *rottura totale*: si riesce a determinare la chiave di cifratura, e si ha così l'accesso completo a tutto quanto viene cifrato e la capacità di creare messaggi cifrati a nostra volta.

- *deduzione globale*: si riesce a individuare un algoritmo alternativo che decodifica tutti i messaggi anche senza conoscere la chiave.

- *deduzione locale*: si riesce a decifrare alcuni messaggi.

- *deduzione di informazione*: si riesce ad avere una qualche informazione sul testo in chiaro o sulla chiave.

Oltre a quelli citati esiste un ulteriore attacco che è sempre possibile e che porta sempre ad un successo totale, l'attacco detto *a forza bruta*. Questo consiste semplicemente nel provare tutte le chiavi possibili fintanto che dal testo cifrato non si riottiene un testo in chiaro. Ovviamente benché questo attacco sia sempre possibile, la scelta di una chiave sufficientemente complessa permette di renderlo inutile sul piano pratico.

Un algoritmo crittografico robusto deve essere allora in grado di resistere a tutti gli attacchi delle categorie precedentemente esposte per tutti i livelli di successo appena illustrati. In pratica questo significa che un buon algoritmo di crittografia è un algoritmo che rende il tentativo di decifrare il messaggio senza la chiave dello stesso ordine di difficoltà di un attacco a forza bruta.

Si tenga presente però che in molti casi, quando la chiave è una *parola chiave*, si può utilizzare una versione molto efficace dell'attacco a forza bruta, detta *attacco a dizionario*, in cui invece di provare tutte le chiavi possibili, se ne prova un numero molto più limitato, fra quelle presenti in un dizionario; se la chiave scelta non è sufficientemente *complessa* il compito si semplifica notevolmente e spesso è un gioco da ragazzi trovarla. In questo caso non è l'algoritmo ad essere debole, quanto la chiave stessa, e questo è un problema che c'è sempre con qualunque algoritmo.

Un esempio elementare di algoritmo debole è quello dell'enigma cifrato della settimana enigmistica, in questo caso la chiave è la tabella delle corrispondenze fra lettere e numeri. Come narrato da Edgar Allan Poe nel racconto *Lo Scarabeo d'oro*, e come chiunque si diletti di enigmistica avrà notato, basta fare una semplice analisi statistica dei numeri che ricorrono di più per individuare le lettere principali del testo, e con un po' di tentativi diventa semplice decifrare completamente il messaggio.

Un algoritmo che apparentemente sembrerebbe forte, tanto che a lungo è stato ritenuto indistruttibile, è il cosiddetto cifrario di Vigenère[17] in cui si usa una parola chiave per cifrare un testo. Il meccanismo prevede che si affianchino il testo e la parola chiave, ripetuta ciclicamente per tutta la lunghezza del testo, prendendo come risultato della cifratura di ciascuna lettera del testo la lettera che lo segue nell'alfabeto di tante posizioni quante sono quelle della lettera corrispondente della parola chiave.[18]

Per quanto possa sembrare difficile da decifrare, con una sufficiente quantità di messaggi questo cifrario è suscettibile di analisi statistiche non particolarmente complesse.[19] Non può comunque essere considerato robusto essendo completamente vulnerabile ad attacchi con testo in chiaro, nel qual caso diventa immediato trovare la chiave.

Nella crittografia moderna sono stati ideati vari algoritmi di cifratura; tutti si basano su elaborazioni attinenti ad una certa classe di problemi matematici. Alcuni algoritmi hanno mostrato, in seguito a successive analisi, diversi gradi di debolezza, ma quelli di uso più comune non ne hanno finora dimostrata alcuna. Dal punto di vista teorico però non esiste nessuna dimostrazione che detti algoritmi siano effettivamente robusti. Tutto quello che si può dire è che allo stato attuale delle conoscenze non esiste alcun modo, né si ritiene che possa esistere, per romperli, non si ha però la certezza matematica di tutto ciò, con una eccezione.

Esiste infatti, ed è noto da molto tempo, un algoritmo matematicamente indistruttibile, chiamato OTP (*One Time Pad*). Questo consiste semplicemente nell'avere una chiave completamente casuale più lunga del messaggio stesso, con la quale questo viene cifrato. Fintanto che la chiave è più lunga del messaggio, è completamente casuale e viene usata *una sola volta* non esiste modo, se non la conoscenza della chiave stessa, di decifrare il messaggio.

Benché sia stato usato in varie occasioni, il problema di base di questo algoritmo è che sul piano del suo utilizzo pratico ci sono delle difficoltà enormi consistenti nel doversi scambiare precedentemente quantità di chiavi sufficienti a coprire tutto quello che si cifrerà in seguito, per cui, per banali esigenze di natura gestionale, quando si parla di crittografia si fa sempre riferimento ad uno degli algoritmi che tratteremo nelle sezioni seguenti.

1.2.2 La crittografia a chiave simmetrica

La crittografia moderna nasce in sostanza nel dopoguerra con il lavoro di Claude Shannon,[20] uno dei più grandi geni matematici del ventesimo secolo, i cui lavori han posto i fondamenti della

[17] questa è una attribuzione tardiva fatta nel 1700, in realtà il meccanismo è stato ideato nel 1553 da Giovan Batista Belaso.

[18] se cioè la lettera del testo è `C` e quella della chiave `B` il risultato sarà `E`.

[19] anche questo cifrario ha una sua notorietà letteraria, essendo stato usato da Jules Verne in vari suoi racconti.

[20] che in un articolo del 1949 (vedi [ShannonCrypt]) ha posto le basi teoriche della crittografia moderna.

teoria delle comunicazioni. Ma a lungo lo studio di questa materia è stato mantenuto segreto dai governi, in particolare nel caso degli Stati Uniti era appannaggio della NSA.[21]

Una classificazione molto grossolana degli algoritmi di crittografia moderni è quella che li divide in due categorie principali, gli algoritmi a *chiave simmetrica*, che vedremo in questa sezione, e quelli a *chiave asimmetrica*, che vedremo nella prossima.

Il primo algoritmo crittografico moderno distribuito pubblicamente è il DES (*Data Encryption Standard*), proposto dalla IBM in risposta ad un bando dell'allora *National Bureau of Standards* statunitense per la creazione di un sistema di comunicazione sicura ad uso di banche e grandi organizzazioni finanziarie.[22]

Il DES è l'esempio classico di quello che viene chiamato un *algoritmo a chiave simmetrica*, e funziona secondo lo schema illustrato in fig. 1.2. Questa è la tipologia di algoritmo crittografico più nota, quella a cui si pensa naturalmente quando si parla di crittografia. La *crittografia a chiave simmetrica* si basa sull'uso di una chiave unica condivisa fra mittente e destinatario, che per questo la si dice simmetrica, da mantenere segreta. La conoscenza della chiave permette sia di cifrare che di decifrare il messaggio, senza di essa il messaggio è incomprensibile.

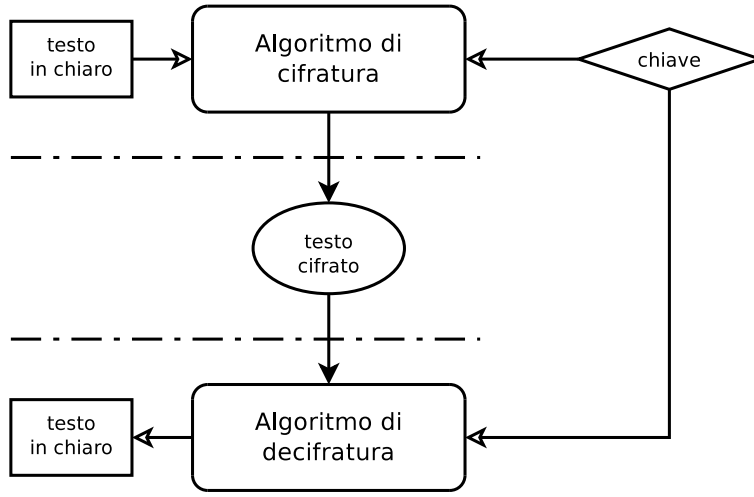


Figura 1.2: Schema di funzionamento di un algoritmo crittografico a chiave simmetrica.

Il DES appartiene ad una classe di algoritmi detti *block cipher* (*cifrari a blocchi*); questi prevedono che la cifratura avvenga per blocchi di dati di dimensione fissa (la dimensione tipica è di 64 bit) che vengono cifrati con l'applicazione dell'algoritmo di cifratura. Questo comporta che il testo in chiaro deve essere di dimensione multipla del blocco di dati elementare. Spesso non è così e deve essere per questo completato con degli opportuni bit di riempimento, detti *padding*.

[21] acronimo di *National Security Agency*, che per la sua segretezza è stata sarcasticamente chiamata anche *No Such Agency*.

[22] l'algoritmo ha una storia curiosa, in quanto una prima versione era stata considerata troppo "forte" per cui ne fu ridotta la dimensione della chiave; inoltre esso fu rivisto dalla NSA e a lungo c'è stato il sospetto, dimostratosi infondato ad analisi successive, che vi fosse stata inserita una qualche *backdoor*.

I *block cipher* hanno il vantaggio di poter essere facilmente parallelizzati, ma presentano, se usati senza accortezza, il notevole svantaggio di essere facilmente soggetti ad attacchi di crittanalisi basati sul fatto che due blocchi uguali di testo in chiaro producono lo stesso testo cifrato.[23] Per ovviare a questo tipo di attacchi si possono però usare meccanismi di *feedback*, il più comune è il cosiddetto CBC (*Cipher Block Chaining*), prevede che ciascun blocco in chiaro, prima di essere cifrato, venga opportunamente "*sommato*"[24] con il blocco in cifra calcolato al passo precedente.[25] In questo modo allo stesso testo in chiaro corrisponderanno sempre testi cifrati diversi e scorrelati fra di loro.

Una seconda classe di algoritmi a chiave simmetrica sono i cosiddetti *stream cipher* (*cifrari a flusso*), che in contrapposizione ai precedenti *block cipher*[26] possono operare su un flusso indistinto di dati. In genere il meccanismo di funzionamento di uno *stream cipher* è quello della generazione di una sequenza *pseudocasuale*[27] di bit, che vengono usati per cifrare il testo in chiaro con una opportuna operazione matematica (in genere uno XOR aritmetico).

Il più noto degli algoritmi a chiave simmetrica resta il DES, che si è dimostrato robusto; esso però soffre di un grave problema dovuto alla dimensione troppo ridotta della chiave. Questo al giorno d'oggi rende insicuro questo algoritmo, non tanto per la presenza di vulnerabilità nello stesso, quanto per il fatto che con la potenza di calcolo attualmente disponibile è relativamente facile eseguire un attacco a forza bruta in tempi ragionevoli.

Essendo nota questa debolezza del DES negli anni sono stati introdotti altri algoritmi di crittografia, che prevedessero delle dimensioni delle chiavi di cifratura di lunghezza adeguata a rendere totalmente impraticabili attacchi a forza bruta. Un elenco degli algoritmi più noti è il seguente:[28]

IDEA sviluppato nel 1991 all'Istituto Federale Svizzero di Tecnologia nasce come rimpiazzo del DES. Prevede una chiave di 128 bit ed è considerato molto sicuro anche se piuttosto lento. Inoltre è brevettato e questo porta ad ulteriori problemi (legali) di implementazione.

3DES il *Triple DES* è un tentativo di estendere il DES ampliando la lunghezza delle chiavi. Il meccanismo prevede l'utilizzo del DES con tre chiavi diverse (ciascuna di 56 bit, per un totale di 168), applicate con una opportuna sequenza sul testo.

blowfish algoritmo progettato da Bruce Schneier e considerato robusto, il suo unico problema è la dimensione della chiave di 64 bit, considerata un po' troppo breve. I suoi vantaggi sono la velocità nelle operazioni di cifratura e decifratura e la semplicità realizzativa.

[23] questa classe di attacchi vanno sotto il nome di *reply block*.

[24] l'operazione matematica usata è lo XOR aritmetico, che prevede che per ciascun bit dei due operandi la "*somma*" dia 1 se questi sono diversi e 0 se sono uguali.

[25] inoltre per il primo blocco viene usato un valore casuale, detto *vettore di inizializzazione* che viene trasmesso insieme al messaggio.

[26] in realtà esistono procedimenti di *feedback* (CFB e OB) che permettono di trasformare un qualunque *block cipher* in uno *stream cipher*.

[27] si chiama così un numero apparentemente casuale ottenuto però attraverso un preciso algoritmo matematico attraverso il valore di un numero iniziale, detto *seme*; nel caso di uno *stream cipher* il seme corrisponde alla chiave, e la sequenza generata deve essere crittograficamente robusta secondo i criteri illustrati in sez. 1.2.1.

[28] sono tutti *block cipher*, lo *stream cipher* più noto è RC4, che però presenta forti debolezze crittografiche ed il suo uso è pertanto sconsigliato.

AES è l'algoritmo, precedentemente noto come *Rijndael* dal nome dei suoi autori, scelto dal governo USA come sostituto del DES dopo un procedimento di ricerca e standardizzazione durato 5 anni. Prevede diverse possibili lunghezze di chiavi, le più comuni sono 128, 192 e 256 bit. È considerato molto sicuro, tanto che le versioni con chiavi più lunghe sono state dichiarate valide anche per informazioni classificate come "*TOP SECRET*".

Gli algoritmi a chiave simmetrica sono ben conosciuti, e quelli citati sono, allo stato delle conoscenze attuali, affidabili dal punto di vista della robustezza crittografica, essi però hanno in comune un difetto di base: occorre che il mittente ed il destinatario siano in possesso della stessa chiave.

Si pone pertanto il problema di come questi possano scambiarsela in modalità sicura: se infatti si dispone di un canale di comunicazione che è già sicuro non ha senso usare la crittografia. Pertanto, in tutte le situazioni in cui non è possibile essersi scambiati la chiave in maniera sicura prima di iniziare a comunicare, la crittografia a chiave simmetrica non assicura nessuna sicurezza aggiuntiva rispetto a quella del canale su cui ci si scambia la chiave.

Un secondo grave problema è la gestione stessa delle chiavi quando la comunicazione non è più fra due soli estremi, ma fra molti attori indipendenti. Se si usa la stessa chiave per tutti gli attori il rischio di scoperta della stessa viene moltiplicato per il numero degli attori, e basta che una sola comunicazione sia compromessa, o che uno degli attori non sia fidato, perché si compromettano tutte le comunicazioni fra tutti gli attori.

Viceversa se si decide di usare chiavi diverse si pone il problema pratico che ciascun attore deve possedere le chiavi per comunicare con tutti gli altri, ed il numero delle chiavi da gestire cresce in maniera esponenziale,[29] rendendone ingestibile il procedimento di distribuzione.

Inoltre più si usa una chiave (simmetrica), più materiale si fornisce ad un attaccante per cercare di decifrarla, rendendo più facile il compito e più esteso il danno in caso di scoperta. Per questo una chiave dovrebbe essere cambiata spesso, ed il problema della gestione delle chiavi non può essere considerato secondario.

1.2.3 Crittografia a chiave asimmetrica

La soluzione ai problemi della crittografia a chiave simmetrica illustrati alla fine della precedente sezione è venuta nel 1976 con l'invenzione da parte di Whitfield Diffie e Martin Hellman (e di Ralph Merkle sul cui lavoro essi si erano basati) della crittografia a *chiave asimmetrica*, che ha modificato radicalmente i criteri di funzionamento degli algoritmi crittografici rispetto a come erano stati fino ad allora concepiti.

Il concetto alla base della crittografia a chiave asimmetrica, illustrato in fig. 1.3, è che si usano sempre delle coppie di chiavi. Una chiave viene usata per cifrare, e l'altra per decifrare, come un forziere con una serratura con due chiavi, di cui una può solo chiudere e l'altra può solo aprire. Il meccanismo viene detto asimmetrico proprio perché un capo della comunicazione può solo trasmettere e l'altro solo ricevere.

Per questo quando si parla di crittografia a chiave asimmetrica si parla sempre di *chiave pubblica*, quella che viene usata per cifrare, che viene pubblicata, che chiunque può usare, e la cui

[29] in effetti si è esagerato un po', per pignoleria si deve dire che il numero di chiavi diverse necessarie per n attori cresce con $n(n-1)/2$, che non è un andamento esponenziale, ma quadratico; questo comunque non lo rende meno ingestibile.

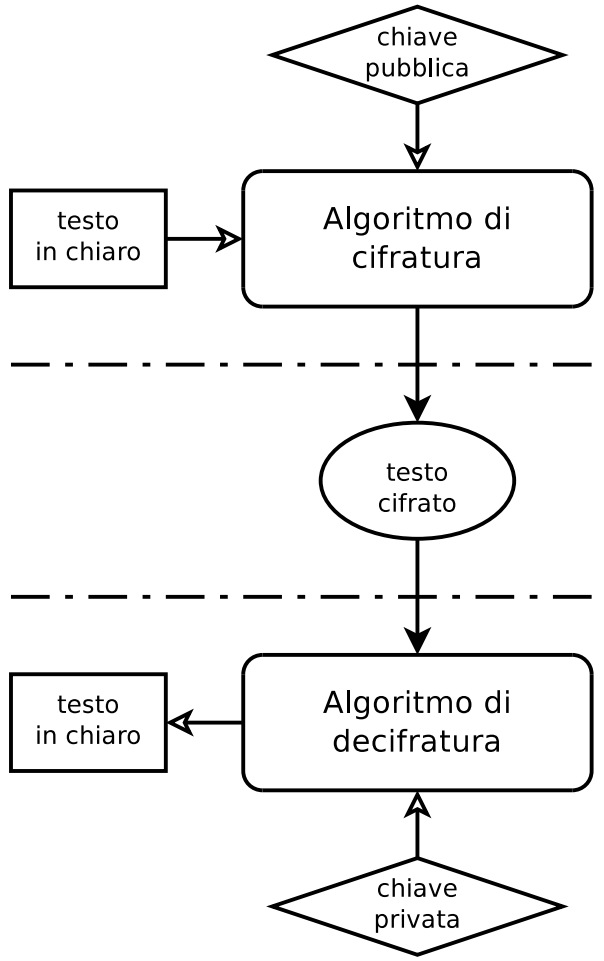


Figura 1.3: Schema di funzionamento di un algoritmo crittografico a chiave asimmetrica.

scoperta non comporta alcun rischio, e di *chiave privata*, quella che serve per decifrare, da tenere gelosamente custodita e da proteggere con la massima sicurezza, la cui scoperta compromette completamente la segretezza della comunicazione.

In realtà l'algoritmo originale di Diffie ed Hellman non è propriamente un cifrario a crittografia asimmetrica, quanto un algoritmo (denominato *procedura di Diffie-Hellman*),[30] che consente di stabilire in maniera sicura fra i due capi di una comunicazione un segreto crittografico con-

[30] la trattazione dettagliata di questa procedura va al di là dello scopo di questo testo, ma banalizzando brutalmente si può dire che il concetto è che i due capi della comunicazione estraggono un numero a caso che spediscono al corrispondente opportunamente "*moltiplicato*" con la propria chiave privata e la sua chiave pubblica; rimettendo insieme il tutto e "*fattorizzando*" opportunamente le chiavi, entrambi ottengono un valore identico combinazione dei due numeri a caso, che diventa la chiave di sessione (per una spiegazione meno vaga si consulti ad esempio `http://it.wikipedia.org/wiki/Diffie-Hellman`).

diviso (la cosiddetta *chiave di sessione*, da usare con crittografia a chiave simmetrica) senza che questo venga mai trasmesso sul canale di comunicazione e senza che sia possibile ottenerlo ad un terzo che intercetti la comunicazione stessa.[31] Il primo algoritmo di vera e propria cifratura a chiave asimmetrica venne realizzato, sempre nel 1976 da Rivest, Shamir ed Aldeman e venne chiamato RSA, dalle loro iniziali.

Benché concettualmente il metodo di funzionamento sia completamente diverso, anche per gli algoritmi a chiave asimmetrica valgono i criteri di sicurezza di robustezza rispetto agli attacchi illustrati in sez. 1.2.1; in questo caso la chiave che si cerca di determinare è ovviamente la chiave privata, e fra le informazioni disponibili si avrà ovviamente la chiave pubblica. Un elenco dei principali algoritmi è il seguente:

RSA è il primo algoritmo a chiave asimmetrico creato, la sua sicurezza è ritenuta equivalente alla difficoltà della fattorizzazione di numeri interi di grande dimensione, problema matematico per il quale al momento non è nota nessuna soluzione semplice[32] per cui l'algoritmo è considerato robusto. Non è però neanche stato dimostrato che tale soluzione non esista, per cui non si può avere la certezza della robustezza.

El-Gamal è basato sul calcolo di esponenziali di numeri interi, e la sua sicurezza si basa sulla difficoltà matematica del problema dei logaritmi discreti. La versione originale presenta una debolezza nei confronti di attacchi a testo in chiaro scelto, ma esistono versioni modificate che non lo sono.

DSA il *Digital Signature Algorithm* è un algoritmo proposto dal NIST (*National Institute of Standards and Technology*) come standard per le firme digitali dal governo statunitense. La versione originale usa una matematica analoga a quella di *El-Gamal*, ma ne esiste una versione basata sulle proprietà delle curve ellittiche, un'altra classe di altri problemi matematici ritenuta più difficile di quella dei logaritmi discreti. L'uso delle curve ellittiche consente in genere l'uso di chiavi più piccole, ma comporta tempi più lunghi nella cifratura.

Il grande vantaggio della crittografia a chiave asimmetrica, che costituisce una vera e propria rivoluzione rispetto alla crittografia a chiave simmetrica, è che è possibile distribuire senza problemi la chiave pubblica, mettendo a disposizione di chiunque la capacità di spedire un messaggio cifrato che solo il destinatario, quello che possiede la chiave privata, può decifrare.

La forza dei meccanismi di crittografia a chiave asimmetrica sta nel fatto che pur conoscendo la chiave pubblica non è possibile risalire alla chiave privata, né decifrare quanto con essa è stato cifrato. Solo la conoscenza della chiave privata consente la decifrazione. Ovviamente per poter ottenere una comunicazione bidirezionale occorre usare una coppia di chiavi ciascuno.

Lo svantaggio principale degli algoritmi a chiave asimmetrica è che le chiavi devono avere dimensioni nettamente superiori a quelle di chiavi simmetriche di forza equivalente, e che il procedimento di cifratura è nettamente più esigente in termini di risorse e molto più lento nell'esecuzione.

[31] e a questo scopo viene a tutt'oggi utilizzato da praticamente tutti i protocolli di comunicazione che prevedono la cifratura del traffico di rete.

[32] nel caso semplice vuol dire computazionalmente più semplice dell'attacco a forza bruta.

Per questo motivo oggi tutti i sistemi di comunicazione cifrata sono sistemi ibridi, in cui si usa la crittografia a chiave asimmetrica per creare un canale sicuro su cui scambiare la chiave simmetrica che serve poi per cifrare il grosso del traffico, anzi in genere si usa direttamente la procedura di *Diffie-Hellman* per creare una chiave simmetrica condivisa in maniera sicura. In questo modo si risolve il problema della trasmissione delle chiavi simmetriche usate per la comunicazione, inoltre il meccanismo permette di cambiare le chiavi di cifratura con grande frequenza, rinegoziandole attraverso il canale sicuro creato con le chiavi asimmetriche.

In sostanza il grande vantaggio degli algoritmi a chiave asimmetrica è che non soffrono del problema relativo alle modalità in cui realizzare la distribuzione sicura delle chiavi, dato che non esiste più il problema di comunicare in maniera sicura la propria chiave (quella pubblica ovviamente), in quanto renderla nota è quanto serve al funzionamento stesso del meccanismo.

Inoltre basta che ciascuno curi la conservazione della sua chiave privata per mantenere la sicurezza di quanto viene comunicato a lui, senza riflessi, in caso di compromissione, sulle comunicazioni destinate ad altri. Questo significa che quando è necessario mantenere diversi canali di comunicazione fra un numero arbitrario di persone basterà che ciascuna di esse abbia la sua coppia di chiavi; il problema cioè cresce linearmente con il numero delle persone ed è pertanto gestibile con relativa semplicità.

La crittografia a chiave asimmetrica sembra pertanto essere in grado di fornire la soluzione al problema teorico che sta alla base della crittografia contemporanea, quello di mettere in grado due persone[33] (convenzionalmente indicate dai nomi *Alice* e *Bob*) che non si sono mai incontrate né si conoscono di comunicare in maniera sicura, senza che un terzo (indicato convenzionalmente con il nome *Charlie*) possa intercettarne la comunicazione.[34] La crittografia a chiave asimmetrica sembra una soluzione, in quanto basta che Alice e Bob si scambino la chiave pubblica ed il gioco è fatto.

In realtà questo non risolve il problema, perché restano sempre possibili gli attacchi *man-in-the-middle*, in cui Charlie si spaccia per Bob con Alice e per Alice con Bob, ed essendo in mezzo può decifrare quanto Alice manda a lui credendo che sia Bob (e che invierà a Bob fingendosi Alice) e viceversa. Resta dunque aperto il problema dell'identificazione, perché niente consente di dire che la chiave pubblica che io ricevo sia davvero di Alice, a meno di non fare una verifica diretta, metodo che però fuoriesce dall'ambito della crittografia.

È comunque possibile, utilizzando le PKI (*infrastrutture di chiave pubblica*, che vedremo in sez. 2.1) fornire una soluzione, anche se solo parziale, al problema degli attacchi *man-in-the-middle*. Infatti una delle caratteristiche più interessanti della crittografia a chiave asimmetrica è quella che consente anche di realizzare la cosiddetta *firma digitale*, che vedremo in sez. 1.2.4. La direzione di cifratura illustrata in fig. 1.3 infatti può essere invertita; si può cioè cifrare con la chiave privata e generare un messaggio che potrà essere decifrato solo con la chiave pubblica. In questo caso chiaramente non si avrà un messaggio confidenziale, dato che chiunque potrà decifrarlo con la chiave pubblica, ma si potrà dimostrare di essere in possesso della corrispondente chiave privata, essendo questo l'unico modo con cui si può creare tale messaggio.

[33] in generale il problema riguarda i due capi di un canale di comunicazione, anche se la formulazione più comune è proprio questa.

[34] si, le iniziali dei nomi hanno un significato, neanche tanto velato, (per una lista più completa si consulti `http://en.wikipedia.org/wiki/Alice_and_Bob`) quello che ancora non si riesce a capire è perché diamine Alice e Bob abbiano tutta questa voglia di chiacchierare in segreto senza neanche conoscersi, e cosa importi a Charlie di spiarli.

1.2.4 Le funzioni di hash crittografico

Finora abbiamo trattato delle tecniche crittografiche applicandole esclusivamente ad una classe di problemi; quelli della *confidenzialità* dei dati. In realtà le tecniche crittografiche vengono usate in tutti gli ambiti della sicurezza delle comunicazioni, in particolare esse possono essere utilizzate anche per verificare l'*integrità* delle informazioni e che i dati non siano stati modificati, per realizzare meccanismi di *autenticazione* che identifichino gli utenti e di *non ripudiabilità* che consentano di impedire che il mittente di un messaggio possa negare di averlo spedito.

Per realizzare queste ulteriori funzionalità le tecniche crittografiche prevedono una ulteriore classe di algoritmi, relativi ai cosiddetti *hash crittografici*, o *funzioni ad una via*. Una funzione di *hash* è in generale una funzione che dato un ingresso qualunque (in genere un testo di dimensione arbitraria) genera un valore in uscita, detto appunto *valore di hash*, di dimensione fissa (ed in genere ridotta) che viene usato come indice per identificare il testo in ingresso.

In genere per una funzione di *hash* la caratteristica più importante è quella di non generare *collisioni*, vale a dire di dare valori in uscita completamente diversi in corrispondenza di testi in ingresso diversi (anche se questi differiscono poco). Spesso le si chiamano anche *checksum* in quanto vengono utilizzate per un controllo elementare di integrità.

Rispetto ad una *checksum* o ad una normale funzione di *hash* una funzione di *hash crittografico* richiede alcune ulteriori proprietà che ne consentano l'uso a scopi crittografici. Se si indica con m il testo iniziale e con h il valore dell'hash, e con f la funzione di *hash* tale che $h = f(m)$, queste proprietà sono:

- la funzione deve essere *ad una via* (o a *senso unico*), una volta cioè che sia noto soltanto il valore di hash $h = f(m)$ deve essere difficile risalire da esso al valore di m. Questa proprietà è chiamata *preimage resistence*.
- noto un certo testo iniziale m_1 ed il suo valore di hash $h = f(m_1)$ deve essere difficile trovare un m_2 (con $m_2 \neq m_1$) che dia lo stesso hash (cioè tale che $f(m_2) = h$). Questa proprietà è chiamata *second preimage resistence*.
- infine deve essere molto difficile trovare due testi qualsiasi[35] m_2 e m_1 diversi, ma tali da avere lo stesso valore di hash (cioè con $f(m_2) = f(m_1)$). Questa proprietà è chiamata *collision resistence*.[36]

Le prima proprietà è necessaria ad esempio per far sì che una funzione di hash crittografico possa essere utilizzata per realizzare meccanismi di *autenticazione* (il caso classico è quello delle password di un sistema Unix). In tal caso in genere si calcola il valore di hash della password confrontandolo con uno precedentemente memorizzato; se corrisponde si ha l'autenticazione. Ovviamente in questo caso anche potendo leggere il valore di hash deve essere molto difficile poter ottenere da questo la password originaria.

Il secondo impiego delle funzioni di hash è quello della verifica di *integrità*; in questo caso la seconda proprietà ci assicura che una volta associato ad un testo il suo valore di hash non sarà possibile mettere al suo posto un testo diverso che generi lo stesso valore. Rispetto alle classiche

[35] si tenga presente che questa, anche se apparentemente simile, in realtà esprime una condizione molto più stringente della precedente.

[36] questo requisito è diverso rispetto a quanto serve per una funzione di hash normale, in cui le collisioni devono essere solo poco probabili, ma può anche essere relativamente facile trovare testi diversi che le generano.

funzioni *checksum* si vuole avere la certezza che non sia possibile, non solo in caso di errore, ma anche in caso di costruzione deliberata, ottenere lo stesso valore di hash per un testo diverso.

Un passo ulteriore è quello in cui usando una funzione di hash ed una chiave secreta è possibile creare un *Message Authentication Code* (MAC). L'algoritmo usato più comunemente, ad esempio nei vari protocolli di comunicazione sicura come SSL e IPSEC, si chiama HMAC; in questo caso si utilizza una opportuna combinazione del testo e della chiave per la generazione di un hash crittografico che consente di ottenere il valore di un codice che non solo dimostra l'integrità del messaggio, ma anche il possesso della chiave segreta, e pertanto ne autentica il contenuto.

Infine un ultimo impiego delle funzioni di hash crittografico è, quando vengono unite all'uso della crittografia a chiave asimmetrica, quello della *firma digitale*, grazie alla quale non solo si ha la verifica dell'integrità di un messaggio e della identità del mittente, ma anche la *non ripudiabilità*.[37]

In questo caso dato un testo si genera un valore di hash che poi viene cifrato con la propria chiave privata, distribuendo quanto ottenuto (la firma digitale appunto) insieme al documento si dimostra che quel documento è stato scritto da noi (in quanto possessori della chiave privata che ha generato la cifratura) e che esso è integro. Inoltre dato che non è possibile negare che sia stata usata la nostra chiave privata, né che il valore di hash possa essere relativo ad un documento diverso, si avrà anche la non ripudiabilità dello stesso.

Come per gli algoritmi di crittografia a chiave simmetrica e asimmetrica, esistono anche varie funzioni di hash crittografico; le più usate al momento sono le seguenti:

MD5 è una funzione molto utilizzata (ad esempio per le password su GNU/Linux) creata nel 1991 da Rivest (lo stesso di RSA). Deriva da una serie di altre funzioni di *Message Digest* precedentemente create da Rivest (in particolare MD4) che avevano dimostrato seri problemi. La funzione produce un valore di hash di 128 bit. Nel 2004 sono stati trovati seri problemi anche per questa funzione, ed è pertanto opportuno programmare un passaggio ad altre funzioni, visto che i problemi si sono ulteriormente aggravati negli anni seguenti e ad oggi[38] questo algoritmo non è più considerato sicuro.

SHA-1 la sigla sta per *Secure Hash Algoritm*, pensato come sostituto più sicuro del precedente MD5, è stato progettato dalla NSA ed è pubblicato come standard per il governo statunitense e produce un valore di hash di 160 bit. Viene usato da molti protocolli ed applicazioni come GPG, SSH, SSL, IPSEC. La versione originale (SHA-0) non è considerata sicura ed è stata ampiamente sostituita da SHA-1; anche per quest'ultima però sono stati recentemente (Febbraio 2005) rilevati problemi che potrebbero ridurne l'efficacia. Esistono comunque altre versioni che producono valori di hash più ampi (SHA-256, SHA-512) che possono fare da sostituti.

RIPMED-160 la sigla sta per *RACE Integrity Primitives Evaluation Message Digest* ed è stato sviluppato in Europa da un gruppo di crittografi e pubblicato nel 1996.

[37] è quello che fanno, con la posta elettronica, programmi come GPG (*Gnu Privacy Guard*, vedi sez. 2.2.2), una reimplementazione libera e aderente allo standard del più noto PGP (*Pretty Good Privacy*), un programma che consentiva appunto di cifrare e firmare i messaggi di posta elettronica.

[38] nota riferita all'aprile 2010.

Nasce come sostituzione del precedente RIPMED-128, che risultava debole, e prevede un valore di hash di 160 bit. La sua caratteristica è quella di essere stato progettato in maniera aperta in ambito accademico. È meno usato e per questo meno studiato.

Capitolo 2

Le applicazioni della crittografia

2.1 Infrastrutture di chiave pubblica

Come primo passo nell'affrontare la trattazione delle varie applicazioni della crittografia ai servizi di sicurezza, è necessario introdurre alcuni concetti di base sulle cosiddette *infrastrutture di chiavi pubbliche* dato che queste vengono a costituire un quadro concettuale comune utilizzato, sia pure in modi diversi, da tutte quelle applicazioni che usano la crittografia sia per la riservatezza dei dati, che per i servizi di identificazione ed autenticazione.

2.1.1 Breve introduzione alle *PKI*

Uno dei punti chiave di qualunque politica di sicurezza è quello della gestione dei servizi di identificazione e di autenticazione già illustrati in sez. 1.1.2; un tentativo di risolvere in maniera generica questo problema è quello effettuato tramite l'uso delle cosiddette *infrastrutture di chiavi pubbliche*, comunemente note sotto la sigla *PKI*, che sta per *Public Key Infrastructure*.

Abbiamo visto in sez. 1.2.3 e sez. 1.2.4 come la crittografia chiave asimmetrica oltre a fornire un meccanismo molto comodo per la gestione di comunicazioni riservata, permette, quando abbinata alle funzioni di hash crittografico, di realizzare funzionalità di verifica dell'integrità. Questo tipo di utilizzo può essere esteso ed organizzato in maniera da fornire servizi di autenticazione, identificazione e firma digitale.

L'idea è che dimostrando di avere la chiave privata corrispondente ad una certa chiave pubblica posso dimostrare la mia identità, cosa che può essere utilizzata con meccanismi di *challenge and response*,[1] per effettuare una autenticazione.[2] Al contempo l'uso delle funzioni di hash crittografico combinato alla cifratura con la chiave privata, permette di ottenere la firma digitale, in cui allo stesso tempo si dimostra sia la provenienza che l'integrità delle informazioni.

[1] si chiamano così quelle tecniche che prevedono una *sfida* a cui bisogna saper dare una *risposta*: usualmente il procedimento viene eseguito tramite l'invio di un segreto (in metodo più comune è quello di un numero casuale cifrato con la chiave pubblica del destinatario), che bisogna restituire decifrato, (che nel caso si può fare solo con il possesso della chiave privata).

[2] come ad esempio si fa con l'autenticazione a chiavi di SSH.

Fintanto che si resta in un ambito in cui è possibile eseguire un controllo diretto della corrispondenza fra una coppia di chiavi ed il relativo possessore non serve nient'altro, ma quando si cerca di applicare tutto questo in maniera generica nei confronti di chiunque, riemerge il problema degli attacchi di *man-in-the-middle* di cui abbiamo parlato in sez. 1.2.3. Occorre allora un ulteriore meccanismo che consenta di verificare la corrispondenza fra una chiave pubblica e l'identità a cui la si associa (che questa sia una persona o altro).

In teoria con PKI si dovrebbe indicare in maniera assolutamente generica quell'insieme di procedure, applicazioni, entità che consente di gestire, creare, diffondere e mantenere l'associazione fra una una determinata chiave pubblica e l'identità a questa abbinata che ne potrà fare uso per i citati servizi di identificazione e autenticazione, in modo che chiunque possa essere confidente della validità di questa associazione, risolvendo il problema del *man-in-the-middle*.

Quasi sempre però piuttosto che a questa definizione generica, la sigla PKI fa riferimento all'implementazione di uno dei possibili modelli strutturali che cercano di dare una risposta a queste problematiche, ed anche noi nel seguito di questa sezione faremo riferimento a questa accezione, quello che si basa sull'idea di delegare la verifica della corrispondenza fra chiavi crittografiche ed identità ad una entità terza (il cosiddetto TTP, o *Trusted Third Party*) che si presuppone essere fidata senza verifica e che viene a svolgere un ruolo affine a quello di un notaio.

Questo comunque non è l'unico modello possibile, dato che presenta un importante punto critico nel dilemma costituito dal fatto che si ripone una fiducia incondizionata in chi svolge il ruolo di *Trusted Third Party*, che potrebbe non meritarla. Esiste pertanto un altro modello strutturale che è quello del cosiddetto *web of trust*, scelto da programmi come PGP e GPG, che tratteremo in sez. 2.2.

Inoltre benché l'idea di *Trusted Third Party* sia implementabile in modi diversi, la modalità principale in cui oggi viene realizzata è attraverso le cosiddette *Certification Authority*, (in breve CA), una serie di soggetti a cui è delegato, con una forma alquanto complessa ed tutt'altro che scevra da problemi di affidabilità e sicurezza, questo compito. Quello delle CA è il modello a cui fa riferimento il protocollo SSL (che tratteremo con maggiori dettagli in sez. 2.1.3) che è quello più utilizzato per fornire riservatezza e verifica dell'identità nei servizi di rete.

2.1.2 Chiavi, certificati e *Certification Authority*

Il componente di base su cui si costruisce una PKI basata sulle CA è il cosiddetto *certificato digitale* (in inglese *public key certificate* o *digital identity certificate*). Dal punto di vista realizzativo questo non è altro che un file che contiene la chiave pubblica di una coppia di chiavi asimmetriche, a cui vengono aggiunte una serie di dati il più importante dei quali è la firma digitale rilasciata dell'entità (la *Certification Authority*) che attesta l'autenticità delle informazioni (la chiave ed il resto) riportate nel certificato stesso.

Un utente potrà usare il suo certificato insieme alla *chiave*[3] ad esso associata per stabilire una comunicazione cifrata con la procedura di *Diffie-Hellman* (o altro meccanismo equivalente), potrà usarlo per effettuare firme digitali, e potrà usare la chiave per decifrare messaggi che siano stati cifrati con il suo certificato, esattamente come si fa con una qualunque coppia di chiavi asimmetriche.

[3] nella terminologia delle PKI si parla sempre di *certificato* e *chiave*, dal punto di vista crittografico il primo corrisponde alla chiave pubblica, la seconda alla chiave privata.

La differenza con la chiave pubblica di una ordinaria coppia di chiavi asimmetriche sta nel fatto chi riceve il certificato ottiene anche le informazioni aggiuntive, fra cui appunto la firma digitale del certificato stesso e detta firma copre non solo la chiave pubblica, ma anche i dati aggiuntivi presenti nel certificato, garantendone autenticità ed integrità nei confronti di attacchi di *man-in-the-middle*. Sulla base della verifica di quest'ultima si potrà dedurre l'autenticità, nel senso di corrispondenza all'identità proclamata, sia della chiave pubblica che delle informazioni presenti nel certificato.

L'idea di base di una PKI strutturata in questo modo è che, avendo una firma all'interno di un certificato, la si potrà usare per verificarne l'autenticità con il certificato di chi l'ha apposta. È sempre possibile infatti usare un certificato per firmarne un altro,[4] e questo è il meccanismo che consente di delegare il compito di garantire la effettiva correttezza dei dati presenti in un certificato all'*ente certificatore* (la *Certification Authority*) poi lo firmerà. In questo modo ci si deve preoccupare di ottenere in maniera fidata solo il certificato della CA, riducendo di molto il rischio di attacchi di *man-in-the-middle*.

Questo comporta anche un cambiamento nella modalità con cui si effettua l'autenticazione usando il certificato. Nel meccanismo classico, ad esempio quello usato da SSH, ci si limitava a verificare che chi presentava il certificato avesse la corrispondente chiave pubblica, dando per scontato che il certificato fosse "*autentico*". In questo caso basta controllare che il certificato sia valido con la firma in esso contenuta. Pertanto non è più necessario verificare direttamente le chiavi pubbliche di ogni corrispondente, e l'unico certificato della cui autenticità occorre preoccuparsi in prima persona è quello della *Certification Authority*.

Resta ovviamente il problema di come verificare la correttezza del certificato dell'ente certificatore rispetto ad attacchi *man-in-the-middle*. Questo semplicemente non è possibile, dato che a questo punto non esiste più un terzo a cui rivolgersi che possa garantire per lui, per cui si deve assumere il rischio e darlo per valido senza conferme. Per questo il caso più comune per i certificati usati dalle *Certification Authority* è che sono autofirmati. Questo significa che una CA attesta da se stessa la validità del proprio certificato, firmandolo con la chiave ad esso associato.

Questo problema di fiducia è meno teorico di quanto si possa pensare, infatti ci sono da sempre serie critiche relative alla scarsa qualità dei controlli posti in atto dalle varie *Certification Authority*, che in molti casi si sono dimostrati tutt'altro che adeguati; si tenga presente infatti che basta che una *Certification Authority* sia sovvertita o segua procedure poco sicure emettendo certificati senza verifiche sufficienti per compromettere la sicurezza di tutto il sistema, che è condizionata da quella del suo anello più debole.

Resta inoltre aperto un problema tutt'altro che banale relativo a questo modello; quello della distribuzione dei certificati e della verifica degli stessi. Quando il sistema è nato si era pensato di ricalcare il meccanismo degli elenchi telefonici in uso nelle telecomunicazioni, un meccanismo molto complesso associato allo standard X.509,[5] per cui sarebbe stato possibile creare una gerarchia nella distribuzione usando le informazioni contenute nei certificati stessi,

[4] questa affermazione in termini stretti è scorretta, in quanto non si usa un certificato per firmarne un altro, ma si usa la chiave ad esso associata, ed il certificato (che contiene una chiave pubblica) serve solo per fare la verifica della firma; per semplicità espositiva si parla spesso di certificato firmato con un altro certificato, dando per sottinteso che si sta facendo riferimento alla chiave associata quest'ultimo.

[5] da cui è tratto anche lo schema dei dati originale di LDAP, nato anche questo per mantenere rubriche, e poi trasformatosi in un servizio generico, e parte della nomenclatura resta la stessa, per un approfondimento si consulti il capitolo di [IS-LDAP].

con delle relazioni di dipendenza che facessero ottenere in maniera diretta i certificati di un attore finale attraverso una ricerca nella gerarchia dei fornitori, in cui a ciascun livello sarebbe stata presente una *Certification Authority* che certificava quella del livello successivo, con una singola *Certification Authority* radice da cui sarebbero state certificate tutte le altre.

Sigla	Nome	Significato
`C`	*Country*	Il paese, espresso con il codice a due lettere usato anche per la localizzazione e i domini di primo livello, ad esempio `C=IT`.
`ST`	*State*	Lo stato[6] (negli USA) o la provincia, ad esempio `ST=FI`.
`L`	*Locality*	La località, come la città o il comune, ad esempio `L=Firenze`.
`O`	*Organization*	Il nome dell'organizzazione o della società, ad esempio `O=Truelite Srl`.
`OU`	*Organizational Unit*	Il nome dell'unità operativa (il dipartimento o la sezione) della precedente organizzazione, ad esempio `OU=IT department`.
`CN`	*Common name*	Il nome dell'identità a cui si fa riferimento, che può essere a seconda dei casi un nominativo, un indirizzo di posta o un nome a dominio, ad esempio `CN=www.truelite.it`.

Tabella 2.1: Le informazioni di identificazione presenti nei certificati.

Questo approccio comportava una specifica scelta delle informazioni di identificazione inserite all'interno del certificato, che dovevano appunto seguire il modello gerarchico dello standard X.509, che prevede una suddivisione di stampo geografico, con indicazioni relative allo stato, alla località ed alla organizzazione, un po' come si organizzano gli elenchi del telefono. Per questo le informazioni previste erano quelle riportate nello schema illustrato in tab. 2.1. Ripercorrendo la gerarchia a partire dal livello più alto (il singolo stato) a quello più basso (una unità interna ad una organizzazione) si sarebbe potuto ottenere il certificato relativo al singolo appartenente alla stessa.

Però, in mancanza di una autorità centrale che stabilisse questa gerarchia, e visto anche lo scarso interesse delle varie organizzazioni che avrebbero dovuto usarla come utenti (in particolare di quelle private) a rendere pubblica la loro struttura organizzativa interna, la gerarchizzazione descritta è mai stata realizzata, ed invece sono nate invece una serie di *Certification Authority* totalmente indipendenti fra loro e senza nessuna relazione gerarchica, e non è mai stato realizzato nessun sistema di distribuzione diretta dei certificati. Tra l'altro l'avere impostato tutta l'architettura in forma gerarchica rende ora estremamente complesso realizzare relazioni di fiducia paritarie in cui due CA si attestano reciprocamente la loro validità.

Le informazioni di identificazione previste dallo standard X.509 sono però rimaste all'interno del formato dei certificati anche se ai nostri giorni buona parte di esse non risultano di nessuna utilità, o vengono usate per contenuti del tutto diversi da quelli indicati in tab. 2.1. Il solo dato che viene effettivamente usato resta quello del *Common Name* che in molti casi è l'unica informazione utilizzata per determinare l'identità a cui il certificato è associato, cosa che tra l'altro pone problemi tutt'altro che banali in caso di omonimia. Per i nomi a dominio infatti non ci sono problemi, essendo questi univoci, ma la cosa diventa assai complicata con i certificati personali ad uso dei programmi di posta.

[6] stato deriva dalla visione centrata sugli Stati Uniti, in cui lo stato fa riferimento ad uno degli stati dell'unione, e che altrove è stato associato a realtà territoriali considerate equivalenti, in Italia in genere si fa riferimento a una provincia.

Oltre alle informazioni di tab. 2.1, che sono indicate dal soggetto che richiede il certificato per identificarsi,[7] e che vengono inserite in quella che si chiama una *Certificate Signing Request* (o CSR),[8] all'interno di un certificato vengono riportati una serie di altri dati che vengono inseriti dalla *Certification Authority* al momento della firma. Le informazioni vengono raggruppate in diverse sezioni (che si sono riportate in tab. 2.2) ciascuna della quale è identificata da un nome ed il cui contenuto è stato definito da vari standard. Si noti come in tab. 2.2 non viene citata la firma digitale, questa infatti non fa parte dei dati ivi descritti, ma certifica la validità degli stessi.

Sezione	**Significato**
Version	Versione dello standard X.509 usato (oggi il valore in uso è sempre 3).
Serial Number	Numero seriale del certificato; ogni CA assegna un seriale a ciascun certificato che firma, usato per identificarlo ad esempio nelle liste di revoca (CRL).
Signature Algorithm	Algoritmo crittografico usato per la firma, identificato dal suo nome convenzionale.
Issuer	I dati identificativi dell'ente che firma il certificato, espressi in forma compatta con le sigle di tab. 2.1.
Validity	L'intervallo di validità del certificato, con la data di inizio e la data di scadenza.
Subject	I dati identificativi del soggetto del certificato, espressi sempre con le sigle di tab. 2.1.
Subject Public Key Info	La chiave pubblica del certificato, che comprende sia il valore della chiave stessa, che il tipo di chiave usata (ad esempio se RSA, DSA o di altro tipo).
X509v3 extensions	Una serie di estensioni che consentono di aggiungere informazioni aggiuntive previste dalla terza revisione dello standard X.509.

Tabella 2.2: Le sezioni in cui sono suddivisi i dati inseriti nei certificati.

Fra i dati di tab. 2.2 uno dei più importanti è quello relativo al periodo di validità del certificato della sezione *Validity*, in cui vengono inserite una data di inizio ed una data di scadenza, oltre la quale, pur restando corretta la firma, il certificato non verrà più considerato valido. La scadenza serve ad evitare la permanenza di certificati validi relativi ad identità magari non più esistenti, ma soprattutto a garantire a chi gestisce le CA un reddito continuativo per il rinnovo.

Le estensioni del protocollo (quelle della sezione *X509v3 extensions*) consentono inoltre di indicare altri dati significativi. Fra questi ad esempio è possibile indicare la tipologia di uso del certificato,[9] se si tratta di un certificato personale per la firma digitale, o di un certificato di un server o di quello di un client.[10] Un'altra informazione importante, sempre inserita all'interno

[7] anche se nella pratica in genere l'unica informazione che un richiedente può stabilire è quella del *Common Name*, le altre in genere vengono sovrascritte dalla CA quando firma il certificato.

[8] si tratta in genere di un file in cui, sempre in opportuno formato (in genere PEM), si inserisce la chiave pubblica e le suddette informazioni pubbliche, così che poi una *Certification Authority* possa firmarle.

[9] il dato si chiama *Netscape Cert Type*, e prevede tipologie indicate come *SSL Client*, *S/MIME*, *Object Signing*, *SSL Server*.

[10] ad esempio nel protocollo SSL alcuni client non riconoscono come valido il certificato di un server che contattano se questo non viene qualificato come *SSL Server* nel campo *Netscape Cert Type*.

delle estensioni, è quella di poter indicare delle ulteriori identità alternative, da aggiungersi ai dati indicati nel *Common Name* della sezione *Subject*.[11]

Come già detto, in assenza di un meccanismo centralizzato di distribuzione dei certificati validi in tempo reale da parte delle *Certification Authority* e del proliferare di queste ultime, uno dei punti critici del sistema è quello dell'aggiornamento dei dati in caso di compromissione di un certificato. Per quanto riguarda le *Certification Authority* in genere quello che accade è che una lista dei loro certificati viene distribuita direttamente da loro sul web o inclusa direttamente nei programmi (questo è ad esempio quello che avviene con i browser web), ed essendo questi in numero limitato ed in genere piuttosto controllati il problema veniva considerato non significativo,[12] e non è pertanto prevista nessuna procedura specifica per affrontarlo (questo significa che in sostanza i certificati radice non sono revocabili).

La compromissione dei certificati degli utenti è invece considerata eventualità assolutamente normale, ed è prevista una contromisura specifica nella forma delle cosiddette *Certificate Revocation List* o CRL. Il meccanismo prevede che in caso di compromissione del suo certificato un utente debba notificare l'avvenuto alla propria *Certification Authority* la quale provvederà ad inserire il numero seriale dello stesso all'interno di questa lista di certificati revocati. Un altro utente potrà così accorgersi, utilizzando detta lista, dell'avvenuta compromissione e non considerare più valido detto certificato. É compito delle *Certification Authority* compilare e pubblicare queste liste (che sono da loro firmate), e questo in genere avviene esattamente, come per i propri certificati, tramite sito web.

Il grandissimo problema di tutto ciò è che è facoltà dell'utente finale effettuare il controllo di una eventuale revoca, e che se decide di farlo è lui che deve curarsi di ottenere una CRL aggiornata dalla sua CA, verificare che il certificato esaminato non vi compaia e solo allora passare alla verifica della firma. La cosa comporta un notevole aggravio gestionale anche perché a differenza del certificato della CA, che si scarica in sostanza una volta sola e che molti programmi inseriscono direttamente al proprio interno, le CRL devono essere mantenute continuamente aggiornate, e questo per tutte le CA esistenti, che finiscono per l'avere meccanismi di distribuzione diversi e tutt'altro che coordinati.

La conseguenza è che il meccanismo di gestione della revoca dei certificati comporta una notevole complicazione della gestione totalmente a carico dell'utente finale dato che non è mai stato previsto un sistema automatico di applicazione delle CRL.[13] Il risultato è che nella stragrande maggioranza dei casi il meccanismo risulta troppo oneroso e complesso e le CRL vengono completamente ignorate.

Benché l'effettiva sicurezza del modello delle PKI basate sulle *Certification Authority* possa essere soggetta a parecchi dubbi, illustrati nelle pagine precedenti, esso resta dominante in particolare per la sua applicazione nel protocollo SSL che tratteremo in sez. 2.1.3. Inoltre questi

[11] il dato viene identificato come *X509v3 Subject Alternative Name*, e prevede indicazioni di nomi, indirizzi IP o nomi a dominio alternativi (ci ritorneremo in sez. 2.1.4) che per chi supporta questa estensione risultano validi come se fossero stati inseriti nel *Common Name*.

[12] salvo poi rivelarsi estremamente serio con la compromissione di una CA e conseguente emissione di certificati falsi validi intestati a domini di grande rilevanza, cosa che ha costretto gli sviluppatori dei browser più usati ad inserire delle patch ad hoc per escludere questi singoli certificati (dato che escludere il certificato della CA avrebbe reso invalidi tutti i certificati da essa firmati) mostrando quanto possa essere fragile il sistema.

[13] anche se in conseguenza dei problemi che ciò comporta ne sono stati proposti alcuni che dovrebbero effettuare il controllo in tempo reale, solo che a questo punto verrebbe da chiedersi perché non distribuire direttamente i certificati validi.

problemi emergono nella sua applicazione a livello globale, e non si pongono affatto qualora si voglia utilizzare una CA propria di cui si abbia un controllo diretto. Per questo motivo i concetti esposti in questo paragrafo torneranno utili in molte altre occasioni, dato che sono parecchi i programmi che fanno uso di chiavi e certificati per garantire riservatezza ed identificazione.

2.1.3 Il protocollo SSL/TLS e il programma `stunnel`

La principale applicazione del modello di PKI basato sulle *Certification Authority* descritte in sez. 2.1.2 è quella del protocollo SSL (oggi ribattezzato TLS) che si appoggia a questo tipo di infrastruttura per introdurre un maggiore livello di sicurezza nei protocolli di rete. Come accennato anche in sez. 1.1.3 infatti i protocolli di rete non forniscono nessuna forma di autenticazione né di riservatezza nella comunicazione, pertanto diventa possibile sia spacciarsi per altri che intercettare il traffico in maniera tutto sommato molto semplice.

Una delle possibili risposte a questo problema è stata quella di definire un protocollo specifico, originariamente chiamato SSL (sigla che sta per *Secure Socket Layer*), che si interponesse fra i protocolli del livello di trasporto e quelli di applicazione,[14] per introdurre meccanismi di autenticazione e verifica dell'identità del corrispondente e di cifratura del traffico.

Il protocollo SSL originale venne sviluppato da Netscape Corporation principalmente come risposta al problema di consentire un accesso sicuro ai siti web, nell'ottica di favorire il commercio elettronico. Il protocollo nacque come *strato* intermedio fra il TCP i vari possibili protocolli del livello applicativo; all'inizio era usato per HTTP, ma è utilizzabile in generale con qualunque protocollo che usi TCP.

Il concetto è che una applicazione, invece di scrivere direttamente su un socket TCP, utilizzi uno speciale "*socket sicuro*" gestito dal protocollo. Tutte le operazioni di apertura del socket con la verifica del corrispondente e tutte le successive operazioni di cifratura dei dati prima della loro trasmissione sulla rete attraverso un socket ordinario vengono gestite dal protocollo. In sostanza una applicazione deve solo utilizzare le funzioni di libreria per creare il *socket sicuro* e poi usarlo come se fosse un socket ordinario.

La prima versione ampiamente diffusa del protocollo SSL, nato come standard industriale basato sulle specifiche interne create da Netscape, è stata la 2.0, L'azienda pubblicò successivamente in forma di *Internet Draft* (ormai scaduto) un proposta per la successiva versione 3.0, che forniva alcune correzioni per prevenire alcuni attacchi, ampliava gli algoritmi di cifratura utilizzabili ed introduceva il supporto per le catene di certificati.

Questa bozza, pubblicata nel 1996, divenne la base su cui l'IETF sviluppò una standardizzazione non più proprietaria, cambiando però nome al protocollo in *Transport Layer Security* (in breve TLS). La prima versione è stata il TLS 1.0 (classificato anche come SSL 3.1) che è definito nell'RFC 2246. Il protocollo è stato soggetto ad ulteriori perfezionamenti,[15] e nel seguito ci riferiremo ad esso in modo generico sia con la sigla TLS che con la sigla SSL.

Come accennato SSL nasce per garantire un accesso sicuro (cifrato ed autenticato) ai siti web, così che si potesse essere sicuri di comunicare in maniera non intercettabile con l'interlocutore scelto e si potesse essere garantiti sull'identità di quest'ultimo. Il protocollo, astraendo l'inter-

[14] tratteremo in sez. 4.1.2 un diverso approccio al problema.

[15] nel 2006 è stata rilasciata la versione 1.1 (SSL 3.2) definita dall'RFC 4346, nel 2008 è stata rilasciata la versione 1.2 (SSL 3.3) definita dall'RFC 5246.

faccia di programmazione dei socket, è comunque generico e può essere usato con qualunque tipo di comunicazione tanto che oggi viene usato da una grande varietà di servizi di rete.

A grandi linee[16] il funzionamento del protocollo può essere diviso in tre passi principali. Il primo passo, che viene svolto in fase di creazione del socket sicuro, prevede che venga effettuata una negoziazione fra i due capi della connessione sugli algoritmi di crittografia, le eventuali compressioni dei dati e la versione di protocollo da usare, in modo da accordarsi su una scelta che sia utilizzabile da tutti e due. Il protocollo prevede che venga usata la versione più recente del protocollo supportata da entrambi, mentre si possono indicare delle preferenze per gli algoritmi di cifratura in fase di configurazione.

Il secondo passo, sempre in fase di creazione del socket sicuro, prevede lo scambio delle chiavi e la verifica dell'identità del corrispondente che viene stabilita proprio usando il modello di PKI basato sulle *Certification Authority* illustrato in sez. 2.1.2. Normalmente è il server che deve autenticarsi presso il client presentando un certificato valido in quanto firmato da una delle *Certification Authority* riconosciute dal client, ma se non interessa si può ignorare l'autenticazione per ottenere solo la cifratura del traffico, o può essere richiesta anche l'autenticazione del client, che a questo punto dovrà presentare anche lui un certificato al server che lo verificherà con le sue *Certification Authority*.

Dato che la compromissione di una *Certification Authority* è critica, con TLS 1.0 è stato introdotto il supporto per le *catene di certificati* che consentono l'uso dei cosiddetti *certificati intermedi*. In questo caso una CA può delegare la firma dei certificati finali degli utenti ad un *certificato intermedio*. Il meccanismo prevede che una CA possa firmare con il suo certificato, (detto in questo caso *radice*) un certificato intermedio, che questo ne possa firmare un altro e così via, creando appunto una *catena di certificati* la cui validità può essere verificata ripercorrendo all'indietro la catena fino ad arrivare alla firma del certificato della CA sul primo certificato della catena, in genere comunque per non complicare troppo le cose viene usato un solo certificato intermedio.

Il vantaggio dell'uso dei certificati intermedi è che questi possono essere sostituiti più facilmente in caso di compromissione in quanto vengono forniti direttamente dal server all'interno della negoziazione fatta dal protocollo SSL,[17] per cui non è necessario che vengano sostituiti i certificati delle CA in tutti i client ma basta sostituire i certificati intermedi sul server. In questo modo si può pensare di applicare una politica di sicurezza estremamente rigida sul certificato radice, usando sempre i certificati intermedi per apporre le firme, riducendo così il rischio di compromissione del certificato radice di una CA.

Una volta completato il secondo passo e verificata l'identità del corrispondente, o dei corrispondenti qualora si richieda la mutua autenticazione, l'ultimo passo prevede la determinazione di una chiave di sessione sfruttando la crittografia a chiave asimmetrica. Da questo punto in poi il socket sicuro sarà pronto per essere usato dalle applicazioni come un socket normale e tutto il traffico inviato su di esso verrà automaticamente cifrato e decifrato, e la sua integrità verrà garantita con la generazione degli opportuni codici di controllo.

[16] una trattazione dettagliata va al di là dello scopo di questo testo, per approfondire si può consultare `http://en.wikipedia.org/wiki/Transport_Layer_Security`.

[17] questo comporta che se il proprio certificato è firmato da un certificato intermedio nella configurazione di SSL occorrerà indicare opportunamente di fornire anche questo, a differenza di quanto avverrebbe per un certificato firmato direttamente da un certificato *radice*.

Originariamente il supporto del protocollo richiedeva l'uso diretto di una porta dedicata per il traffico cifrato con SSL; ad esempio per i siti web si usa la porta 443 e l'indicazione `https`, al posto della ordinaria porta 80 e di `http`. Con le versioni più recenti è possibile utilizzare le porte dell'ordinario traffico in chiaro utilizzando un meccanismo denominato *STARTLS*, in cui, se supportato dall'applicazione (in genere con una estensione del relativo protocollo applicativo) si fa partire la negoziazione dalla connessione sicura all'interno di una connessione in chiaro; questo tipo di applicazione è ad esempio comune con i server di posta elettronica che prevedono una apposita estensione del protocollo SMTP.

La parte in genere più complessa della gestione del protocollo SSL è quella relativa alla indicazione dei certificati e delle chiavi da usare per la verifica delle identità dei corrispondenti ed alle varie opzioni che consentono di controllare le modalità con cui viene effettuata la creazione dei socket sicuri. Dato che oggi nella gran parte dei casi il supporto per l'uso del protocollo viene gestito direttamente tramite l'uso di opportune librerie all'interno delle singole applicazioni,[18] ognuna di esse fornisce anche le proprie direttive di configurazione.

Non tratteremo quindi la configurazione di SSL, dato che questa varia da applicazione ad applicazione, ma prenderemo invece in esame il programma `stunnel` che, come dice il nome, consente di creare dei *tunnel* fra traffico in chiaro e traffico protetto da SSL e che consente di fare usare il protocollo anche da applicazioni che non lo supportano direttamente. Il programma infatti è in grado sia di accettare connessioni SSL, decifrare il traffico e reinoltrarlo in chiaro su un socket ordinario, che di accettare connessioni in chiaro, eseguire una connessione SSL e inviarvi sopra il traffico cifrato.[19]

Questo consente ad un client che supporta il protocollo SSL di connettersi ad un servizio che non lo supporta tramite `stunnel` (usato nella cosiddetta modalità server in cui accetta connessioni SSL e reinoltra il traffico in chiaro) o viceversa far collegare ad un server con SSL un client che non supporta il protocollo, usando `stunnel` per la connessione (in modalità client, in cui riceve connessioni in chiaro e le reinoltra su SSL). Infine mettendo in comunicazione fra loro due istanze, una in modalità server e l'altra in modalità client, si potranno connettere usando il protocollo anche applicazioni che non supportano SSL sia lato client che server.

Il comportamento del programma viene determinato dal file di configurazione passato come argomento, ma se non si indica niente di default viene usato `/etc/stunnel/stunnel.conf`.[20] Si può inoltre usare l'opzione `-fd` seguita da un numero per fargli leggere la configurazione dal corrispondente file descriptor, per il quale si usa esclusivamente 0, per la lettura dallo *standard input*. Buona parte delle distribuzioni provvedono inoltre il supporto per l'avvio automatico all'interno delle procedure di bootstrap.[21]

Il formato del file di configurazione è quello dei `.ini` di Windows contenenti assegnazioni di valori a direttive di configurazione; nella parte iniziale del file si impostano le caratteristiche generali di funzionamento del programma, mentre per ogni tunnel che si vuole creare (il comando

[18] esistono in realtà diverse implementazioni alternative del protocollo, come quella delle `libssl` del progetto OpenSSL, che ha alcuni problemi di compatibilità di licenza con la GPL, o quella delle `libtls` del progetto GNU, nata anche per risolvere detti problemi.

[19] si fa riferimento solo a traffico TCP, `stunnel` non supporta UDP, anche se con TLS sono state introdotte estensioni del protocollo (*Datagram Transport Layer Security* o DTLS) in grado di funzionare anche con UDP.

[20] si fa riferimento alla versione 4 del programma, ma dato che questa ha visto una completa revisione delle modalità di funzionamento è talvolta disponibile anche la versione 3 che ha una sintassi completamente diversa.

[21] ad esempio Debian avvia una istanza per ogni file terminante in `.conf` che si trova in `/etc/stunnel/`, preso come rispettivo file di configurazione (si deve però abilitare la cosa in `/etc/default/stunnel`).

supporta la gestione di più tunnel in contemporanea) si deve usare una apposita sezione identificata da un nome fra parentesi quadre che indichi il relativo servizio. È comunque possibile utilizzare le direttive specifiche per i singoli tunnel anche globalmente nel qual caso verranno applicate come default a tutti i tunnel creati.

Per una maggiore sicurezza si può far girare il programma in un *chroot*, la relativa directory si imposta con la direttiva `chroot`, nel qual caso si deve tener conto che tutti i pathname che si specificheranno in seguito saranno relativi a detta directory. Si può inoltre indicare con le direttive `setuid` e `setgid` per conto di quale utente e gruppo eseguire il programma in modo da fargli cedere i privilegi di amministratore.[22]

Benché sia possibile specificarli per ciascun tunnel, in genere si indicano in maniera generica anche i file relativi ai certificati necessari per l'uso di SSL. La principale direttiva usata a tale scopo è `cert`, che serve ad indicate il file contenente il certificato che `stunnel` deve presentare ai propri corrispondenti. La direttiva è necessaria solo in modalità server, dato che in modalità client, a meno che non sia richiesta la mutua autenticazione, non è necessario presentare nessun certificato.

Il programma supporta solo il formato PEM (torneremo con qualche dettaglio in più su questo in sez. 2.1.4) che ha il vantaggio di consentire l'unione di più certificati in formato PEM in un singolo file con la semplice concatenazione. Questo consente di inserire in unico file anche la chiave (che però deve essere posta in testa al certificato) ed eventuali certificati intermedi qualora il certificato necessitasse anche di questi; anche in quest'ultimo caso l'ordine conta e questi devono essere inseriti nel file nel relativo ordine gerarchico, con in fondo il certificato finale. È sempre possibile comunque utilizzare la direttiva `key` per specificare in sede separata il file della chiave, che contenendo informazioni sensibili deve essere adeguatamente protetto e non leggibile da estranei.

Qualora invece si usi il programma in modalità client e si voglia verificare il destinatario della propria connessione SSL occorrerà indicare quali CA si intendono utilizzare per la verifica. Per questo si può utilizzare la direttiva `CAfile` per indicare il file con il relativo certificato (o i certificati, se sono più di una). Si può inoltre indicare una CRL per eliminare i certificati non più validi con `CRLfile`.

La direttiva che permette di determinare il comportamento del programma (se cioè questo deve lavorare in modalità client o server) è `client`. Di default infatti, se la direttiva non viene specificata, `stunnel` si comporta come server, accettando connessioni in SSL e reinoltrandole in chiaro. Se si definisce la direttiva come `yes` si attiva invece la modalità client, e si può usare `stunnel` per reinoltrare su SSL connessioni in chiaro. Si può impostare esplicitamente la modalità server definendo la direttiva come `no`.

Una seconda direttiva che controlla il funzionamento del programma, significativa principalmente per la modalità client, è `verify` che imposta il livello di verifica richiesta in una connessione SSL, il default infatti, se non la si usa, è di non controllare. La direttiva richiede un valore numerico che indica un livello di verifica secondo quanto riportato in tab. 2.3. In genere si usa il livello 2 per accertarsi, in modalità client, di effettuare la connessione ad un server SSL corretto, mentre il livello 3 viene usato principalmente per connessioni punto punto in cui si richiede che il corrispondente presenti esattamente il certificato indicato dalla direttiva `cert`.

[22] ad esempio su Debian viene usato l'utente `stunnel4`, su RedHat/CentOS `stunnel`.

Si tenga presente comunque che le verifiche eseguite da `stunnel` quando si imposta un livello di verifica si basano esclusivamente sul fatto che la firma del certificato sia valida rispetto alle CA utilizzate e che questo non sia stato revocato (se si è indicata una CRL con le relative direttive). Non viene ad esempio controllata la corrispondenza di un eventuale nome a dominio contenuto nel certificato con quello quello usato per la effettuare la connessione.

Per ogni tunnel che si crea si devono poi utilizzare le direttive `accept` e `connect` per definire gli estremi del tunnel. La sintassi è identica per entrambi, e nella forma `indirizzo:porta`, dove sia l'indirizzo che la porta possono essere specificati in forma sia numerica che simbolica. Si può indicare anche soltanto una porta omettendo l'indirizzo.

Con `accept` si imposta la porta locale su cui si accettano connessioni, in questo caso se lo si vuole impostare l'indirizzo IP deve essere uno di quelli assegnati alla macchina; se non si specifica nulla verrà usato l'indirizzo generico. Il significato della direttiva dipende dalla modalità di funzionamento scelta. In modalità server questa è la porta su cui si accettano le connessioni SSL, mentre in modalità client si accettano le connessioni in chiaro.

Direttiva	**Significato**
`chroot`	indica la directory da usare per il *chroot*.
`setgid`	gruppo per conto del quale eseguire il programma.
`setuid`	utente per conto del quale eseguire il programma.
`debug`	attiva la registrazione delle informazioni di debug da specificare nella forma *facility.priority* usata dal *syslog*.
`compression`	algoritmo di compressione dei dati (prende i valori `zlib` o `rle`), di default è disattivata.
`pid`	indica il file su cui salvare il PID del processo, (relativo alla directory specificata con `chroot`).
`accept`	porta sulla quale si ricevono le connessioni.
`CApath`	directory con i certificati della CA, nel formato delle cosiddette *standard certificate directory* (vedi sez. 2.1.4).
`CAfile`	file con il certificato (o i certificati) delle CA.
`client`	specifica se il programma deve agire come client, cioè ascoltare per connessioni in chiaro e reinoltrare il traffico su una connessione SSL, il default è il contrario (si ascolta in SSL e si reinoltra in chiaro).
`connect`	porta verso la quale ci si connette per reinoltrare il traffico.
`CRLpath`	directory con i file delle CRL (indicizzati come per `CApath`).
`CRLfile`	file con le *Certificate Revocation List* dei certificati da invalidare.
`cert`	file con il certificato (ed eventualmente la relativa chiave e i necessari certificati intermedi).
`key`	file con la chiave del certificato (se su file separato).
`sslVersion`	versione del protocollo da utilizzare, prende i valori `all`, `SSLv2`, `SSLv3` e `TLSv1`.
`verify`	livello di verifica del corrispondente richiesto nella connessione SSL, i possibili valori sono: `1`, se esiste il certificato del corrispondente, deve essere valido; `2`, il certificato del corrispondere deve esistere ed essere valido; `3`, il certificato deve corrispondere a quello indicato localmente; se non si specifica nulla non viene eseguita nessuna verifica.

Tabella 2.3: Le direttive per il file di configurazione di `stunnel`.

Viceversa con `connect` si definisce la destinazione verso cui si reinoltrano le connessioni, ed

anche in questo caso il significato dipende dalla modalità di impiego di `stunnel`, in modalità server in genere si indica una porta locale verso cui si redirige il traffico in chiaro, anche avrebbe poco senso dirigere il traffico in chiaro verso un'altra macchina, rendendo vano l'uso di `stunnel` per evitare la trasmissione dei dati in chiaro, mentre in modalità client si indica l'indirizzo del server SSL a cui ci si rivolge.

Le altre principali direttive sono riportate in tab. 2.3; nella prima parte si sono riportate le direttive globali, mentre nella seconda parte quelle impostabili a livello di singolo servizio. Per una documentazione completa si rimanda al solito alla lettura della pagina di manuale del comando.

2.1.4 Gestione di chiavi, certificati e CA

Benché esistano diversi programmi che consentono di creare e gestire chiavi e certificati ed anche mantenere una *Certification Authority* in maniera completa, con applicazioni web come OpenCA (`www.openca.org`) o ad interfaccia grafica come `tinyca`, ci concentreremo sulla suite di programmi `openssl` fornita dal progetto omonimo (`www.openssl.org`) e presente nella maggior parte delle distribuzioni. Detta suite fornisce un insieme di programmi per operare su chiavi e certificati a riga di comando, usati da buona parte delle applicazioni per la gestione delle CA.

In realtà il comando `openssl` non è altro che un wrapper che consente di invocare una serie di sottocomandi che consentono di effettuare tutte le più varie operazioni relative alla gestione di una infrastruttura di chiavi, certificati e *Certification Authority*; esso infatti viene invocato sempre nella forma:

```
openssl sottocomando [opzioni e argomenti]
```

dove con `sottocomando` si indica quale dei programmi che fanno parte della suite si vuole utilizzare; un breve elenco dei più significativi è riportato in tab. 2.4.[23]

Sottocomando	Significato
`ca`	gestione di una *Certification Authority*.
`crl`	gestione delle liste di revoca (CRL).
`dsa`	gestione delle chiavi DSA (soprasseduto da `pkey`).
`rsa`	gestione delle chiavi RSA (soprasseduto da `pkey`).
`gendsa`	creazione delle chiavi DSA (soprasseduto da `genpkey`).
`genpkey`	creazione di chiavi pubbliche (generico, sostituisce i due comandi `genrsa` e `gendsa`).
`genrsa`	creazione delle chiavi RSA (soprasseduto da `genpkey`).
`pkey`	gestione delle chiavi pubbliche (generico, sostituisce i due comandi `dsa` e `rsa`).
`req`	gestione delle richieste di certificati (*Certificate Signing Request*, o CSR) X.509.
`x509`	gestione dei certificati X.509.
`verify`	verifica dei certificati X.509.
`s_client`	client per il protocollo SSL, per uso diagnostico.
`s_server`	server SSL minimale per uso diagnostico.

Tabella 2.4: I principali sottocomandi di `openssl`.

[23] trattare tutte le funzionalità di `openssl` va al di là dello scopo di questo testo, gli interessati possono consultare la documentazione del progetto e le pagine di manuale, che si accedono usando solo il nome del sottocomando.

Ciascun sottocomando di `openssl` prevede delle opzioni e degli argomenti specifici, ma esistono una serie di opzioni generiche che restano le stesse per tutti i sottocomandi. In particolare si può indicare il file da cui leggere i dati in ingresso con `-in` (il default è usare lo *standard input*) e quello su cui scrivere i dati in uscita con `-out` (il default è usare lo *standard output*). Si può richiedere una uscita testuale con `-text` e la rimozione dall'uscita del contenuto delle versioni codificate con `-noout`.

Di particolare rilevanza sono poi le due opzioni `-passin` e `-passout` (e in certi casi anche semplicemente `-pass`), che vengono usate dai vari sottocomandi che richiedono l'immissione di una password, da utilizzare rispettivamente in ingresso per decifrare, e in uscita per cifrare.

Entrambe le opzioni prendono un parametro che indica la modalità in cui si fornisce la password, secondo la sintassi generica illustrata in tab. 2.5. Si tenga presente che se una password è necessaria e non si è indicato nulla, questa verrà richiesta esplicitamente sul terminale, e che in questo caso una semplice redirezione dello *standard input* non funzionerà, in quanto viene controllato che i dati ricevuti in ingresso siano immessi proprio da un terminale.

Parametro	**Significato**
`pass:password`	la password viene scritta direttamente sulla riga di comando (insicuro in quanto osservabile con `ps`).
`env:var`	la password viene letta dalla variabile di ambiente indicata (potenzialmente insicuro perché osservabile con `ps` con alcuni Unix).
`file:pathname`	la password viene letta dalla prima riga del file indicato.
`fd:number`	la password viene letta dal file descriptor indicato dal numero passato come parametro.
`stdin`	la password viene letta dallo *standard input*.

Tabella 2.5: Sintassi dei parametri con cui indicare le password nelle opzioni `-passin` e `-passout` di `openssl`.

Come accennato `openssl` supporta diversi formati per la gestione di file e certificati, fra questi i due principali sono il formato binario DER (*Distinguished Encoding Rules*), ed il formato testuale PEM (*Privacy Enhanced Mail*) che è un DER trasformato in ASCII puro con la codifica *base64* e l'aggiunta di righe di separazione in testa ed in coda alle informazioni.

Benché sia meno efficiente in termini di spazio rispetto al DER, il PEM è di gran lunga il formato più comune, principalmente per la maggiore maneggevolezza. Si possono infatti copiare facilmente i dati con un taglia e incolla da un qualunque editor, ed unire più certificati ed anche le chiavi in un solo file, semplicemente accodandoli uno all'altro. Per questo motivo è il formato usato di default da tutti i comandi della suite `openssl`. Un esempio di un certificato in formato PEM è il seguente:

cert_sample.pem

```
-----BEGIN CERTIFICATE-----
MIIHBjCCBO6gAwIBAgIBKzANBgkqhkiG9w0BAQUFADCBoDELMAkGA1UEBhMCSVQx
CzAJBgNVBAgTAkZJMRAwDgYDVQQHEwdGaXJlbnplMRUwEwYDVQQKEwxUcnVlbGl0
ZSBTcmwxIDAeBgNVBAsTF0NlcnRpZmljYXRpb24gQXV0aG9yaXR5MRgwFgYDVQQD
...
Eo32qyn9RP66aBV3EIFduasYGAk5M2AX4yIdhv4HPDW1XZMRdTJRsNnue5rSV0ER
AWu2pBug3i5sc33R1eymWuh5wfRNPzS2XqUExXKgo/oPEZF85QkPsK/vjZcXUriV
yVsE5GgC6Nos4dkbRo6RPBAsDk0vmWdIRuo=
-----END CERTIFICATE-----
```

Ovviamente l'esame dei dati di un certificato in questa forma non è proprio immediato, per questo uno dei sottocomandi di `openssl` più utilizzati è `x509`, che permette di visualizzarne le informazioni in forma esplicita, oltre a fornire operazioni di gestione e di conversione in formati diversi. Un esempio di invocazione per la visualizzazione dei dati è il seguente:

```
$ openssl x509 -text -noout < www.fi.trl-cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            93:b0:c1:d4:92:f8:9b:c0
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=FI, O=Truelite, OU=IT, CN=CA/emailAddress=info@truelite.it
        Validity
            Not Before: Jun  1 09:26:09 2011 GMT
            Not After : May 31 09:26:09 2012 GMT
        Subject: C=IT, ST=FI, L=Firenze, O=Truelite, OU=IT,CN=www.trl.tl/emailAddr
ess=info@trl.tl
        Subject Public Key Info:
            ...
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                B1:2B:49:96:BE:2D:A8:AE:13:A4:3D:05:9A:C0:DF:E8:AB:57:0C:FB
            X509v3 Authority Key Identifier:
                keyid:2D:13:7D:AB:78:6B:BC:E8:57:9F:4F:51:78:97:8F:27:E8:78:2D:5B

    Signature Algorithm: sha1WithRSAEncryption
        ...
```

e si noti come vengano riportati i contenuti delle varie sezioni in cui sono suddivise le informazioni, il cui significato è stato illustrato in tab. 2.2.

Il sottocomando supporta una serie di opzioni per visualizzare soltanto i dati delle singole sezioni o altre informazioni; fra queste vale la pena citare l'opzione `-hash` che consente di ottenere l'hash delle informazioni della sezione *Subject* del certificato, che deve essere usato per i nomi dei file che si mettono nelle cosiddette *standard certificate directory*.

Le librerie che implementano il protocollo SSL infatti consentono di indicare un elenco di certificati specificando semplicemente una directory che li contenga come singoli file. In tal caso però i file però vengono cercati, come forma di indicizzazione, con un nome nella forma `XXXXXXXX.0` dove `XXXXXXXX` è appunto questo hash. Una directory che contiene file di certificati con questi nomi viene appunto chiamata *standard certificate directory*.[24] Questo significa che se si vuole usare una di queste directory non è sufficiente copiare all'interno i file dei certificati, ma occorre usare esplicitamente questo tipo di nomi; in genere quello che si fa è creare dei link simbolici, ottenendo il nome con l'opzione `-hash` del comando `x509`.

In genere la creazione di un certificato è un procedimento che richiede diversi passi, l'ultimo dei quali è quello della firma da parte di una *Certification Authority*. Su quest'ultimo, che si

[24] come esempio si può vedere il contenuto si `/etc/ssl/certs` dove in genere vengono installati i certificati delle *Certification Authority* commerciali più note.

può effettuare direttamente qualora si gestisca una CA in proprio, torneremo più avanti, ma nel caso si debba ottenere un certificato da una CA terza occorre seguire una procedura ben precisa fin dall'inizio.

A differenza di quanto avveniva con SSH infatti in questo caso non esiste, essendoci in ballo un eventuale ente terzo, un comando singolo per creare chiave e certificato, ma occorre eseguire diverse operazioni. Volendo fare le cose un passo alla volta, il primo da compiere è quello di creare una chiave. Come illustrato in tab. 2.4 esistono vari sottocomandi deputati alla generazione di una chiave, per semplicità prenderemo in considerazione solo `genrsa` che consente di generare chiavi RSA.

Questo è il comando più semplice, gli altri sono comunque più o meno equivalenti essendo i concetti gli stessi, anche se per alcuni formati può essere necessario generare dei dati intermedi, cosa che comunque è documentata nelle rispettive pagine di manuale. Un esempio di uso del comando è:

```
$ openssl genrsa
Generating RSA private key, 512 bit long modulus
.++++++++++++
....++++++++++++
e is 65537 (0x10001)
-----BEGIN RSA PRIVATE KEY-----
MIIBPAIBAAJBALWFfAmGzuq08FgioYn3v0qhPGeZtpr+GPLa1j30/tTeSAZY65C3
GQEeoZYaanNFbLxU0NrnwkeMlgmXbgpSE0UCAwEAAQJBAK0lS6CGr36f4xYeynmh
ER2t2MF1lQIY2FFmvq9Qr3b5iN/MMMElrGOOR4BMMaO/SFsAtyYH5UqdzArBNghm
Fs0CIQDcTl89kmA9DnDGUNjRbfnum1n1FoynwKMOfmZPUezBfwIhANLucMXCp6RX
1bGjC2k203amYQtIuBUn+mci9QAABgU7AiEAtDrNgcKxRuH8ctGLP3JYSNMA+hez
hD19aAsSSgv8w6MCIHqAvMI+MPrDgHpwhPoQO3QXYRkanL68vzMTHFNpsBVtAiEA
pDG2dCvmn/GWR0W0ycLAu5shCv9XOs+mhCmgSXLwykw=
-----END RSA PRIVATE KEY-----
```

Il comando stampa sullo *standard input* i suoi risultati in formato PEM, ma si può far salvare direttamente soltanto le informazioni su un file con la citata opzione `-out`, lo stesso risultato si ottiene comunque redirigendo l'uscita su un file. Il comando supporta l'uso di un argomento per specificare la dimensione in bit della chiave (il default, come mostrato, è di 512 bit).

La chiave viene generata senza nessuna protezione, ma si può richiedere la cifratura della stessa con una fra le opzioni `-des`, `-des3` e `-idea` che utilizzano i corrispondenti algoritmi di cifratura, (rispettivamente DES, 3DES e IDEA, vedi sez. 1.2.2), nel qual caso verrà chiesta una password, che si potrà impostare con `-passout` con uno qualunque dei metodi indicati in tab. 2.5 o immettere direttamente sul terminale.

Se viceversa si vuole decifrare una chiave RSA si dovrà utilizzare il sottocomando `rsa`, che è anche quello che si usa per la gestione di questo tipo di chiavi. In tal basterà usarlo per fargli leggere il file con la chiave cifrata, ed eventualmente redirigere l'uscita su un altro file. Al solito verrà richiesta la password sul terminale o la si potrà indicare con `-passin`. Usando le stesse opzioni indicate per `genrsa` si potrà invece cifrare una chiave in chiaro o ottenere una cifratura con un diverso algoritmo (in quel caso si dovrà anche specificare una password in uscita). Per il resto la sintassi è analoga a quella di `x509` e lo si può utilizzare allo stesso modo per far generare una stampa in formato testuale dei dati associati alla chiave, anche se in questo caso, trattandosi soltanto di dati di natura crittografica, il contenuto esplicito è di scarso interesse.

Nella creazione di un certificato il sottocomando più importante è però `req`, che è quello che consente di creare una *richiesta di certificato* (una *Certificate Signing Request* o CSR), cioè il

file che le *Certification Authority* firmano per la creazione di un certificato. Una CA infatti non ha nessuna necessità di avere a disposizione la chiave associata ad un certificato, e per questo tutto quello che è necessario inviare è una richiesta che contenga solo la chiave pubblica ed i propri dati informativi (quelli di tab. 2.1).

La creazione di una richiesta si effettua usando il sottocomando con l'opzione `-new`; se usata da sola questa comporta anche la creazione di default di una chiave RSA a 1024 bit per la quale viene richiesta la password, altrimenti si può indicare l'uso di una chiave precedentemente creata con `genrsa` usando l'opzione `-key`. Questo ad esempio è utile in quanto è normale effettuare la creazione di una CSR con dati diversi in caso di rinnovo del certificato, pur mantenendo la stessa chiave, così da non dover sostituire anche questa. Pertanto se si è salvata la chiave precedente sul file `key.pem` si potrà creare una richiesta di certificato con:

```
$ openssl req -new -key key.pem
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) [AU]:IT
State or Province Name (full name) [Some-State]:FI
Locality Name (eg, city) []:Firenze
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Truelite Srl
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:www.truelite.it
Email Address []:info@truelite.it

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
-----BEGIN CERTIFICATE REQUEST-----
MIIBRjCB8QIBADCBizELMAkGA1UEBhMCSVQxCzAJBgNVBAgMAkZJMRAwDgYDVQQH
DAdGaXJlbnplMRUwEwYDVQQKDAxUcnVlbGl0ZSBTcmwxCzAJBgNVBAsMAklUMRgw
FgYDVQQDDA93d3cudHJ1ZWxpdGUuaXQxHzAdBgkqhkiG9w0BCQEWEGluZm9AdHJ1
ZWxpdGUuaXQwXDANBgkqhkiG9w0BAQEFAANLADBIAkEAnITR9WvfIToVy8ER6T2l
HEEJcxeVeIdpytOTXs1O/rGBoW9aw8d0GPPcKVq6aI+lqIGS4L3JbdSgs3lc46jj
7wIDAQABoAAwDQYJKoZIhvcNAQEFBQADQQBqfHAIOxY4wBXIy5DBZjY2oA07MQpP
oqMpGEb9as71sJNVP2UE/mfrnv0Nv5kn9jgYjmD7apZBF7f+/djn0yGb
-----END CERTIFICATE REQUEST-----
```

Si noti come il comando richieda l'immissione di una serie di informazioni da terminale (quelle di tab. 2.1 più alcuni dati aggiuntivi) evidenziati in grassetto nell'esempio, e poi stampi la richiesta di certificato ottenuta in formato PEM. Nel fare le richieste inoltre il comando suggerisce per alcuni campi dei valori di default che possono essere accettati premendo invio, o si può specificare il carattere "`.`" per lasciare il campo vuoto. Questi default provengono dal file `/etc/ssl/openssl.cnf`, che come vedremo più avanti serve da file di configurazione per una serie di impostazioni relative a tutti vari sottoprogrammi di `openssl`, e possono essere opportunamente preimpostati per velocizzare la creazione.

Inoltre il sottocomando `req` consente anche di esaminare i dati contenuti in una richiesta in formato PEM (o DER) in forma esplicita testuale, ad esempio per verificare la correttezza del

Common Name e dei dati della sezione *Subject*, e lo si può utilizzare in tal senso esattamente con le stesse opzioni già viste in precedenza per `x509` e `rsa`. Ad esempio con il precedente certificato otterremo:

```
$ openssl req -text -noout < csr.pem
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=IT, ST=FI, L=Firenze, O=Truelite Srl, OU=IT,CN=www.truelite.it/
emailAddress=info@truelite.it
        Subject Public Key Info:
            ...
    Signature Algorithm: sha1WithRSAEncryption
        ...
```

Una volta creata una CSR il passo successivo è quello di farla firmare da una *Certification Authority* per ottenere un certificato. Se si vuole usare il certificato per farsi identificare da terzi occorrerà rivolgersi ad una delle varie CA commerciali riconosciute come valide da questi ultimi,[25] inviandogli la richiesta e riottenendo indietro il certificato firmato; cosa che normalmente implica anche un pagamento ed una qualche forma di verifica dell'identità.

Quest'ultima è quasi sempre estremamente blanda, e viene effettuata nei termini della verifica della risposta ad una email o con un controllo superficiale della proprietà del dominio, a meno di che non sia stato richiesto un cosiddetto *Extended Validation Certificate* per l'emissione del quale sono richiesti dei controlli leggermente più approfonditi.

Questa è una estensione introdotta di recente, che cerca di rispondere agli evidenti problemi dimostrati dalle CA nel fare il loro lavoro (quello di verifica) che con le procedure ordinarie si è dimostrato inadeguato. Chi emette questi certificati (caratterizzati da specifici valori del campo *Certificate Policies* nelle estensioni X509.3) dichiara di aver seguito una procedura più stringente nella verifica della identità del richiedente e marca il certificato come rispondente a questa politica. Il risultato è che i browser più recenti classificano questo tipo di certificati come più affidabili, (in genere colorando di verde qualche elemento grafico) e che le CA chiedono un prezzo molto più alto per la sua emissione.

Dato che, come accennato in sez. 2.1.2, quando si chiede la firma ad una CA commerciale in genere l'unica informazione presente nella CSR che rimane disponibile nel certificato che si riottiene indietro è il *Common Name* della sezione *Subject*, si deve avere cura di specificarlo in maniera corretta. In genere, essendo questo tipo di applicazione mirato principalmente ai siti web, è essenziale che il *Common Name* corrisponda al nome a dominio con cui il sito viene raggiunto (quello completo che si mette nella URL di richiesta) dato che è su questa corrispondenza che viene eseguito il controllo da parte dei browser.

Questo comporta in genere una forte limitazione riguardo l'uso del certificato, in quanto esso può essere utilizzato solo per un sito;[26] se un browser si rivolgesse allo stesso sito con un nome diverso si avrebbe un errore, o meglio uno di quegli avvertimenti allarmistici riguardo la presunta

[25] in genere, dato che l'uso principale dei certificati è per garantire siti web, chi stabilisce se una CA è valida sono i fornitori dei browser, che includono all'interno dei propri programmi i certificati di quelle ritenute tali.

[26] si tenga inoltre presente che in genere non è possibile usare certificati diversi per diversi *Virtual Host* se questi sono riferiti allo stesso indirizzo IP (cioè *host based*), a meno di non utilizzare versioni molto di browser e server web che supportano una specifica estensione del protocollo SSL; per maggiori dettagli si consulti la sez. 1.4.5 di [WebServ].

inaffidabilità e pericolosità del sito stesso, che può essere rimosso solo accettando esplicitamente il certificato.

Esiste comunque la possibilità, fintanto che si resta all'interno di uno stesso dominio, di usare un cosiddetto certificato *wildcard*, valido per tutti i nomi all'interno di quel dominio. In sostanza si tratta in questo caso di specificare il *Common Name* nella forma:

```
*.dominio.it
```

e questo sarà considerato dai browser come valido per qualunque nome all'interno di quel dominio. Si tenga presente però che questo vale solo per nomi nella forma `xxx.dominio.it`, con `xxx` nome qualunque, mentre in generale (la cosa varia a seconda del browser) non saranno validi né ulteriori sotto-domini (cioè nomi nella forma `yyy.xxx.dominio.it`), né il semplice `dominio.it`.

Se non interessa che un certificato sia riconosciuto dai browser di terzi, o se interessa solo cifrare le connessioni, o se i servizi sono ad uso interno o non riguardano il web, si può decidere di realizzare in proprio una CA con la quale firmare i propri certificati. Come accennato questo può facilitare la distribuzione delle credenziali, in quanto non è più necessario verificare una manualmente una ad una tutte le chiavi pubbliche dei corrispondenti, ma basterà distribuire il certificato della CA per garantire l'autenticità dei certificati utilizzati dai vari servizi.

Dato che in questo caso la CA viene ad assumere un ruolo centrale nella verifica di integrità della propria infrastruttura, è in generale buona norma creare i certificati e le chiavi su una macchina dedicata, non connessa alla rete e tenuta in luogo sicuro. L'accesso alle chiavi della CA permette infatti di creare certificati in maniera arbitraria e compromettere tutte le comunicazioni fra i servizi che la usano spacciandosi per altri. Non disponendo di una macchina dedicata si può eseguire l'operazione su un portatile isolato dalla rete, e salvare tutti i file su un supporto temporaneo (magari cifrato). I veri paranoici possono anche stampare per ulteriore precauzione le chiavi (per riporle in luogo sufficientemente sicuro), e poi cancellare il tutto, ovviamente usando un programma di cancellazione sicura come `wipe` o `shred`.

Benché sia disponibile un apposito sottocomando `ca` di `openssl` il suo utilizzo diretto per la creazione totalmente manuale di una *Certification Authority* risulta piuttosto macchinoso, dovendo ad esempio salvare i certificati emessi, tracciare i numeri seriali, e mantenere manualmente vari dati. Lo stesso progetto OpenSSL considera detto comando "*stravagante e talvolta ostile*" (citazione dalla pagina di manuale). Per questo motivo insieme ai singoli sottocomandi di `openssl` il progetto OpenSSL mette a disposizione alcuni script che consentono di semplificare l'operazione di gestione di una CA, che sono quelli che prenderemo in considerazione.

Gli script distribuiti dal progetto sono due, `CA.pl` è la versione in Perl mentre `CA.sh` è uno script di shell,[27] ed il loro uso permette di semplificare la creazione della *Certification Authority*, facendo da interfaccia ad `openssl` per la gestione delle varie operazioni da eseguire per creare la propria CA personale. Entrambi gli script funzionano allo stesso modo e prendono gli stessi parametri, per semplicità faremo riferimento allo script di shell.

Entrambi gli script sono piuttosto grezzi, si usano invocandoli su un terminale, e salvano tutti i dati in una directory `demoCA` creata nella directory da cui vengono lanciati, il nome comunque può essere cambiato con una delle variabili di configurazione di `openssl.cnf` che tratteremo più avanti. In realtà gli script non fanno altro che chiamare nella opportuna sequenza i vari sottocomandi di `openssl`, e salvare i file in quella directory e nella directory corrente adottando una convenzione sui nomi che gli consente di riutilizzarli in successive invocazioni.

[27] con il pacchetto Debian di `openssl` questi vengono installati sotto `/usr/lib/ssl/misc/`.

Entrambi gli script fanno riferimento ad alcune variabili di ambiente che usano per impostare dei default. Se queste non sono presenti i valori di dette variabili vengono preimpostati all'interno degli script stessi.[28] In particolare la variabile `SSLEAY_CONFIG` serve per indicare un file di configurazione alternativo rispetto `openssl.cnf`, un possibile esempio del suo uso è impostare:

```
export SSLEAY\_CONFIG="-config ./openssl.cnf"
```

All'inizio di entrambi gli script sono poste le definizioni di una serie di ulteriori variabili interne che impostano dei default per nomi di file e comandi ed altri parametri interni che possono essere cambiati solo modificando gli script,[29] fra queste è particolarmente rilevante `CADAYS` che imposta la durata del certificato della CA, il suo valore di default infatti è di tre anni, che può risultare troppo breve.

Il primo passo per la creazione di una propria *Certification Authority* personale è quello di creare l'infrastruttura che conterrà i vari dati, fra cui il certificato della CA e la chiave corrispondente con cui verranno poi firmati i certificati rilasciati dalla stessa. Questo si fa una volta per tutte invocando lo script con l'opzione `-newca` che eseguirà in sequenza le varie operazioni necessarie. Come primo passo verrà creata la chiave privata, si otterrà pertanto un'uscita del tipo:

```
$ /usr/lib/ssl/misc/CA.sh -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Generating a 1024 bit RSA private key
..................++++++
....++++++
writing new private key to './demoCA/private/./cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

dove prima viene richiesta l'immissione di un eventuale file da cui importare dati già pronti (deve essere un file in formato PEM contenente chiave e certificato). Se come nel nostro caso si vuole fare una creazione da zero si dovrà premere invio per generare la chiave privata della CA e poi sarà richiesta una password con cui proteggerla che va inserita due volte per conferma.

Come secondo passo lo script crea internamente una CSR e richiede l'immissione delle informazioni previste dallo standard X.509 per il certificato della CA come per il precedente esempio di richiesta, per cui l'uscita a video del comando prosegue con qualcosa del tipo:

```
-----
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.
-----
```

[28] fra queste può trarre in inganno la variabile `DAYS`, che dovrebbe specificare di quanti giorni deve essere la validità di un certificato, in realtà questo valore viene impostato dentro `openssl.cnf` e nelle versioni recenti di `openssl` qualunque cosa si scriva nello script viene ignorato.

[29] fra queste c'è anche `DAYS`, ma come detto viene comunque ignorata, la durata deve essere impostata con `openssl.cnf`.

```
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:FI
Locality Name (eg, city) []:Firenze
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Truelite S.r.L.
Organizational Unit Name (eg, section) []:CA
Common Name (eg, YOUR name) []:truelite.it
Email Address []:info@truelite.it

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

dove si devono immettere i dati che verranno inseriti nel certificato della nostra CA esattamente come si è fatto con il sottocomando `req`.

Infine nell'ultimo passo verrà richiesta la password della chiave privata impostata all'inizio per firmare la richiesta appena generata ed ottenere il certificato della CA. Di questo poi verranno stampate le informazioni più rilevanti, per cui la parte finale dell'uscita del comando sarà qualcosa del tipo:

```
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/./cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            8d:34:8f:65:45:ad:86:80
        Validity
...
```

Come accennato tutti i file e le directory necessarie per la gestione della *Certification Authority* verranno creati dallo script in una sottodirectory `demoCA` a partire da quella in cui lo si è eseguito: in particolare la chiave della CA verrà posta in `demoCA/private/cakey.pem` ed il certificato con la chiave pubblica in `demoCA/cacert.pem`; si potranno poi controllare i dati relativi a quest'ultimo con il comando:

```
$ openssl x509 -text -noout < demoCA/cacert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            8d:34:8f:65:45:ad:86:80
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=FI, O=Truelite S.r.L., OU=CA, CN=truelite.it/emailAddre
ss=info@truelite.it
        Validity
            Not Before: Jun 14 13:24:57 2011 GMT
            Not After : Jun 13 13:24:57 2014 GMT
        Subject: C=IT, ST=FI, O=Truelite S.r.L., OU=CA, CN=truelite.it/emailAddr
ess=info@truelite.it
        Subject Public Key Info:
            ...
        X509v3 extensions:
            ...
            X509v3 Basic Constraints:
```

```
                CA:TRUE
    ...
```

e si noti come *Issuer* e *Subject* in questo caso siano identici.

Oltre a questi file la directory `demoCA` conterrà pure tutti gli altri dati necessari alla gestione della CA, in particolare sono rilevanti il file `demoCA/index.txt` che contiene la lista dei certificati rilasciati con numero seriale e contenuto del campo *Subject* e `demoCA/serial` che contiene il numero seriale che verrà usato per il prossimo certificato che verrà firmato.

Una volta inizializzata la propria *Certification Authority* si potrà passare alla creazione dei certificati. Per questo occorrerà anzitutto creare una CSR, che si potrà generare sia come descritto in precedenza che lanciando lo script con l'opzione `-newreq`, in questo caso verranno richieste una password per la chiave e le solite informazioni per la richiesta, che verranno salvate rispettivamente nei file `newkey.pem` e `newreq.pem` nella directory corrente.

L'ultimo passo per ottenere un certificato è quello di firmare la richiesta con la propria *Certification Authority*, per questo si deve invocare lo script con l'opzione `-sign`, restando sempre nella directory in cui si era creata la CA e la richiesta, lo script infatti assume di trovare la richiesta di certificato nel file `newreq.pem` della directory corrente ed i file della CA nella sottodirectory `demoCA`. Così facendo si otterrà un risultato del tipo:

```
$ /usr/lib/ssl/misc/CA.sh -sign
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            8d:34:8f:65:45:ad:86:81
        Validity
            Not Before: Jun 14 15:40:39 2011 GMT
            Not After : Jun 13 15:40:39 2012 GMT
        Subject:
            countryName               = IT
            stateOrProvinceName       = FI
            localityName              = Firenze
            organizationName          = Simone Piccardi
            organizationalUnitName    = gnulinux
            commonName                = piccardi.gnulinux.it
            emailAddress              = piccardi@gnulinux.it
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                83:F5:29:87:A0:F0:3D:6F:65:BC:B7:73:1E:6C:B0:E0:5E:40:D3:3E
            X509v3 Authority Key Identifier:
                keyid:FF:5D:CC:D7:34:88:C0:FE:9D:DE:D1:E3:F1:C0:78:34:82:C7:62:4C

Certificate is to be certified until Jun 13 15:40:39 2012 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
```

```
Write out database with 1 new entries
...
Signed certificate is in newcert.pem
```

il comando chiederà la password della CA per firmare il certificato, e dopo averne ricapitolato i dati chiederà di dare conferma riguardo la firma e la memorizzazione del nuovo certificato che sarà salvato nel file `newcert.pem`. Oltre a questo lo script genererà le informazioni (come i file `index.txt` e `serial`) necessarie alla CA per tenere traccia dei vari certificati firmati, e questo saranno copiati in `demoCA/newcert`, dove sono mantenuti tutti i certificati firmati in altrettanti file nominati per numero seriale. Qualora si voglia creare un altro certificato basterà ripetere l'operazione mettendo una nuova richiesta in `newreq.pem` avendo cura di spostare un eventuale precedente `newcert.pem` che verrebbe sovrascritto.

Eseguendo la firma in questo modo si utilizzeranno i valori di default sia per la durata del certificato che per i dati aggiuntivi previsti per la sezione delle *X509v3 extensions*. Come accennato questi sono definiti all'interno di `/etc/ssl/openssl.cnf`, che fa da file di configurazione per i vari sottocomandi di `openssl`. La documentazione al riguardo è scarsa e inorganica, e divisa nelle pagine di manuale dei vari comandi, ma la versione del file fornita con il pacchetto di OpenSSL è ampiamente commentata e sufficientemente autoesplicativa; essendo l'argomento molto vasto, ne tratteremo solo le parti più rilevanti.

Il file ha il formato dei file `.ini` di Windows, con assegnazioni di valori a chiavi di configurazione suddivise in sezioni introdotte da un nome fra parentesi quadre, corrispondente al sottocomando a cui le chiavi fanno riferimento. Alcune chiavi di configurazione fanno a loro volta riferimento ad insiemi di valori, pertanto la loro assegnazione non è ad un valore diretto, ma al nome di una successiva sezione in cui sono raggruppate le assegnazioni delle ulteriori sottochiavi. Faremo riferimento alla versione del file distribuita con il pacchetto `openssl` di Debian Squeeze.

Nella sezione `[req]` si trovano le configurazioni relative alla generazione di richieste di certificato con il sottocomando `req`. Fra queste c'è la dimensione di default delle chiavi private che viene specificata con la chiave di configurazione `default_bits` mentre i valori di default ed i limiti che devono essere applicati ai vari campi delle informazioni di tab. 2.1 vengono specificati dalla chiave `distinguished_name`. Questa fa riferimento ai valori che sono raggruppati nella successiva sottosezione `[req_distinguished_name]`,[30] dove si trovano le assegnazioni delle ulteriori di chiavi di configurazione che consentono di specificare descrizione, valori di default e dimensioni dei vari dati; queste sono nella forma `countryName`, `countryName_default`, `countryName_min`, ecc. Essendo il formato piuttosto esplicito non staremo a dilungarci.

Le impostazioni per la gestione di una *Certification Authority* sono contenute nella sezione relativa al comando `ca`, la quale rimanda direttamente alla sottosezione `[CA_default]`. È all'interno di questa che si controllano i valori di default usati nella gestione della propria CA, in particolare i vari default per il nome della directory dove si salvano i vari file (è qui che è indicato l'uso di `demoCA` e degli ulteriori file e sottodirectory illustrati in precedenza) ma soprattutto la chiave `default_days` che è quella che imposta la durata di un certificato all'atto della firma.

Sempre qui la chiave `x509_extensions` rimanda alla successiva sezione `[usr_cert]` per configurare quali delle informazioni aggiuntive previste nelle *X509v3 extensions* devono essere inserite nel certificato all'atto della firma, presenti nella omonima sezione del certificato (vedi tab. 2.2).

[30] in sostanza nel file si trova impostato `distinguished_name = req_distinguished_name`.

Questa è in genere la sezione più rilevante per l'uso pratico perché queste informazioni aggiuntive possono essere usate dai client del protocollo SSL per prendere decisioni riguardo la validità o meno del certificato.

Valore	Significato
`objsign`	certificato per firma digitale (*Object Signing*).
`email`	certificato per posta elettronica (*S/MIME*).
`client`	certificato di tipo client (*SSL Client*).
`server`	certificato per server (*SSL Server*).

Tabella 2.6: Valori possibili per la chiave di configurazione `nsCertType` di `openssl.cnf`.

Una di queste informazioni riguarda la tipologia di uso del certificato a cui abbiamo accennato anche in sez. 2.1.2; questa viene impostata con la chiave `nsCertType`. Di default questa chiave non viene definita per cui la relativa informazione non viene aggiunta, ma qualora la si voglia indicare si deve assegnare detta chiave ad uno o più dei valori illustrati in tab. 2.6 (se sono più di uno si deve usare una lista separata da virgole), che indicano, ai programmi che usano questa informazione, quale è l'uso consentito per il certificato.

Un'altra informazione importante che si imposta in questa sezione è quella relativa ai nomi alternativi (*X509v3 Subject Alternative Name*) che si possono associare al *Common Name* specificato nel *Subject* del certificato. Questo consente ad esempio di inserire nel certificato ulteriori nomi a dominio, indirizzi IP o indirizzi di posta elettronica che verranno ritenuti validi per una connessione SSL da un programma che utilizzi questa estensione. Si tenga presente però che questa informazione, come tutte le altre della sezione *X509v3 extensions*, viene inserita all'atto della firma, cosa che può essere fatta solo da chi gestisce la CA, e non da chi genera la richiesta di certificato.

Tipo	Significato
`DNS`	un nome a dominio, da indicare come FQDN.
`IP`	un indirizzo IP, da indicare in notazione *dotted decimal*.
`email`	un indirizzo email, da indicare nella notazione elementare `nome@dominio.it`.

Tabella 2.7: Tipi utilizzabili per i valori della chiave di configurazione `subjectAltName` di `openssl.cnf`.

La chiave di configurazione che consente di impostare queste informazioni è `subjectAltName`; anche questa di default è commentata per cui la relativa informazione non viene aggiunta ed anche in questo caso si può indicare una lista di valori, sempre separati da virgole. I valori possono indicare diversi tipi di nomi alternativi, e devono essere specificati secondo il formato "`TIPO:valore`", dove i vari tipi si sono riassunti in tab. 2.7.

Come esempio di personalizzazione delle informazioni inserite in un certificato consideriamo di avere modificato `openssl.cnf` definendo le seguenti chiavi di configurazione:

openssl.cnf

```
...
default_days    = 730
...
nsCertType      = server, client
...
```

```
subjectAltName  = DNS:gapil.gnulinux.it,IP:78.46.114.60
...
```

si otterrà, eseguendo la firma con questa nuova versione di `openssl.cnf`, un certificato con una scadenza di due anni, ed al cui interno compariranno le ulteriori informazioni aggiuntive:

```
$ ./CA.sh -sign
Using configuration from ./openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            89:77:13:a8:58:7b:ae:42
        Validity
            Not Before: Jun 15 14:35:16 2011 GMT
            Not After : Jun 14 14:35:16 2013 GMT
        Subject:
            countryName               = IT
            stateOrProvinceName       = FI
            localityName              = Firenze
            organizationName          = Simone Piccardi
            organizationalUnitName    = gnulinux
            commonName                = piccardi.gnulinux.it
            emailAddress              = piccardi@gnulinux.it
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Client, SSL Server
            ...
            ...
            X509v3 Subject Alternative Name:
                DNS:gapil.gnulinux.it, IP Address:78.46.114.60
Certificate is to be certified until Jun 14 14:35:16 2013 GMT (730 days)
Sign the certificate? [y/n]:y
...
```

Come si vede la procedura di firma è abbastanza grezza dato che se si deve cambiare la durata o uno aggiungere o modificare uno qualunque dei dati aggiuntivi previsti per la sezione delle *X509v3 extensions*, si deve eseguire la relativa modifica a `/etc/ssl/openssl.cnf` (o ad una sua alternativa) tutte le volte che si esegue una firma. Una cosa che rende alquanto scomodo l'uso di questi script quando si hanno richieste appena un po' più sofisticate e diversificate rispetto a valori di default uguali per tutti, come potrebbe essere anche solo l'inserimento di un nome a dominio alternativo.

2.2 Il *web of trust*

In questa sezione tratteremo quel modello alternativo per la realizzazione di una infrastruttura di chiavi pubbliche che viene denominato "*web of trust*" (cioè "*ragnatela di confidenza*") e che si pone come principale alternativa rispetto a quello delle *Certification Authority* che abbiamo appena esaminato. In questo contesto prenderemo in esame il programma GPG (*GNU Privacy*

Guard), che è la principale applicazione che usa questo modello alternativo, analizzandone il funzionamento e le modalità di utilizzo.

2.2.1 Il modello del *web of trust*

Il modello del *web of trust* è il principale (se non l'unico) modello alternativo rispetto a quello del *Trusted Third Party* per la realizzazione di una PKI (intesa nel significato generico e non come la implementazione classica vista in sez. 2.1. Anche in questo caso si continuano ad utilizzare gli stessi algoritmi crittografici a chiave pubblica già illustrati in sez. 1.2.3 e sez. 1.2.4 per la cifratura, l'autenticazione e la verifica di integrità, si cambiano però le modalità per rispondere alle problematiche poste dagli attacchi di *man-in-the-middle*.

In questo caso infatti, invece di delegare il compito di verifica della corrispondenza fra le credenziali ed una identità reale ad un soggetto terzo, assunto fidato a priori, si distribuisce il compito fra gli stessi utenti finali. Il meccanismo cioè è quello in cui ognuno cerca di verificare queste corrispondenze direttamente per quanto gli è possibile, ed inserisce le verifiche che ha fatto in un meccanismo di distribuzione di questa informazione.

In sostanza quello che viene fatto è che ciascun soggetto firma le chiavi pubbliche dei corrispondenti di cui ha verificato direttamente l'identità, restituendo agli stessi la chiave firmata. Ognuno potrà così raccogliere tutte le firme apposte sulla propria chiave dai terzi che l'hanno verificata direttamente, e che avendola firmata ne attestano la "*validità*".

Questo è il meccanismo che permette di realizzare quella *ragnatela di confidenza* in cui ciascuno può decidere se fidarsi o meno delle credenziali (la chiave pubblica) di un terzo non conosciuto direttamente, sulla base delle verifiche che sono state fatte dagli altri soggetti partecipanti alla ragnatela. Queste sono disponibili e possono essere controllate proprio perché su ogni chiave pubblica è prevista la presenza delle firme che sono vi sono state apposte da coloro che l'hanno verificata direttamente.

Anche in questo modello il problema resta quello dell'affidabilità delle verifiche che non sono state effettuate direttamente, e di quanto ci si possa fidare dei terzi che le compiono. Solo che stavolta le verifiche sono distribuite e l'onere della decisione sulla fiducia o meno sulle stesse non viene delegato ad un terzo di cui si è obbligati a fidarsi, e invece può essere determinato in base alla fiducia che ciascuno può assegnare ad ogni altro membro della ragnatela.[31] In questo modo anche se non si ha una verifica diretta, ma ve ne sono molte indirette da parte di terzi, si può decidere di aver raggiunto un ragionevole livello di fiducia.

Per questo motivo e per la sua natura distribuita, almeno dal punto di vista teorico, il modello del *web of trust* è più sicuro, sia perché consente un controllo più dettagliato dell'attribuzione della fiducia, sia perché in questo caso la compromissione di una *Certification Authority* non è sufficiente per creare una falsificazione generale di qualunque chiave, come invece è già successo nel caso di un *Trusted Third Party* risultato non troppo affidabile.[32]

Inoltre il modello si presta naturalmente anche alla distribuzione diretta delle informazioni, è infatti possibile per il singolo pubblicare le proprie chiavi pubbliche, corredate di tutte le firme

[31] vedremo in seguito come ciascuno può stabilire un valore di confidenza (il cosiddetto *trust*) relativo ad ogni altro membro della ragnatela.

[32] si fa riferimento a quanto verificatosi nel 2011 con la compromissione di varie CA (Comodo, DigiNotar) che ha portato all'emissione di numerosi certificati falsi per conto anche di terzi che non facevano riferimento a dette *Certification Authority*.

raccolte, così che un terzo possa ottenerle direttamente, ed esiste una rete dei cosiddetti *keyserver* che si scambiano direttamente queste informazioni, a cui ci si può rivolgere per ottenere le chiavi pubbliche altrui e pubblicare le proprie.

Il sistema fornisce inoltre anche una risposta molto più efficace ai problemi dovuti alla eventuale compromissione di una chiave privata, è infatti possibile, fintanto che se ne possiede una copia, perché ovviamente questa operazione può essere effettuata solo dal proprietario, generare una revoca della propria chiave. Questa revoca potrà essere pubblicata e se viene inviata alla citata rete di *keyserver* verrà immediatamente incorporata nei dati della propria chiave pubblica rendendola invalida una volta per tutte. Si evitano così tutti i problemi relativi alla distribuzione periodiche di CRL.

Benché con l'introduzione di PGP (di cui parleremo più avanti) questo modello sia nato e si sia sviluppato in forma alternativa a quello delle *Certification Authority* illustrato in sez. 2.1.1, e faccia quindi riferimento alla nomenclatura classica di chiave pubblica e chiave privata, pur utilizzando un diverso formato per le informazioni, esso resta comunque completamente generico, e può essere implementato anche usando le chiavi e i certificati illustrati in sez. 2.1.2.

Infine si tenga presente che i due modelli si possono sovrapporre, ed in particolare si possono realizzare sistemi misti in cui si sovrappone un ente certificatore centrale alle verifiche distribuite. Un esempio di questo modello è quello realizzato dal progetto *CAcert.org*.

2.2.2 Una panoramica su GPG/PGP

Il progetto *GNU Privacy Guard* (GPG) nasce come reimplementazione libera di PGP (*Pretty Good Privacy*), un programma per la cifratura della posta elettronica scritto da Philip Zimmerman nel 1991 e considerato come la prima applicazione ad aver utilizzato su larga scala la crittografia a chiave asimmetrica.

Utilizzato inizialmente per la cifratura dei messaggi di posta elettronica, il programma permette anche di cifrare file arbitrari, e supporta anche la creazione di firme digitali, tanto che oggi viene comunemente usato come meccanismo principale nella verifica di autenticità ed integrità dei pacchetti software distribuiti via rete dalle principali distribuzioni di GNU/Linux.

Dato che PGP non era software libero e che per le versioni più recenti non sono neanche disponibili i sorgenti, per cui non se ne può verificare la sicurezza,[33] GPG venne creato come alternativa libera e sviluppato anche grazie ad un sostanzioso finanziamento da parte del governo tedesco, che voleva poter disporre di un programma che garantisse in maniera adeguata la sicurezza delle proprie comunicazioni. A tutti gli effetti è equivalente al più famoso antenato ed è usato correntemente al suo posto.

Entrambi i programmi usano crittografia a chiave asimmetrica e sono compatibili fra loro. Noi tratteremo solo GPG ma la nomenclatura è nata con PGP, per cui nella stessa si fa riferimento a questo programma, anche se quanto diremo si applica in maniera identica anche per GPG. In questo caso non esistono certificati e chiavi, ma si parla di *chiave pubblica PGP* (*PGP public key*) e di *chiave privata PGP* (*PGP private key*) che insieme costituiscono la cosiddetta "*coppia di chiavi*" (quella che in gergo si chiama *keypair*). Essendo PGP nato come sistema per lo scambio sicuro di posta elettronica, non è previsto niente di simile al protocollo SSL, e non è possibile cifrare in maniera generica il traffico di rete usando GPG.

[33] in particolare non è possibile sapere se non vi siano state inserite delle *backdoor*, che è critico per gli scopi del programma.

Il formato delle chiavi PGP e l'intero funzionamento del sistema è stato standardizzato con il nome di *OpenPGP*,[34] ed è a questo standard che fanno riferimento le varie implementazioni. Entrambi i tipi di chiave possono essere rappresentate in forma testuale, in un formato molto simile al PEM, con dati codificati in *base64* racchiusi fra delle linee di intestazione che indicano il contenuto.

Come un certificato anche una chiave pubblica PGP contiene informazioni ulteriori oltre ai semplici elementi crittografici, ma non essendovi mai stata, neanche dal punto di vista teorico, una autorità centrale, non esiste niente di affine alle informazioni dello standard X.509 che si trovano nei certificati.

Una chiave pubblica PGP è costituita dalla parte pubblica di una chiave asimmetrica principale (detta *master signing key*), dalle parti pubbliche delle sottochiavi e da una lista di "*identificativi utente*" (il cosiddetto *User ID*). Il sistema infatti prevede che la chiave principale venga usata per solo per firmare le sottochiavi, che sono quelle che possono essere usate per firmare e cifrare i messaggi; queste fanno la vece dei certificati intermedi, e possono essere aggiunte e rimosse. La presenza di queste sottochiavi e di tutti gli identificativi sono validate dalla firma dalla *master signing key*.

Ciascuna chiave e sottochiave porta con sé, oltre alle informazioni crittografiche una data di creazione, una scadenza ed un identificativo, detto *Key-ID*; questo corrisponde alle prime otto cifre della cosiddetta *impronta digitale* (la *fingerprint*) della chiave, che altro non è che l'hash crittografico della stessa. In genere si usa la *fingerprint* della *master signing key* ed il relativo *Key-ID* per identificare univocamente una chiave pubblica PGP, ed è a questo che si fa riferimento quando si parla genericamente di *fingerprint* e di *Key-ID* di una chiave.

Gli "*identificativi utente*" (o *User ID*) servono invece ad indicare la persona o ad l'entità a cui è associata la chiave PGP, e sono composti a loro volta di un nome e cognome, di un commento, e di un indirizzo di posta elettronica, nella forma:

```
Nome Cognome (Commento) <emailuser@dominio.it>
```

in genere se ne indica almeno uno in fase di creazione, a cui ne possono essere aggiunti in seguito degli altri, corrispondenti ad esempio ai diversi indirizzi di posta che una persona può avere. Come per le sottochiavi gli identificativi possono essere aggiunti e rimossi.

Per assicurare l'integrità delle informazioni relative alle sottochiavi ed alle identità presenti in una chiave pubblica, queste sono comunque sempre firmate dalla *master signing key* cosicché risulti impossibile ad un terzo aggiungere ad essa una identità o una sottochiave fittizia. Una chiave privata PGP ha una struttura analoga a quella di una chiave pubblica, ma in questo caso contiene come elementi crittografici solo le parti private della *master signing key* e di tutte le altre sottochiavi, e non vi sono presenti né gli identificativi utente, né le firme ricevute.

Infine con la chiave pubblica sono distribuite anche tutte le firme che questa ha ricevuto da parte dei terzi che ne hanno controllato la validità,[35] che in questo caso però non vengono da una CA, ma da tutti gli altri utenti di GPG/PGP che la hanno verificata, cosa che viene fatta o per conoscenza diretta del titolare della chiave o in uno dei tanti *key signing party* che vengono organizzati presso conferenze, congressi e riunioni varie.

[34] la versione più recente dello standard è specificata nell'RFC 4880.

[35] fra queste è sempre compresa, anche se non elencata esplicitamente la firma da parte della propria *master signing key*.

In tal caso è bene portarsi con se almeno due documenti di identità ed abbondanti stampe della *fingerprint* della chiave da cedere ai presenti. I documenti sono necessari a convincere gli altri partecipanti che voi siete effettivamente voi,[36] la *fingerprint* (in genere stampata in esadecimale su un biglietto insieme alle altre informazioni) ed il relativo *Key-ID* serve a non doversi portar dietro l'intera chiave pubblica fornendo comunque al corrispondente tutto il necessario perché questi possa procurarsela essendo sicuro che sia quella "*giusta*".

2.2.3 Il comando `gpg` e l'uso di GPG

Benché esistano diversi programmi di gestione dotati di interfaccia grafica, noi tratteremo soltanto la versione originale di GPG a riga di comando, `gpg`. Il programma è estremamente vasto e complesso, le varie operazioni che può eseguire vengono attivate tramite un lunghissimo elenco di opzioni la gran parte delle quali si devono specificare in formato esteso.

Alcune di queste costituiscono dei veri e propri sottoprogrammi e devono essere usate singolarmente, attivando speciali operazioni interattive ed anche una riga di comando interno. Altre controllano le operazioni più ordinarie e possono richiedere o meno l'indicazione di un argomento al programma, il cui significato varia a seconda dell'opzione stessa. Infine alcune possono essere usate in combinazione con le altre per controllare le modalità di funzionamento del programma.

Si sono riportate le opzioni più comuni nell'uso ordinario del programma, che in genere sono quelle per cui è presente anche una versione in forma non estesa, in tab. 2.8. Di tutte le altre tratteremo qui soltanto le più rilevanti, mentre per gli approfondimenti e tutti i dettagli si rimanda alla abbondante documentazione disponibile a partire dalla pagina di manuale.

Il primo passo per l'uso di GPG è quello di generare una propria coppia di chiavi. Questa è in genere una operazione da eseguire una volta per tutte, e non rientra nell'uso ordinario del programma. Per farlo occorre invocare `gpg` con l'opzione `--gen-key` che fa partire la procedura interattiva in cui vengono richieste tutte le informazioni necessarie. Dato che il programma è di uso personale, questo deve essere usato come utente normale, e tutti i suoi dati verranno mantenuti nella directory `.gnupg` della propria home directory.

Questo significa anche che non è il caso di usarlo su macchine di cui non si è il solo amministratore, perché per quanto le chiavi private siano protette da una *passphrase* esistono molti modi (ad esempio l'installazione di una versione "*taroccata*" del programma) con cui un amministratore potrebbe impossessarsi della vostra chiave privata.

Un esempio di creazione delle proprie chiavi, fatto passo per passo, è quello che riportiano di seguito. Anzitutto occorre scegliere algoritmi di crittografia, dimensione delle chiavi e scadenza:

```
$ gpg --gen-key
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
```

[36] questo se volete dimostrare la vostra identità pubblica, è possibile anche usare identità anonime, nel qual caso però dovrete dimostrare di essere chi dichiarate di essere in modo diverso.

```
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
```

ed in questo caso si sono accettati i default per le impostazioni e si è risposto di sì alla domanda finale (in caso di errore in una delle scelte si può ripetere la procedura rispondendo no), una volta fatto questo bisogna scegliere un opportuno *User ID*:

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Truelite Srl
Email address: info@trl.tl
Comment: Truelite
You selected this USER-ID:
    "Truelite Srl (Truelite) <info@trl.tl>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

dove vengono richieste le varie informazioni che ne fanno parte. Una volta completata la fornitura delle informazioni di identificazione verrà chiesta una *passphrase* per proteggere la propria chiave privata. Dato che la *passphrase* costituisce l'unica difesa contro un uso non autorizzato della chiave (in caso di compromissione o perdita della macchina su cui si lavora) è opportuno che sia scelta con molta cura e sia sufficientemente lunga e complessa; non venendo visualizzata la si dovrà immettere due volte per conferma:

```
You need a Passphrase to protect your secret key.

Enter passphrase:
Repeat passphrase:
```

una volta fatto questo il programma provvederà alla generazione della chiave privata, per farlo è necessario avere una quantità sufficiente di numeri casuali, cosa che in genere causa la richiesta di generare sufficiente entropia nel sistema con varie azioni,[37] una volta ottenuta la quale verranno creati i dati necessari e stampato un riassunto conclusivo:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

[37] la pressione di tasti fa riferimento alla tastiera fisica, se si esegue il comando in remoto con una sessione `ssh` la cosa non avrà alcun effetto (la rete non è considerata una fonte affidabile di rumore).

```
Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 144 more bytes)
....+++++
..........+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 24 more bytes)
..+++++
..+++++
gpg: /home/piccardi/.gnupg/trustdb.gpg: trustdb created
gpg: key 58AF19F7 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/58AF19F7 2011-07-07
      Key fingerprint = 3A62 34E7 E2EB E1DA 3EF5  9D60 66F3 F29A 58AF 19F7
uid                  Truelite Srl (Truelite) <info@trl.tl>
sub   2048R/F1493ABF 2011-07-07
```

Si tenga conto che `gpg` consente di gestire anche più coppie di chiavi; se pertanto si vogliono usare due chiavi diverse per due identità diverse si potrà ripetere l'operazione di generazione delle chiavi una seconda volta, e la nuova chiave verrà aggiunta nel proprio *keyring* (il "*portachiavi*") all'interno di `.gnupg`. É sempre possibile avere due chiavi con riferimento ad un *User ID* identico, in caso di ambiguità infatti si potrà fare comunque riferimento al *Key-ID* per indicare quale delle chiavi si vuole usare.

Un secondo passo iniziale, che si consiglia di mettere in pratica subito dopo la creazione di una coppia di chiavi, è quello di generare un certificato di revoca. Questa è una misura di precauzione nei confronti dell'eventualità della perdita totale della chiave privata, (come lo smarrimento o il furto dell'unico computer su cui la si teneva) o di impossibilità di accesso alla stessa, (ad esempio o se non ci si ricorda più la passphrase) che permette di invalidare permanentemente la propria chiave PGP anche quando questa non è più disponibile. É ovviamente sempre possibile generare in un secondo tempo il certificato in caso di perdita o compromissione della propria chiave, ma per farlo occorre averne a disposizione una copia.

Ovviamente occorrerà proteggere adeguatamente tale certificato di revoca, perché se questo pervenisse in mano ad un terzo ostile questi potrebbe invalidare la chiave a cui fa riferimento. La sua perdita però è meno dannosa rispetto a quella della chiave stessa, in quanto il suo scopo è solo quello della revoca, e quindi non è possibile usarlo per impersonare il titolare della chiave.

Per la generazione del certificato di revoca occorre usare l'opzione `--gen-revoke` che, come molte altre opzioni di `gpg`, richiede che si specifichi come argomento del comando la chiave a quale si fa riferimento. Questo si può fare in due modi, si può indicare il valore in esadecimale del *Key-ID* della chiave scelta, o si può indicare una stringa il cui testo verrà usato per individuare la chiave con una ricerca nel testo degli *User ID* ad esse associati; nel caso di corrispondenze multiple verrà sempre selezionata la prima, per evitare le ambiguità occorre usare il *Key-ID*.

Un esempio della generazione del certificato di revoca, in cui si è usata anche l'opzione `--output` per indicare un file di uscita (altrimenti il testo del certificato sarebbe stato stampato sullo *standard output*), è il seguente:

```
$ gpg --output revoca.asc --gen-revoke Truelite
sec  2048R/58AF19F7 2011-07-07 Truelite Srl (Truelite) <info@trl.tl>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 0
Enter an optional description; end it with an empty line:
> Revoca precauzionale
>
Reason for revocation: No reason specified
Revoca precauzionale
Is this okay? (y/N) y

You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

Una volta fatto questo si otterrà all'interno del file `revoca.asc` (la convenzione è quella di usare l'estensione `.asc` per i file che contengono rappresentazioni testuali delle chiavi PGP) si otterrà un contenuto del tipo:

revoca.asc

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)
Comment: A revocation certificate should follow

iQEzBCABAgAdBQJOAKulFh0AUmV2b2NhIHByZWNhdXppb25hbGUACgkQm3VwX32z
UE0hNwgA2/+LV+6QGlVrDBRgFwoEoKgq378s9S6Jy0hC/Yru52DnAVgV/SoUvTj3
...
jIG1qsLdzLJCNzNf+Rkyh6ywGsOjWw==
=g2AK
-----END PGP PUBLIC KEY BLOCK-----
```

e si noti come in precedenza il comando ci abbia notificato che è stata forzata una uscita in formato testuale (il cosiddetto *ASCII armored*); questo rende possibile anche una semplice stampa del certificato.

Una volta che si disponga di una propria coppia di chiavi si dovrà iniziare ad utilizzarla. Perché altri ci possano inviare messaggi cifrati dovranno però disporre della nostra chiave pubblica, lo stesso vale qualora volessero firmarla dopo aver verificato la nostra identità, occorre pertanto avere un meccanismo di distribuzione.

Il modo più semplice per distribuire la propria chiave pubblica è quello di pubblicarla su un file, per questo esiste l'opzione `--export` che consente di salvare un chiave pubblica in un formato standard che la renda utilizzabile da terzi. Se usata da sola i dati verranno stampati a video in formato binario, pertanto è comune usare l'opzione `--output` (abbreviabile in `-o`) per indicare un file di destinazione e l'opzione `--armor` (abbreviabile in `-a`) per richiedere la codifica *ASCII armored*. In questo modo la chiave sarà in formato testo, che può essere facilmente scambiato per email o pubblicato, e la cui copia non richiede nulla di più di un taglia ed incolla.

Dato che come accennato un *keyring* personale può contenere diverse chiavi, sia pubbliche che private, anche `--export` richiede che si indichi quella che si vuole esportare con un *User ID* o un *Key-ID*, con le stesse convenzioni riguardo la scelta *User ID* sulla base della stringa passata come argomento già illustrate per `--gen-revoke`. Un esempio di esportazione è allora il seguente:

```
$ gpg --output key.gpg --armor --export info@trl.tl
```

e si otterrà un contenuto di `key.gpg` del tipo:

key.gpg

```
 -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)

mQENBE4ApmQBCADv7AuurV1SnpOtylZazkl39QOAQdMhGyPtr8AVFJOKVd8qr5Da
...
awgseHHXiHZD2Mdc7p5SQTjqzWTW1ZcM1zItbpkt8xjcXCdCX8NhXyQT2msqKyGT
=TwxL
-----END PGP PUBLIC KEY BLOCK-----
```

Una volta ottenuto il file con la chiave, o il relativo testo, lo si potrà inviare ai propri corrispondenti, ma una delle funzionalità più interessanti di `gpg` è che il programma supporta direttamente anche l'interazione con i cosiddetti *keyserver*, una rete di server presenti su Internet e dedicati alla distribuzione delle chiavi, che possono essere interrogati direttamente tramite il comando con le due opzioni `--send-keys` e `--recv-keys`.

In particolare `--send-keys` di inviare ad un *keyserver* una qualunque delle chiavi pubbliche che si hanno nel portachiavi, da indicare esclusivamente per *Key-ID*. Pertanto si potrà eseguire la pubblicazione di una chiave con:

```
$ gpg --send-key 58AF19F7
```

ed il comando inserirà la chiave nel *keyserver* la prima volta, e la aggiornerà con eventuali dati (altre identità o firme ricevute) le volte successive.

Una volta che una chiave è stata pubblicata su un *keyserver* sarà possibile per chiunque scaricarla direttamente via rete, `gpg` supporta una specifica opzione, `--recv-keys`, che scarica ed importa direttamente una chiave, sempre indicata per *Key-ID*, con un comando del tipo:

```
$ gpg --recv-keys 2A972F9D
gpg: requesting key 2A972F9D from hkp server keys.gnupg.net
gpg: key 2A972F9D: public key "Simone Piccardi (Truelite SrL) <piccardi@truelite.it>" imported
```

```
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: Total number processed: 1
gpg:               imported: 1
```

Nei due esempi precedenti si è sempre usato il *keyserver* di default, `keys.gnupg.net`, ma è sempre possibile usare l'ulteriore opzione `--keyserver` per specificare un *keyserver* alternativo; l'opzione richiede l'indicazione dello stesso come parametro da specificare in forma di indirizzo IP o di nome a dominio.[38]

Se invece si dispone direttamente di un file contente una chiave la si potrà importare direttamente con l'opzione `--import`, se non si specifica altro il comando leggerà la chiave dallo standard input, altrimenti si potrà passare come argomento il file che la contiene. Pertanto se si è scaricata la chiave con cui viene firmato l'archivio di Debian Squeeze nel file `archive-key-6.0.asc`, la si potrà importare con:

```
$ gpg --import archive-key-6.0.asc
gpg: key 473041FA: public key "Debian Archive Automatic Signing Key (..." imported
gpg: Numero totale esaminato: 1
gpg:              importate: 1  (RSA: 1)
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
```

Si tenga presente che importare una chiave non significa ancora darle alcun livello di fiducia e che il fatto che sia disponibile su un *keyserver* non ha nessuna implicazione su questo piano; la chiave resterà considerata *untrusted* fintanto che non la si sia verificata esplicitamente (ed eventualmente firmata) o sia firmata da altri corrispondenti dei quali si è stabilito di avere fiducia (torneremo sulla gestione di questi aspetti più avanti).

Opzione		Significato
`--output`	`-o`	indica di scrivere l'uscita del comando sul file passato come parametro.
`--armor`	`-a`	indica la generazione di una uscita in testo puro codificato, il cosiddetto *ASCII armored*.
`--recipient`	`-r`	indica il destinatario per cui si vuole cifrare, da indicare per *User ID* o *Key-ID*.
`--encrypt`	`-e`	esegue la cifratura del file indicato come argomento del comando (o letto dallo *standard input*).
`--decrypt`	`-d`	esegue la decifratura del file indicato come argomento del comando (o letto dallo *standard input*).
`--sign`	`-s`	crea una firma del file indicato come argomento (o letto dallo *standard input*).
`--detach-sign`	`-b`	firma il file indicato come argomento (o letto dallo *standard input*) su un file a parte.
`--local-user`	`-u`	richiede di usare la chiave privata corrispondente all'*User ID* passato come parametro.
`--verify`	–	richiede la verifica della firma.

Tabella 2.8: Principali opzioni di `gpg`.

[38] in realtà il parametro ha una forma generica più complessa: `scheme://keyserver.name.or.ip:port` dove `scheme` può assumere i valori `hkp`, `ldap` o `mailto` per l'uso di diversi schemi di comunicazione (via HTTP, LDAP o email); dato che la gran parte dei *keyserver* usa il protocollo via web, si omette `hkp://` essendo questa il default.

Una volta completate le operazioni preliminari per creare la propria coppia di chiavi ed ottenere le chiavi dei corrispondenti che ci interessano, l'uso ordinario di `gpg` consiste principalmente nella ricezione e nell'invio di documenti cifrati o firmati o entrambi. Le opzioni relative a queste operazioni, insieme alle altre di più comune utilizzo, sono riassunte in tab. 2.8.

Una volta che si disponga della chiave di un corrispondente si potrà cifrare un documento per lui con l'opzione `--encrypt` (abbreviabile in `-e`); questa opzione richiede l'uso della successiva opzione `--recipient` (abbreviabile in `-r`) per indicare il destinatario che deve essere specificato con un ulteriore parametro che ne identifichi la chiave pubblica, al solito il parametro viene usato come stringa di ricerca negli *User ID* delle chiavi pubbliche presenti nel proprio *keyring* o come indicazione del *Key-ID*.

L'opzione `-r` può essere usata più volte se i destinatari sono più di uno. Di nuovo i dati da cifrare verranno letti dallo *standard input* a meno di non indicare un file come argomento, e verranno scritti sullo *standard output* in firmato binario a meno di non usare le opzioni `-o` per indicare un file diverso e `-a` per richiedere l'*ASCII armored*.

Si tenga presente inoltre che qualora si cifri verso un corrispondente non verificato il comando stamperà le caratteristiche della chiave e chiederà se andare avanti lo stesso, fermandosi se non si da una approvazione esplicita; si otterrà pertanto un risultato del tipo:

```
$ gpg -o cifrato.gpg -a -e --recipient 26257B68 chiaro.txt
gpg: 56EC3680: There is no assurance this key belongs to the named user

pub  2048g/56EC3680 1999-09-23 Christian Surchi <christian@truelite.it>
 Primary key fingerprint: D1E2 9A9D 1712 0E94 8671  834E 0CFF 30E1 2625 7B68
      Subkey fingerprint: 4152 402F 61B0 3D3E 06A4  A587 18B4 4D99 56EC 3680

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

e si dovrà indicare di usare la chiave esplicitamente.[39] A questo punto nel file `cifrato.gpg` si avrà il messaggio cifrato che si potrà inviare al destinatario, con un contenuto del tipo:

cifrato.gpg

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.11 (GNU/Linux)

hQQOA9pvZzRosObsEBAAsixRt/H5+tCoofUcpKZGCbPB/1Nfb1I9Jgp7pvHoP4xx
...
ni9WWMSr1bnA4oHEcpGNyEe4J5I8r1IF9Rm2N6Tl/J+kElg=
=4ZR7
-----END PGP MESSAGE-----
```

e soltanto il destinatario potrà decifrarlo.

Qualora invece si voglia firmare un documento, invece di cifrarlo, si hanno due possibilità: con l'opzione `--sign` (abbreviabile in `-s`) si può generare un file firmato, contenente sia il contenuto

[39] la cosa si può fare quando quello che interessa è la cifratura dei dati e non l'autenticità del corrispondente, per essere ragionevolmente sicuri di questa, e rimuovere l'avviso del comando, occorrerà assegnare un livello di fiducia alla chiave, cosa che vedremo più avanti.

originale che la firma digitale. Spesso però è più semplice, ad esempio quando si pubblicano dei file, mettere a disposizione la firma su un file a parte con la cosiddetta *detached signature*, nel qual caso occorrerà invece usare l'opzione `--detach-sign` (abbreviabile in `-b`). Qualora si abbiano più chiavi private (relative a diverse identità) si potrà indicare quale si vuole usare con l'opzione `--local-user` (abbreviabile in `-u`). Un esempio di creazione di un file firmato è il seguente:

```
$ gpg --sign chiaro.txt
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

Enter passphrase:
```

il cui risultato è creare un file binario `chiaro.txt.gpg` contenente sia il contenuto del file originario che la firma in forma di messaggio PGP. L'opzione si può specificare anche insieme a `-e` così da generare un messaggio cifrato e firmato allo stesso tempo. Se invece si vuole firmare un file e distribuire la *detached signature*, si potrà usare il comando:

```
$ gpg --armor --detach-sign chiaro.txt
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

Enter passphrase:
```

e in questo caso si otterrà un file separato `chiaro.txt.asc`, in formato *ASCII armored*, contenente solo la firma nella forma:

chiaro.txt.asc

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iQEcBAABAgAGBQJOFxivAAoJEGbz8ppYrxn3jOoH/i7aSgq/8PxnPIy+vjYAO+GG
...
=7ysA
-----END PGP SIGNATURE-----
```

che in questo caso dovrà essere distribuito insieme al file originario.

Per verificare una firma si deve usare invece l'opzione `--verify`, che richiede come argomento il file della firma. Se questo è un messaggio completo non occorre altro, se questo è una *detached signature* il comando cerca il file originale sulla base del nome di quello della firma (di default viene aggiunta l'estensione `.asc` o `.sig`), togliendo l'estensione, ma si può indicare come originale un file qualsiasi specificandolo come secondo argomento. Pertanto si potrà verificare la firma del precedente file con:

```
$ gpg --verify chiaro.txt.asc
gpg: Signature made Fri Jul  8 16:48:15 2011 CEST using RSA key ID 58AF19F7
gpg: Good signature from "Truelite Srl (Truelite) <info@trl.tl>"
```

in cui il riferimento al file originale `chiaro.txt` è automatico.

Se infine si deve decifrare un messaggio che ci è stato inviato l'opzione relativa è `--decrypt` (abbreviabile in `-d`), che richiede si specifichi il file con il messaggio. Il comando decifra il

messaggio e lo stampa, se questo è firmato verifica automaticamente anche la firma, posto che si abbia la chiave pubblica del mittente. Un esempio del relativo risultato è:

```
$ gpg --decrypt cifrato.gpg
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID F1493ABF, created 2011-07-07 (main key ID 58AF19F7)

gpg: encrypted with 2048-bit RSA key, ID F1493ABF, created 2011-07-07
      "Truelite Srl (Truelite) <info@trl.tl>"
PROVA --- PROVA --- PROVA

Prova di testo da cifrare.

PROVA --- PROVA --- PROVA
```

Come abbiamo visto in precedenza nel generare un messaggio cifrato per un corrispondente che non si conosce `gpg` richiede una conferma esplicita non potendosi considerare fidata la relativa chiave. Questa non è che la diretta conseguenza del modello del *web of trust* in cui ciascuno deve assumersi la responsabilità di decidere di chi si fida o meno.

L'opzione di `gpg` che consente di effettuare le varie operazioni di gestione delle chiavi è `--edit-key` che richiede si passi come argomento un *User ID* o un *Key-ID*. Quando usata l'opzione stampa i dati riassuntivi della chiave (con i vari *User ID* ad essa associati) e poi porta su una riga di comando interna da cui si possono eseguire le varie operazioni, ad esempio:

```
$ gpg --edit-key 2A972F9D
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.


pub  1024D/2A972F9D  created: 2000-08-14  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  4096g/68B0E6EC  created: 2000-08-14  expires: never       usage: E
[ unknown] (1). Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
[ revoked] (2)  Simone Piccardi <piccardi@fi.infn.it>
[ unknown] (3)  Simone Piccardi <piccardi@firenze.linux.it>
[ unknown] (4)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.it>
[ unknown] (5)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.org>

gpg>
```

e si noti come nella prima parte diano riportati i dati relativi alla validità (`validity`) ed al livello di fiducia (`trust`) associato alla chiave, che essendo questa stata soltanto scaricata risultano indefiniti.

Una volta ottenuta la riga di comando interna si potranno eseguire le varie operazioni di gestione della chiave; per queste si potranno usare i vari comandi interni che `gpg` mette a disposizione, i principali dei quali sono riportati in tab. 2.9.

Come detto in generale scaricare la chiave di un corrispondente non è sufficiente a considerarla valida, perché potremmo essere soggetti a un attacco di *man-in-the-middle*, ed occorre quindi verificarne l'autenticità. Come accennato se si è avuto un incontro diretto il modo più semplice è quello di scambiarsi la *fingerprint* della chiave e verificare in seguito se essa corrisponde a quella della chiave che si è scaricato, per questo basterà usare il comando `fpr`:

Opzione	Significato
`fpr`	Stampa la *fingerprint* della chiave.
`help`	Stampa un elenco dei comandi disponibili.
`sign`	Firma la chiave con la propria.
`list`	Ristampa i dati della chiave.
`trust`	Imposta un livello di fiducia per la chiave.
`save`	Salve le modifiche fatte alla chiave ed esce.
`quit`	Esce dalla riga di comando (chiede di salvare eventuali modifiche).

Tabella 2.9: Principali comandi interni dell'opzione `--edit-key` di `gpg`.

```
gpg> fpr
pub   1024D/2A972F9D 2000-08-14 Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
 Primary key fingerprint: 8548 9BA4 FE34 A74B 4504  E8FA C12D C806 2A97 2F9D
```

e con il risultato si potrà effettuare la verifica.

Se la chiave risulta valida si potrà decidere di firmarla con il comando `sign`, il comando ci chiederà se si intendono firmare tutti gli *User ID* ad essa associati e chiederà ulteriore conferma; dopo di che verrà chiesta la passphrase per la propria chiave per poter apporre la firma. Si avrà pertanto qualcosa del tipo:

```
gpg> sign
Really sign all user IDs? (y/N) y
User ID "Simone Piccardi <piccardi@fi.infn.it>" is revoked.  Unable to sign.

pub  1024D/2A972F9D  created: 2000-08-14  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  4096g/68B0E6EC  created: 2000-08-14  expires: never       usage: E

     Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
     Simone Piccardi <piccardi@firenze.linux.it>
     Simone Piccardi (AsSoLi) <piccardi@softwarelibero.it>
     Simone Piccardi (AsSoLi) <piccardi@softwarelibero.org>

Are you sure that you want to sign this key with your
key "Truelite Srl (Truelite) <info@trl.tl>" (58AF19F7)

Really sign? (y/N) y

You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

Enter passphrase:

gpg> quit
Save changes? (y/N) y
```

e si noti come all'uscita del programma venga richiesto il salvataggio dei cambiamenti, infatti tutte le operazioni eseguite non verranno riportate nel proprio *keyring* a mano di non eseguire esplicitamente un salvataggio, cosa che esser fatta eseguendo subito il comando `save`, o uscendo dalla riga di comando con `quit`.

Se se a questo punto si ripeterà la verifica o l'accesso alla gestione della chiave questa volta si otterrà qualcosa del tipo:

```
gpg> list
pub  1024D/2A972F9D  created: 2000-08-14  expires: never       usage: SC
                     trust: unknown       validity: full
sub  4096g/68B0E6EC  created: 2000-08-14  expires: never       usage: E
[  full  ] (1). Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
[ revoked] (2)  Simone Piccardi <piccardi@fi.infn.it>
[  full  ] (3)  Simone Piccardi <piccardi@firenze.linux.it>
[  full  ] (4)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.it>
[  full  ] (5)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.org>
```

Firmare una chiave significa soltanto che si è sicuri che questa appartenga al corrispondente perché lo si è verificato in una maniera che ci soddisfa, ma questo non ha necessariamente nessuna conseguenza sul quanto ci si possa fidare delle verifiche fatte dal corrispondente, perché questi potrebbe essere sia un paranoico della sicurezza che effettua verifiche estremamente pignole, che un utilizzatore occasionale che firma qualunque chiave gli capiti di scaricare senza neanche aver troppo chiaro quello che fa.

Per questo come si può notare nonostante la chiave risulti pienamente valida, dato che la si è firmata, il livello di fiducia resta indefinito, e resterà tale fintanto che non si decide di impostarne uno con il comando `trust`. Eseguendolo ci verrà chiesto un valore numerico che lo indichi in un intervallo da 1 a 5, con relativa spiegazione dei livelli che vanno dal non so (il valore assegnato alle chiavi appena scaricate) alla fiducia assoluta (che però si usa solo per la propria chiave). Un possibile esempio potrebbe essere allora:

```
gpg> trust
pub  1024D/2A972F9D  created: 2000-08-14  expires: never       usage: SC
                     trust: unknown       validity: full
sub  4096g/68B0E6EC  created: 2000-08-14  expires: never       usage: E
[  full  ] (1). Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
[ revoked] (2)  Simone Piccardi <piccardi@fi.infn.it>
[  full  ] (3)  Simone Piccardi <piccardi@firenze.linux.it>
[  full  ] (4)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.it>
[  full  ] (5)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.org>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision? 4

pub  1024D/2A972F9D  created: 2000-08-14  expires: never       usage: SC
                     trust: full          validity: full
sub  4096g/68B0E6EC  created: 2000-08-14  expires: never       usage: E
[  full  ] (1). Simone Piccardi (Truelite SrL) <piccardi@truelite.it>
[ revoked] (2)  Simone Piccardi <piccardi@fi.infn.it>
[  full  ] (3)  Simone Piccardi <piccardi@firenze.linux.it>
```

```
[  full  ] (4)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.it>
[  full  ] (5)  Simone Piccardi (AsSoLi) <piccardi@softwarelibero.org>
Please note that the shown key validity is not necessarily correct
unless you restart the program.
```

L'assegnazione di un livello di fiducia ad una chiave consente di indicare quanto ci si fida delle altre chiavi firmate da quella chiave, ad esempio dando piena fiducia ad una chiave come fatto in precedenza diventeranno completamente valide le altre chiavi da essa firmate, così se prima della assegnazione del livello di fiducia per una altra chiave firmata dalla prima si aveva qualcosa del tipo:

```
$ gpg --edit-key 26257B68
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.


pub  1024D/26257B68  created: 1999-09-23  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  2048g/56EC3680  created: 1999-09-23  expires: never       usage: E
[ unknown] (1). Christian Surchi <christian@truelite.it>
[ revoked] (2)  Christian Surchi <csurchi@mclink.it>
...
```

in cui la validità o meno della chiave era `unknown`, dopo ci si troverà con:

```
$ gpg --edit-key 26257B68
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.


pub  1024D/26257B68  created: 1999-09-23  expires: never       usage: SC
                     trust: unknown       validity: full
sub  2048g/56EC3680  created: 1999-09-23  expires: never       usage: E
[ unknown] (1). Christian Surchi <christian@truelite.it>
[ revoked] (2)  Christian Surchi <csurchi@mclink.it>
...
```

e la validità adesso è `full`.

La ragnatela può allora costituirsi assegnando dei livelli di fiducia alle varie chiavi del proprio portachiavi, e questi saranno impiegati per stabilire la validità o meno delle chiavi da esse firmate. L'impostazione di default è che vengono considerate valide le chiavi firmata da una chiave per il quale si è assegnato una fiducia completa (*full trust*) o da almeno tre chiavi di cui si ha fiducia marginale (*marginal trust*), ma questi valori possono essere modificati.

La spiegazione dei vari dettagli della gestione del *web of trust* di `gpg` va oltre lo scopo di queste dispense, e per una trattazione approfondita si rimanda alla documentazione completa presente nella pagina di manuale del comando.

Capitolo 3

Firewall

3.1 Cenni di teoria dei firewall

In questa sezione prenderemo in esame le basi teoriche ed i criteri generali che si applicano alla realizzazione di un firewall; in particolare introdurremo alcuni concetti essenziali di sicurezza delle reti, definiremo il ruolo dei firewall in questo ambito e prenderemo in esame i criteri per la loro dislocazione.

3.1.1 Un'introduzione alla sicurezza delle reti

Prima di passare alla descrizione dei firewall e dei relativi criteri di impiego è opportuno fare una breve introduzione alle tematiche della sicurezza delle reti, la cui comprensione è necessaria per poter realizzare un uso efficace dei firewall.

In un mondo ideale infatti i firewall sono completamente inutili. Se tutto il software che si pone in uso non presentasse problemi di sicurezza, se tutte le configurazioni utilizzate fossero corrette e se tutti i meccanismi di controllo non fossero eludibili, sarebbe impossibile avere accessi non autorizzati e pertanto i firewall non servirebbero a nulla.

Nel mondo reale però tutti i programmi hanno errori e presentano vulnerabilità, le configurazioni vengono sbagliate, il software viene installato male, i controlli vengono elusi, il che comporta la possibilità che, nelle forme più varie, diventino possibili accessi non autorizzati.

In genere comunque i problemi di sicurezza più comuni sono relativi ad errori di programmazione. Il caso più comune è quello di un mancato (o incompleto) controllo dei dati che vengono inviati al programma nella supposizione, sbagliata, che questi abbiano una certa forma. Per cui inviando dati in una forma non prevista si può bloccare il funzionamento dello stesso, o anche fargli eseguire del codice estraneo (si ricordi quanto visto in sez. 1.1.3).

Buona parte di questi problemi possono essere superati in maniera molto efficace anche senza ricorrere all'uso di un firewall. La riduzione all'essenziale dei programmi da installare (parafrasando Ford un servizio non installato non si può violare), scelta di programmi stabili e realizzati con attenzione ai problemi di sicurezza, la costanza nell'eseguire gli aggiornamenti (si stima che bel oltre il 90% delle violazioni di sicurezza sia dovuta all'uso versioni vulnerabili non aggiornate) l'attenzione a tutti gli aspetti delle configurazioni (ed in particolare l'uso delle

funzionalità di controllo degli accessi che tutti i principali programmi mettono a disposizione) bastano da soli a rendere ragionevolmente sicura una macchina.

Detto questo uno potrebbe allora chiedersi a che pro installare un firewall, visto che almeno in teoria se ne potrebbe fare a meno. In realtà uno dei criteri fondamentali nella sicurezza è sempre quello della ridondanza, per cui anche la semplice duplicazione con un meccanismo indipendente dei controlli di accesso sarebbe di per sé una buona ragione. A questo poi si può aggiungere il fatto che un firewall permette di costruire un punto di accesso unificato per il traffico di rete, permettendone un controllo generale e non limitato alle funzionalità (che potrebbero non esistere neanche) dei singoli programmi o delle singole macchine.

3.1.2 Cosa è un firewall

Il primo passo per addentrarsi nella pratica delle tecnologia di protezione basate su firewall è allora quello di capire cos'è un firewall, cosa fa e a cosa serve. Troppo spesso infatti il firewall è ritenuto un po' come la panacea di tutti i mali, e si pensa che basti installarne uno per risolvere tutti i problemi di sicurezza di una rete.

Un firewall, come la parola suggerisce, è una sorta di porta blindata che permette di bloccare i tentativi di accesso ad una rete, anche se in realtà la parola inglese fa più propriamente riferimento alle porte tagliafuoco usate per bloccare la propagazione degli incendi. Ma come una porta blindata serve a poco se le finestre sono senza inferriate, ancora meno se la chiave della serratura è sotto lo zerbino, e per niente se i ladri li fate entrare voi, lo stesso vale per un firewall.

Un firewall posto fra voi ed Internet cioè non sarà mai in grado di proteggervi da attacchi provenienti da altri accessi meno controllati della vostra rete, ad esempio perché qualcuno ha attaccato un modem al suo PC e apre una connessione da lì, o c'è una rete wireless vulnerabile e qualcuno ci si aggancia. E un firewall non è in grado di proteggervi dalle vulnerabilità delle applicazioni cui dovete accedere da remoto (se ad esempio un servizio aperto è vulnerabile, il firewall non vi sarà di nessun aiuto), né tantomeno dagli attacchi portati da utenti interni alla rete protetta.

Un firewall dunque non è affatto il sostituto di una buona politica di sicurezza, di cui costituisce solo uno (per quanto importante) degli strumenti. Lo scopo specifico di un firewall è in realtà solo quello di permettervi di suddividere opportunamente la vostra rete, in modo da potervi applicare politiche di sicurezza diverse (più o meno restrittive) a seconda delle differenti esigenze di ciascuna parte.

Un altro aspetto dei firewall che è bene comprendere è che benché questi possano essere usati per filtrare il traffico di rete, il livello a cui possono farlo spesso è molto superficiale. Un firewall infatti potrà bloccare il traffico proveniente da certi indirizzi o diretto a certi servizi, ma di norma un firewall non è assolutamente in grado di riconoscere il contenuto del traffico di rete.

Se pertanto si vuole fare del *content filtering* (ad esempio filtrare le pagine web in base al loro contenuto o la posta per eliminare dei virus) occorrerà utilizzare uno strumento dedicato in grado di esaminare e classificare il contenuto dei protocolli di livello superiore e questo non è di norma un firewall, e anche se ci sono programmi chiamati "firewall" che hanno di queste funzionalità, tecnicamente non sono qualificabili come tali.

Un firewall infatti agisce ai livelli più bassi dei protocolli di rete e non è in grado di analizzare il contenuto dei pacchetti che riceve,[1] e nelle forme più elementari esso è in grado di operare solo in base ad alcuni dati presenti nel singolo pacchetto, andando ad osservare i valori contenuti nelle intestazioni dei vari protocolli di rete. I firewall più elementari di questo tipo sono detti *stateless*.

Con l'evolversi della tecnologia ci è resi conto che la sola selezione in base al contenuto delle intestazioni dei pacchetti non sempre è sufficiente ed esistono situazioni in cui può essere necessario identificare del traffico che non è riconoscibile solo in base al contenuto del singolo pacchetto. I firewall più evoluti in grado di superare questo limite sono allora detti *stateful* in quanto sono in grado di classificare, osservando il traffico nel suo insieme, i vari pacchetti in un insieme di stati,[2] permettendo la selezione in base a questi ultimi.

Si tenga presente che questo non ha comunque nulla a che fare con il *content filtering*, e gli stati non vengono definiti in termini del contenuto dei dati dei pacchetti, quanto piuttosto in termini di relazioni fra gli stessi, come ad esempio tutti i pacchetti che fanno parte di una certa connessione di rete.

3.1.3 Principi di dislocamento dei firewall

Il primo passo da fare per installare un firewall è quello di prendere carta e penna (o l'equivalente strumento preferito) per fare un bello schema della rete che si vuole proteggere, evidenziando come le varie parti sono connesse fra di loro, dove viene effettuata la connessione verso l'esterno, e decidere quali sono le macchine che devono essere raggiunte dall'esterno e quali no. Inoltre dovranno essere determinate quali sono le interrelazioni fra le varie parti della rete, ad esempio se è proprio necessario che l'amministrazione abbia accesso anche alle macchine del reparto ricerca/sviluppo e viceversa.

Per quanto possa sembrare banale, la decisione sulla *dislocazione* di un firewall è in realtà il cuore del problema, e spesso anche nella più semplice delle configurazioni (quella con un solo accesso alla rete esterna) si rischiano di fare degli errori che possono compromettere la sicurezza; ad esempio ci si può dimenticare di bloccare le connessioni dalle macchine sulla DMZ (vedremo a breve di che si tratta) alla rete interna, aprendo la possibilità di un attacco a partire da quest'ultime.

La forma più elementare di dislocazione di un firewall è quella mostrata in fig. 3.1, dove il firewall serve soltanto a proteggere la rete interna rispetto all'accesso ad internet. Nel caso di Linux se si ha un modem invece che un router esterno, si può fare eseguire anche la condivisione della connessione attraverso la stessa macchina.

Il compito del firewall in questo caso sarà quello di consentire alle connessioni provenienti dalla rete interna di uscire verso internet (ricevendo indietro le relative risposte), ma di bloccare invece le connessioni provenienti dall'esterno e dirette verso l'interno.

Questa è la forma più elementare di firewall, a parte forse il cosiddetto *personal firewall*, in cui si usa un programma di firewall per proteggere una macchina singola da tutte le connessioni

[1] ci sono strumenti dedicati a questo, come gli IDS, che tratteremo nel cap. 5.

[2] classicamente si intendeva indicare con *stateful* un firewall che riproduce la macchina a stati del protocollo TCP, in realtà il concetto è stato esteso per coprire anche *stati* di protocolli che non supportano neanche il concetto di connessione (cui di solito lo stato fa riferimento) e non è comunque detto che la macchina degli stati debba essere riprodotta interamente.

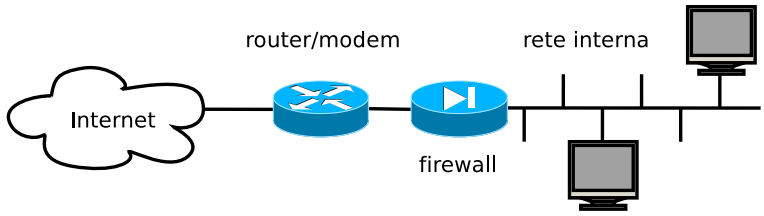


Figura 3.1: Disposizione elementare di un firewall posto a protezione di una rete con un singolo punto di accesso all'esterno che non necessita di accesso da remoto.

entranti. In questa configurazione di rete non c'è molto da dire, in quanto le direzioni dei flussi dei dati sono univoche, e la topologia della rete è elementare; la gestione della sicurezza pertanto è molto semplice e richiede solo di non consentire dall'esterno nessun accesso all'interno.

Una situazione del genere però è anche di scarsa utilità, se non nel caso di un ufficio che condivide una connessione ad internet, proprio in quanto non consente l'accesso dall'esterno a nessuna macchina interna. Il solo introdurre questo requisito rende la configurazione di fig. 3.1 inadeguata allo scopo. Infatti se anche si usasse il firewall per consentire l'accesso ad una ed una sola delle macchine all'interno della rete, questo già porterebbe ad una forte diminuzione della sicurezza della rete, dovendosi a questo punto tenere in conto la possibilità di un attacco a partire dalla macchina raggiungibile dall'esterno, attacco prima impossibile, adesso subordinato al fatto che il servizio raggiungibile dall'esterno non sia vulnerabile.

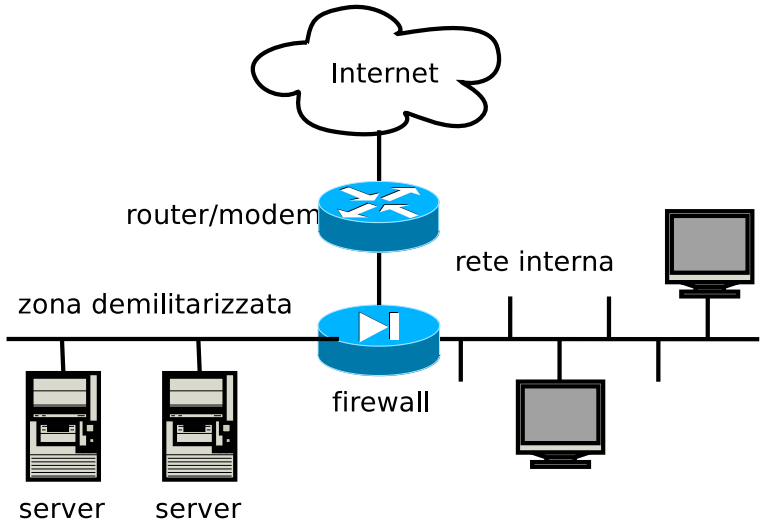


Figura 3.2: Disposizione standard di un firewall messo a protezione di una singola rete con un solo punto di accesso all'esterno e macchine accessibili da remoto.

Per questo, quando si devono fornire servizi raggiungibili dall'esterno che possono introdurre dei rischi di compromissione (i servizi web sono l'esempio più comune), si deve usare una disposizione della rete diversa, introducendo quella che viene chiamata in gergo una *zona demi-*

litarizzata, o più comunemente DMZ (sigla dell'inglese *De-Militarized Zone*), secondo lo schema di fig. 3.2.

In questo caso i servizi che devono essere raggiungibili dall'esterno devono essere messi su un tratto di rete a parte, separato dalla rete interna. Le connessioni dalla rete interna devono poter raggiungere sia internet che la zona smilitarizzata, mentre questa deve poter solo raggiungere internet, ma non la rete interna. In questo modo anche in caso di vulnerabilità di un servizio sulla DMZ la rete interna resta protetta.

Nel caso specifico si è considerato che dietro al firewall ci sia una unica rete locale, ma non è detto che debba essere così; si possono avere situazioni più complesse, con più reti separate. Di nuovo in questo caso occorre prima capire quali sono le interrelazioni fra le varie reti, e quali eventuali altri livelli di protezione si vogliono raggiungere.

3.2 Il *netfilter* di Linux

In questa sezione prenderemo in esame direttamente il funzionamento del cosiddetto *netfilter* di Linux, il sistema di manipolazione e filtraggio dei pacchetti implementato dal kernel, che consente, fra le altre cose, di costruire router e firewall estremamente potenti e flessibili.

3.2.1 La struttura del *netfilter*

Il *netfilter* è quella parte del kernel che si occupa del filtraggio e della manipolazione dei pacchetti della rete. Per poterla utilizzare deve essere stato abilitato il relativo supporto nel kernel, cosa che avviene normalmente per quelli installati da tutte le principali distribuzioni.

In fig. 3.3 è riportato lo schema di funzionamento del cuore del *netfilter* di Linux. Esso consiste in una infrastruttura che consente di inserire all'interno della parte del kernel che gestisce i protocolli di rete una serie di *punti di aggancio* (detti *hooks*, ed identificati in figura dagli ovali) in cui i vari pacchetti che arrivano sullo stack dei protocolli di rete possono essere esaminati e manipolati.

Vedremo più avanti che ciascun punto di aggancio corrisponde ad una delle cosiddette *catene predefinite* del comando `iptables`; a questo livello però il concetto di catena non esiste ancora, si tratta solo della possibilità di far eseguire, attraverso le funzioni che grazie al *netfilter* si possono agganciare a ciascun punto, una serie di elaborazioni sui pacchetti che stanno transitando, potendo decidere la sorte di questi ultimi.

L'infrastruttura del *netfilter* infatti consente alle varie funzioni poste nei punti di aggancio di decidere il destino dei pacchetti, facendoli procedere all'interno del *netfilter*, oppure scartandoli, rinviandoli sul punto di accesso (così che possano essere analizzati da altre funzioni), trattenendoli per ulteriori elaborazioni o inviandoli su una coda da cui possono essere letti da processi in user space.

Questo rende il funzionamento del *netfilter* completamente modulare, in quanto con gli opportuni moduli diventa possibile inserire nel kernel l'infrastruttura per tutte le funzionalità avanzate come il *connection tracking* e quelle relative a qualunque criterio di selezione e filtraggio dei pacchetti, fino al limite, non molto pratico, di potersi scrivere il proprio firewall direttamente in C, come un modulo del kernel.

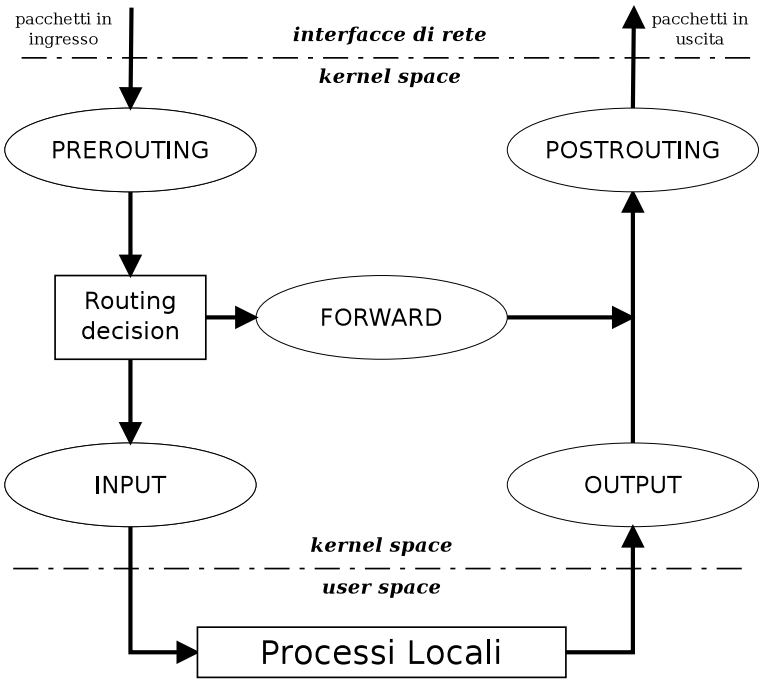


Figura 3.3: Lo schema del *netfilter* di Linux.

Vediamo allora in maggior dettaglio il funzionamento del meccanismo illustrato in fig. 3.3; i pacchetti in arrivo da una interfaccia di rete, così come emergono dal relativo dispositivo (fisico o virtuale che sia), arrivano, dopo aver passato dei controlli elementari di integrità (pacchetti troncati, corrotti, o con il valore delle *checksum* di IP non corrispondente vengono scartati a questo livello, prima di entrare nel *netfilter*) al primo punto di aggancio, indicato nello schema di fig. 3.3 con l'ovale `PREROUTING`.

Una volta attraversato il `PREROUTING` i pacchetti vengono passati al codice di gestione dell'instradamento (indicato in fig. 3.3 dal riquadro marcato *routing decision*), che deve decidere se il pacchetto è destinato ad un indirizzo locale della macchina o deve essere reinviato verso un'altra rete, passando per un'altra interfaccia. Per questo è sempre sul `PREROUTING` che vengono effettuate alcune di quelle operazioni di *traslazione degli indirizzi* (il cosiddetto NAT, acromino di *Network Address Translation*) che sono una delle caratteristiche più interessanti dei firewall contemporanei.

In particolare è qui, prima che i pacchetti entrino nel meccanismo della *routing decision* che ne stabilisce la destinazione finale, che si devono eseguire le manipolazioni sui pacchetti che permettono di cambiarla con il cosiddetto DNAT (acronimo di *Destination Network Address Translation*). Si tratta in questo caso di modificare l'indirizzo di destinazione dei pacchetti in arrivo, in modo da cambiare, in maniera trasparente per chi lo ha iniziato, la destinazione di un certo flusso di dati.[3]

[3]questo comporta, grazie all'uso del *connection tracking* che vedremo in sez. 3.2.3, anche la corrispondente

Se il pacchetto è destinato ad un indirizzo locale esso proseguirà il suo percorso nel *netfilter* attraversando un secondo punto di aggancio, indicato in fig. 3.3 con l'ovale `INPUT`, prima di poter essere recapitato al processo a cui è diretto, se esso esiste, o generare un opportuno messaggio di errore, qualora questo sia previsto.[4] È sull'`INPUT` che in genere si esegue il filtraggio dei pacchetti destinati alla nostra macchina.

Se invece il pacchetto deve essere reinviato ad un indirizzo su una rete accessibile attraverso un'altra interfaccia dovrà prima attraversare un secondo punto di aggancio, indicato in fig. 3.3 con l'ovale `FORWARD`, ed infine l'ultimo punto, indicato con l'ovale `POSTROUTING`, prima di arrivare sull'interfaccia di uscita.

La ragione di questo doppio passaggio è che il filtraggio dei pacchetti che attraversano soltanto la nostra macchina per un'altra destinazione viene effettuato nel `FORWARD`, dato che attraverso il `POSTROUTING` oltre ai pacchetti in transito dalla nostra macchina, possono passare anche i pacchetti generati direttamente su di essa.

Infatti, tutti i pacchetti generati localmente, compresi anche i pacchetti generati in risposta a quelli ricevuti in ingresso attraverso l'`INPUT`, devono passare per un altro punto di aggancio, indicato in fig. 3.3 con l'ovale `OUTPUT`, prima di raggiungere anch'essi il `POSTROUTING`. Pertanto è su `OUTPUT` che è possibile filtrare quanto viene inviato verso l'esterno a partire da processi locali o direttamente dal kernel (ad esempi messaggi di errore) in risposta a pacchetti arrivati in ingresso.

Infine è sul `POSTROUTING` che si possono compiere altre operazioni di NAT, ed in particolare le manipolazioni sugli indirizzi sorgente dei pacchetti per il cosiddetto SNAT (acronimo di *Source Network Address Translation*) in cui si modifica l'indirizzo sorgente dei pacchetti in uscita in modo da poter cambiare, in maniera trasparente per chi lo riceve, l'origine di un certo flusso di dati.[5] In questo modo ad esempio si effettua il cosiddetto *masquerading*, in cui si fa figurare un router/firewall come origine delle connessioni, anche se queste originano da computer posti in una rete interna.

3.2.2 Il sistema di *IP Tables*

Lo schema illustrato in sez. 3.2.1 riguarda la struttura di basso livello del kernel, su di esso però è stato costruita tutta una infrastruttura di più alto livello per la gestione dei pacchetti chiamata *IP Tables*, che è quella che consente di far eseguire delle operazioni sui pacchetti nei cinque punti di aggancio del *netfilter* illustrati in precedenza, senza doversi scrivere un modulo del kernel.

Questa infrastruttura è basata sulla presenza varie "*tabelle*", ciascuna delle quali è dedicata ad ospitare le operazioni relative ad un compito specifico. All'interno delle tabelle sono definite delle *catene predefinite*, che corrispondono agli effettivi punti di aggancio del *netfilter* di fig. 3.3 su cui arrivano i pacchetti, e che hanno lo stesso nome per tutte le tabelle.

Le tabelle non sono altro che dei contenitori per le "*catene*" che a loro volta sono semplicemente dei contenitori per le regole di filtraggio. Una regola è costituita da un criterio di

modifica degli indirizzi sorgente delle relative risposte, che però viene effettuata automaticamente sui pacchetti in uscita senza dover scrivere ulteriori regole.

[4] si tenga presente che un tale messaggio, qualora emesso, seguirà lo stesso percorso di tutti i pacchetti che escono dalla nostra macchina; non ha alcun senso pensare che possa tornare indietro secondo il percorso fatto dal pacchetto di andata.

[5] ed in questo caso grazie al *connection tracking*, verranno modificati automaticamente gli indirizzi di destinazione dei pacchetti in ingresso ottenuti come risposta.

selezione (detto *match*) che permette di scegliere i pacchetti in base alle loro caratteristiche, e da un criterio di destinazione (detto *target*) che permette di stabilire il destino dei pacchetti che soddisfano il criterio di selezione.

I pacchetti che attraversano una catena vengono sottoposti in sequenza a ciascuna delle regole che essa contiene, e se corrispondono al criterio di selezione viene applicato loro il criterio di destinazione scelta. Per semplificare la gestione il sistema permette di creare, oltre a quelle predefinite, su cui comunque transitano i pacchetti, delle nuove catene, da identificare con un nome fornito dall'utente, su cui inserire un qualunque insieme di regole che si vogliono raggruppare.

Una nuova catena così definita può costituire la *destinazione* (il *target*) di una regola e tutti i pacchetti che corrispondono al criterio di selezione di detta regola verranno inviati sulla nuova catena e con questo sottoposti alla serie di regole ivi definite. In questo modo una catena definita dall'utente viene a costituire una specie di "*subroutine*" che permette di trattare gruppi di pacchetti con lo stesso insieme di regole in maniera compatta.

Ciascuna delle tabelle su cui sono presenti le varie catene viene realizzata attraverso un opportuno modulo del kernel. Ognuna di esse prevede la presenza di una o più catene predefinite, a seconda dello scopo della tabella. Al momento esistono solo quattro tabelle:

filter è la tabella principale, quella utilizzata, come indica il nome, per le regole per il filtraggio dei pacchetti. Benché in teoria sia possibile filtrare i pacchetti anche sulle altre tabelle, essa costituisce il luogo naturale su cui operare per la creazione di firewall. Contiene tre catene predefinite: `INPUT`, su cui filtrare i pacchetti in ingresso sulla macchina stessa; `FORWARD`, su cui filtrare i pacchetti che vengono instradati attraverso la macchina da una interfaccia ad un'altra, e `OUTPUT` su cui filtrare i pacchetti generati localmente che escono verso l'esterno.

nat questa tabella viene utilizzata, come suggerisce il nome, per inserire le regole relativo al *Network Address Translation* (sia SNAT che DNAT) che permettono di manipolare gli indirizzi. Ha la caratteristica che solo il primo pacchetto di una connessione raggiunge questa tabella, mentre a tutti i restanti pacchetti nella stessa connessione verrà automaticamente applicata la stessa regola presa per il primo pacchetto. L'uso di questa tabella presuppone quindi l'impiego del sistema del *conntrack* che vedremo in sez. 3.2.3. La tabella contiene tre catene: `PREROUTING`, per modificare gli indirizzi dei pacchetti che arrivano sulla macchina (in genere usata per alterarne la destinazione); `OUTPUT`, per modificare i pacchetti generati localmente prima che essi arrivano all'instradamento finale; `POSTROUTING`, per modificare i pacchetti immediatamente prima che questi escano dall'interfaccia (in genere usata per alterarne l'indirizzo sorgente).

mangle questa tabella viene utilizzata, come suggerisce il nome, per manipolazioni speciali dei pacchetti; è assolutamente sconsigliato usarla per filtrare i pacchetti o alterarne gli indirizzi. Fino al kernel 2.4.17 essa conteneva solo due catene: `PREROUTING`, per modificare i pacchetti prima dell'instradamento; `OUTPUT`, per modificare i pacchetti generati localmente prima dell'instradamento. A partire dal kernel 2.4.18 sono state inserite anche le tre altre catene: `INPUT`, su cui modificare i pacchetti in ingresso sulla macchina locale; `FORWARD`, su cui modificare i pacchetti che vengono instradati

attraverso la macchina da una interfaccia ad un altra; `POSTROUTING`, per modificare i pacchetti immediatamente prima che questi escano dall'interfaccia.

`raw` una tabella speciale usata per gestire le eccezioni alle regole del sistema del *connection tracking* (che vedremo in sez. 3.2.3), principalmente per selezionare i pacchetti per cui disabilitarlo con il target speciale `NOTRACK`. E' posta al livello più alto e viene utilizzata prima di tutte le altre, e prima che vengano eseguite le operazioni del *connection tracking* a cui tutte le altre sono invece sottoposte. Contiene solo le catene `PREROUTING` ed `OUTPUT`.

In sez. 3.2.1 abbiamo visto come i pacchetti attraversano il *netfilter* all'interno del kernel, ed in precedenza abbiamo anche accennato a come le varie catene predefinite delle tabelle di *IP Tables* corrispondano in sostanza ai cinque punti di aggancio all'interno del *netfilter*; la manipolazione dei pacchetti però viene eseguita attraverso le regole inserite nelle omonime catene presenti nelle varie tabelle.

Per questo con `iptables` il diagramma di fig. 3.3 deve essere rivisto per illustrare l'ordine in cui i pacchetti attraversano le catene predefinite all'interno delle varie tabelle, dato che da quest'ordine dipende il risultato finale dell'azione delle varie regole che si sono inserite. In fig. 3.4 si è allora riportato il nuovo schema che illustra l'ordine di attraversamento dei pacchetti delle varie catene nelle varie tabelle.

Si noti come la tabella `raw` abbia un ruolo speciale, rimanendo esterna al sistema *connection tracking* che invece è attivo per tutte le altre. Siccome l'uso della tabella attiene ai pacchetti che *entrano* nel *netfilter* essa prevede solo le catene di `PREROUTING` ed `OUTPUT` che sono i due soli punti di ingresso dello stesso. In entrambi i casi le regole della tabella saranno processate per prime e prima dell'attivazione del *connection tracking*.

Se un pacchetto è destinato ad un processo locale questo passerà prima per le catene di `PREROUTING` prima nella tabella di `mangle` e poi nella tabella di `nat`, poi passerà per il meccanismo di decisione dell'instradamento che invierà il pacchetto alle catene di `INPUT`, prima nella tabella di `mangle` e poi in quella di `filter`, per poi essere ricevuto dal processo locale cui è destinato.

Se invece il pacchetto è in transito attraverso la nostra macchina per essere reinviato su un'altra interfaccia di nuovo passerà per le catene di `PREROUTING` prima nella tabella di `mangle` e poi nella tabella di `nat`, ma dopo la decisione sull'instradamento proseguirà sulle catene di `FORWARD`, prima della tabella di `mangle` e poi di quella di `filter`, infine proseguirà verso l'interfaccia di uscita passando per le catene di `POSTROUTING`, prima nella tabella di `mangle` e poi in quella di `nat`.

Infine se un pacchetto è generato da un processo locale per essere inviato all'esterno prima passerà per le catene di `OUTPUT`, a partire dalla tabella di `mangle`, per proseguire in quella di `nat` ed infine in quella di `filter`, da qui poi passerà alle catene di `POSTROUTING` nello stesso ordine usato dai pacchetti in transito.

3.2.3 Il meccanismo del *connection tracking*

Una delle principali caratteristiche del *netfilter* di Linux è stata quella di aver introdotto una infrastruttura complessa, ma estremamente flessibile, chiamata *connection tracking* (in breve *conntrack*), che permette di analizzare il flusso dei pacchetti, ricostruire lo stato delle connessioni

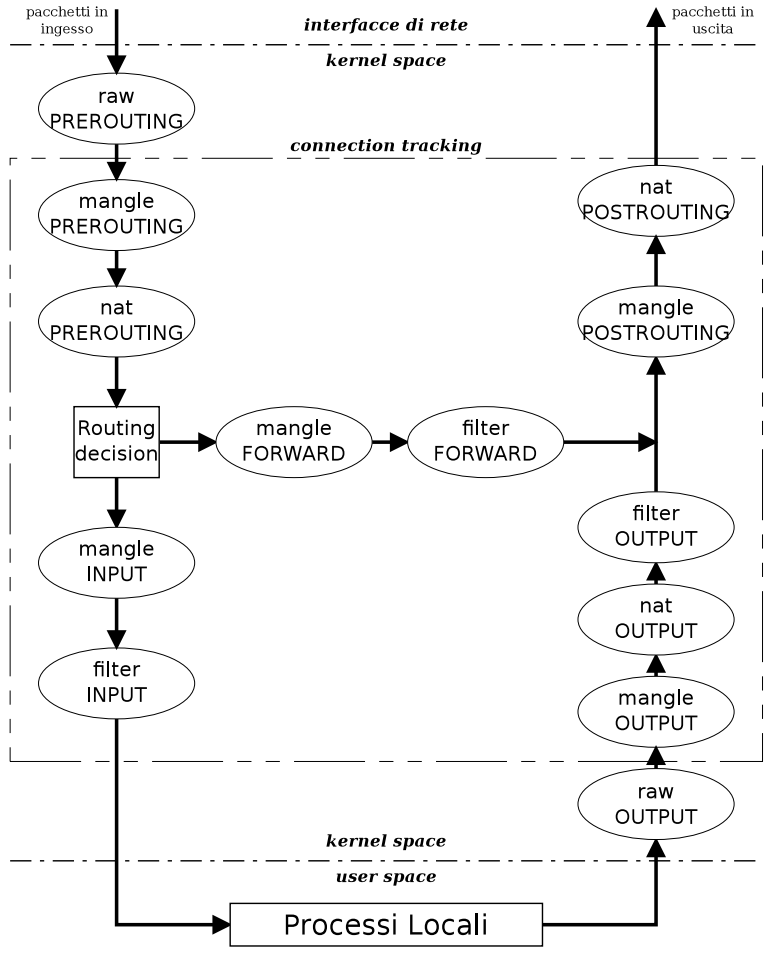


Figura 3.4: Lo schema del flussi dei pacchetti attraverso le tabelle e le catene predefinite di `iptables`.

presenti, ricondurre ad esse e classificare i vari pacchetti. É grazie a questo meccanismo che è possibile la realizzazione del filtro degli stati illustrato in sez. 3.2.4.

L'infrastruttura del *connection tracking* va ben oltre l'uso fattone dal filtro degli stati per realizzare un firewall *stateful*; essa infatti è anche la base su cui è costruito il funzionamento delle capacità di NAT che permettono al *netfilter* di modificare al volo gli indirizzi dei pacchetti, e per poterli redirigere opportunamente, in modo da gestire compiti come il *masquerading*[6] o il *transparent proxing*.[7]

Lo scopo principale del motore del *conntrack* è quello di analizzare il traffico per identificare

[6] si chiama così l'uso dello SNAT per *mascherare* una (o più) reti dietro un singolo indirizzo IP (quello usato in genere dal firewall per uscire su internet).

[7] si chiama così l'uso del DNAT per redirigere, in maniera trasparente per i client, il traffico destinato a certi servizi (il caso classico è il web) su una macchina che faccia da *proxy*.

ogni connessione presente, determinarne lo stato, e poi classificare i pacchetti associandoli a ciascuna di esse. Dato che il concetto di *stato* e di *connessione* è definito nativamente solo per alcuni protocolli (in sostanza solo per il TCP), il meccanismo è stato esteso per permettere di identificare e classificare ogni flusso di dati anche per gli altri protocolli che non supportano le connessioni, come UDP e ICMP. Ad esempio come vedremo più avanti si potrà definire una forma di connessione anche per UDP, identificando univocamente un flusso di pacchetti in base ai relativi indirizzi e porte di origine e destinazione.

L'infrastruttura del *conntrack* è stata introdotta nei kernel della serie 2.4; nei kernel precedenti non esisteva nessun meccanismo che potesse eseguire questo compito e pertanto non era possibile creare dei firewall stateful. Inizialmente disponibile solo per il protocollo IPv4 è stato esteso anche ad IPv6. L'uso del sistema ha però portato anche ad un cambiamento delle regole che gestiscono la frammentazione dei pacchetti IP.

La frammentazione è una caratteristica di IPv4 che prevede che i pacchetti che debbano essere inoltrati su una linea la cui MTU[8] eccede le dimensioni del pacchetto vengano "*frammentati*" dal router che li riceve per poter essere trasmessi sulla linea stessa. Questo significa che un singolo pacchetto IP verrà spezzato in più parti, che poi verranno riassemblate solo alla destinazione finale, con la ricezione di tutti i pezzi. Nei kernel precedenti la serie 2.4.x c'era la possibilità di attivare o meno la deframmentazione dei pacchetti prima di procedere al loro filtraggio, attraverso l'opzione `ip_always_defrag`. Dato che il *connection tracking* non può funzionare con i pacchetti frammentati, questa opzione non esiste più, il codice per la deframmentazione è stato incluso nel *conntrack* ed i pacchetti vengono sempre riassemblati prima di tracciarne lo stato.

Tutte le operazioni relative al *connection tracking* vengono eseguite nella catena di `PREROUTING` eccetto per i pacchetti che vengono generati localmente, che invece sono analizzati nella catena di `OUTPUT`. Questo significa che tutte le operazioni per il calcolo degli stati e la classificazione dei pacchetti vengono eseguite non appena questi arrivano alla catena di `PREROUTING`. La sola eccezione è per i pacchetti generati localmente che creano una nuova connessione, che vengono identificati su `OUTPUT`, ma non appena al pacchetto inviato per la creazione di una connessione arriva una risposta la successiva classificazione dei pacchetti avviene comunque sulla catena di `PREROUTING`. Come accennato l'unico modo di impostare delle regole con `iptables` prima che il *connection tracking* entri in azione è quello di usare la speciale tabella di `raw`.

Il *conntrack* mantiene nel kernel una tabella con tutte le connessioni presenti ed il relativo stato; la dimensione della tabella, e pertanto il limite massimo al numero di connessioni contemporanee che possono esistere, è un parametro configurabile che può essere impostato usando l'interfaccia del filesystem `proc` attraverso il file `/proc/sys/net/nf_conntrack_max`. Il valore di default è calcolato automaticamente sulla base della memoria disponibile, e nel caso di una macchina con 8Gb di RAM otteniamo:

```
# cat /proc/sys/net/nf_conntrack_max
65536
```

detto valore può essere comunque modificato al volo scrivendo un diverso valore nel file.

Si può inoltre accedere direttamente al contenuto della tabella delle connessioni attraverso il filesystem `proc`, usando il file `/proc/net/ip_conntrack` se interessa solo IPv4 oppure il file `/proc/net/nf_conntrack` per tutti i protocolli del livello di rete. Nel seguito esamineremo solo

[8] la *Maximum Transfer Unit* o MTU è la dimensione massima di un pacchetto di dati che può essere trasferita su un segmento fisico di una rete.

il caso di IPv4, che resta comunque l'ambito di maggiore applicazione del sistema, e faremo riferimento al primo dei due file, che manca nei dati del campo della informazione iniziale relativa al protocollo di rete cui fa riferimento la voce (in genere `ipv4` o `ipv6`).

Esaminando `/proc/net/ip_conntrack` si otterranno le informazioni mantenute dal kernel relative a ciascuna connessione; ogni riga corrisponde ad una connessione, un esempio di voce contenuta nel file è:

```
tcp      6 119 SYN_SENT src=192.168.1.1 dst=192.168.1.141 sport=35552 \
      dport=22 [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 sport=22 \
      dport=35552 use=1
```

Il primo campo è sempre quello del protocollo, espresso per nome, e ripetuto per numero (in esadecimale, anche se nell'esempio non lo si nota) nel secondo campo. Il terzo campo indica il tempo di vita restante per la voce nella tabella, nel caso 119 secondi. Il campo viene decrementato progressivamente fintanto che non si riceve traffico sulla connessione, nel qual caso viene riportato al valore predefinito (che varia a seconda del protocollo).

I primi tre campi sono sempre questi, i campi successivi dipendono dal protocollo, nel caso specifico il quarto campo, avendo a che fare con il TCP, indica lo stato della connessione, che è `SYN_SENT`: si è cioè inviato un segmento SYN in una direzione. Seguono i campi con indirizzo sorgente e destinazione, e porta sorgente e destinazione. Segue poi una parola chiave `[UNREPLIED]` che ci informa che non si è ricevuta risposta, e gli indirizzi sorgente e destinazione, e le porte sorgente e destinazione per i pacchetti appartenenti al traffico di ritorno di questa connessione; si noti infatti come rispetto ai precedenti i numeri siano gli stessi ma sorgente e destinazione risultino invertiti.

Vediamo allora il funzionamento del *conntrack* per i tre principali protocolli, cominciando con il più complesso, il TCP. In questo caso infatti il protocollo già prevede l'esistenza degli stati, e come vedremo questo comporterà un certo livello di confusione, in quanto a volte lo stato del protocollo non coincide con quello del *connection tracking*.

La caratteristica principale del TCP è che esso prevede la creazione di una connessione tramite l'invio di tre pacchetti separati, con un procedimento chiamato *three way handshake*, illustrato in fig. 3.5, dove si è riportato lo scambio di pacchetti ed i relativi cambiamenti di stato del protocollo su client e server.

Supponiamo allora di essere sul client; dal punto di vista del *conntrack* una volta che si invia un pacchetto SYN questo passerà per la catena di `OUTPUT` e verrà riconosciuto come un pacchetto che inizia una nuova connessione, creando una voce nella tabella delle connessioni simile a quella mostrata in precedenza, con lo stato `SYN_SENT`[9] ed una connessione marcata come `UNREPLIED`.

Il primo passaggio di stato si ha una volta che si riceve la risposta dal server con il segmento SYN+ACK ricevuto il quale la connessione si porta immediatamente nello stato `ESTABLISHED`, ed avremo una voce del tipo:

```
tcp      6 431996 ESTABLISHED src=192.168.1.1 dst=192.168.1.141 sport=38469 \
      dport=22 src=192.168.1.141 dst=192.168.1.1 sport=22 \
      dport=38469 [ASSURED] use=1
```

[9] si può ottenere questa situazione bloccando sul server la ricezione dei segmenti SYN, in questo modo il client non riceva da questo nessuna risposta, così che lo stato nella tabella delle connessioni resti bloccato.

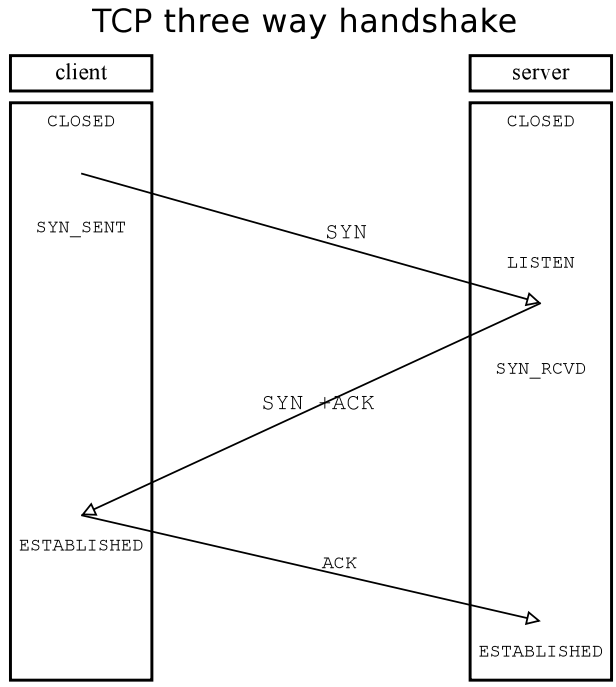


Figura 3.5: Lo schema del *three way handshake* del TCP.

dove il flag che indica la condizione di `[UNREPLIED]` non c'è più ed invece viene attivato quello che indica la condizione di `[ASSURED]`; questa ci dice che è stato osservato del traffico in entrambe le direzioni, e che la voce nella tabella delle connessione non deve essere cancellata anche se questa si riempie, cosa che invece avviene per le voci che non hanno tale flag attivo.

Si tenga presente che questo accade anche se blocchiamo a questo punto il *three way handshake*,[10] interrompendo la creazione della connessione TCP; questa è la sequenza normale per gli stati del TCP, che è anche illustrata in fig. 3.5, e la si può verificare usando `netstat` sul server, dove la connessione sarà riportata essere rimasta nello stato `SYN_RECV`.

La prima differenza fra lo stato del protocollo e quello della connessione all'interno del sistema del *connection tracking* emerge a questo punto: lo stato del protocollo infatti è modificato alla ricezione dei pacchetti (cioè sulla catena di `INPUT`), mentre lo stato della connessione è, come abbiamo già detto, modificato sulla catena di `PREROUTING`. Se si blocca con il firewall la ricezione del pacchetto SYN+ACK, allora il risultato può essere diverso a seconda di come si esegue detto blocco. Se infatti si blocca il pacchetto sulla catena di `INPUT` questo passa comunque per la catena di `PREROUTING`, ed esattamente come prima la connessione viene riconosciuta come `ESTABLISHED`.

Se invece si blocca il pacchetto SYN+ACK direttamente sulla catena di `PREROUTING`[11] la

[10] ad esempio impedendo con una apposita regola nella catena di `OUTPUT` che il segmento ACK di conclusione del procedimento venga inviato al server.

[11] per farlo occorre usare la tabella di `mangle`, bloccando il pacchetto su di essa, dato che la tabella di `nat` opera solo sul primo pacchetto di una connessione.

ricezione dell'ACK non viene riconosciuta e la conclusione della connessione non viene stabilita, viene però rilevato l'arrivo di un segmento SYN, e la connessione nella tabella degli stati viene portata nello stato `SYN_RECV`, mentre la condizione di `UNREPLIED` sparisce, avremo cioè una voce del tipo:

```
tcp      6 44 SYN_RECV src=192.168.1.1 dst=192.168.1.141 sport=35552 \
      dport=22 src=192.168.1.141 dst=192.168.1.1 sport=22 dport=35552 use=1
```

Questa condizione si incontra anche quando si riceve un pacchetto SYN di una connessione diretta verso di noi, in tal caso la voce viene creata alla ricezione di un pacchetto SYN,[12] con una voce del tipo:

```
tcp      6 55 SYN_RECV src=192.168.1.141 dst=192.168.1.1 sport=33616 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 dport=33616 use=1
```

e di nuovo, non appena si emette il segmento di risposta con un SYN+ACK, la connessione andrà in stato `ESTABLISHED`, con una voce del tipo:

```
tcp      6 431998 ESTABLISHED src=192.168.1.141 dst=192.168.1.1 sport=33617 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 \
      dport=33617 [ASSURED] use=1
```

e ci resterà anche se non si conclude il *three way handshake*,[13] questo perché nel caso del protocollo TCP lo stabilirsi di una connessione è identificato dal *conntrack* semplicemente dal fatto che c'è uno scambio di pacchetti TCP nelle due direzioni.

La chiusura di una connessione TCP avviene con lo scambio di 4 pacchetti, come illustrato in fig. 3.6, e non viene mai completamente chiusa fino a che questo non viene completato. Una volta che questa è completata la connessione entra nello stato `TIME_WAIT`, con una voce del tipo:

```
tcp      6 9 TIME_WAIT src=192.168.1.141 dst=192.168.1.1 sport=33614 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 \
      dport=33614 [ASSURED] use=1
```

che ha una durata di default di due minuti, così che tutti gli eventuali ulteriori pacchetti residui della connessione che possono arrivare saranno classificati all'interno della stessa ed essere sottoposti alle stesse regole.

Lo schema di chiusura mostrato in fig. 3.6 non è l'unico possibile, una connessione infatti può anche essere terminata da un pacchetto RST,[14] che la porta in stato di `CLOSE`, che resta attivo per 10 secondi. Si tenga presente che in risposta ad un pacchetto RST non deve essere dato nessun ricevuto e che questi pacchetti sono generati direttamente dal kernel ed utilizzati per chiudere immediatamente una connessione.

Gli stati possibili per una connessione TCP sono tutti quelli definiti per il protocollo (anche se la corrispondenza non è proprio immediata, come abbiamo appena visto). Per ciascuno

[12] si può verificare il comportamento bloccando la risposta del SYN+ACK sulla catena di `OUTPUT`.

[13] cosa che si può ottenere ad esempio bloccando con il firewall la ricezione dell'ultimo ACK di risposta.

[14] il protocollo TCP prevede infatti che in risposta a pacchetti inviati ad una porta su cui non è in ascolto nessun servizio, o in generale non ricollegabili ad una connessione debba essere risposto con l'emissione di un pacchetto RST, la ricezione del quale deve produrre la terminazione immediata di ogni eventuale connessione.

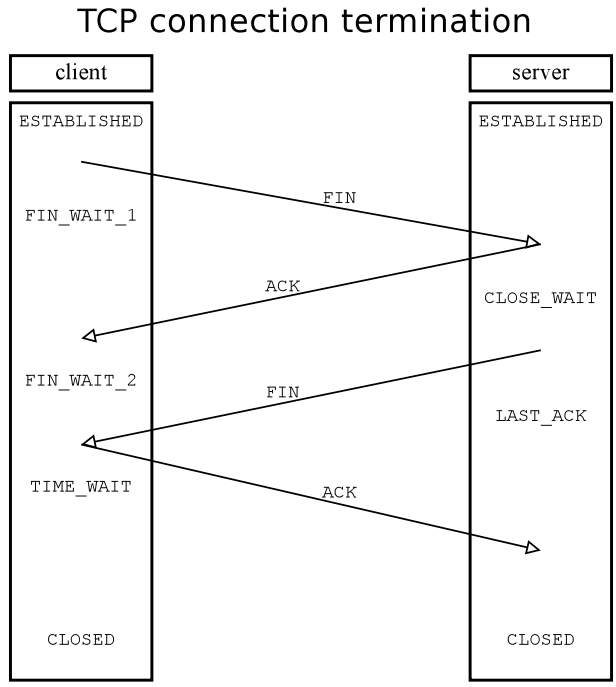


Figura 3.6: Lo schema della conclusione di una connessione TCP.

di questi il *conntrack* definisce un opportuno valore di timeout scaduto il quale la voce viene rimossa dalla tabella delle connessioni. I valori di default per il timeout (che vengono impostati quando la connessione si porta in quello stato, e reimpostati quando si ricevono nuovi pacchetti sulla connessione) sono definiti nel file `net/netfilter/nf_conntrack_proto_tcp.c` nei sorgenti del kernel e riportati in tab. 3.1.[15]

Stato	**Valore**
SYN_SENT	2 minuti.
SYN_RECV	60 secondi.
ESTABLISHED	5 giorni.
FIN_WAIT	2 minuti.
CLOSE_WAIT	60 secondi.
LAST_ACK	30 secondi.
TIME_WAIT	2 minuti.
CLOSE	10 secondi.
SYN_SENT2	2 minuti.

Tabella 3.1: Valori dei tempi di timeout delle connessioni TCP.

Al contrario di TCP, UDP è un protocollo che non supporta le connessioni, il protocollo si limita ad inviare i pacchetti, questi possono arrivare in qualunque ordine, essere perduti o

[15] la tabella è aggiornata ai valori usati dal kernel 3.1.

duplicati, e non ci sono procedimenti per definire l'inizio o la conclusione di una comunicazione. Nonostante questo sia possibile è comunque possibile cercare di classificare il traffico ed inserirlo in una logica di connessioni, anche se in questo caso (come sarà per ICMP) la classificazione non può che essere basata su una euristica dello scambio dei pacchetti.

L'algoritmo di classificazione per UDP si basa sul riconoscimento del traffico dalla stessa coppia di indirizzi IP e porte (destinazione e sorgente) sui due capi della comunicazione. Si considerano cioè parte di una connessione tutti i pacchetti UDP che partono da un nostro IP e hanno una data porta sorgente diretti verso un altro IP ed una data porta di destinazione, e quelli che provengono da detto IP con sorgente la porta usata da noi come destinazione e come destinazione la nostra porta sorgente. Si fa l'assunzione cioè che pacchetti con queste caratteristiche siano emessi e ricevuti in corrispondenza allo scambio di dati fra i due capi di uno stesso socket UDP.

Si tenga presente che questo criterio non assicura che una connessione così identificata comprenda solo i pacchetti appartenenti ad un unico flusso di dati, è infatti possibile, anche se non molto probabile, dato che in genere le porte non vengono immediatamente riusate, una situazione in cui un client si collega ad un nostro server UDP usando una certa porta, conclude le sue operazioni, e poi un client diverso ritenta la stessa connessione usando la stessa porta. In un caso come questo i pacchetti saranno classificati, se la voce nella tabella del *conntrack* non è ancora scaduta, nella stessa connessione, anche se appartengono a due flussi di dati completamente distinti.

Questo meccanismo di classificazione comporta che se si invia un pacchetto UDP che non corrisponde a nessuna voce già presente nella tabella delle connessioni, questo verrà considerato come un pacchetto iniziale e creerà una nuova voce,[16] nella forma:

```
udp      17 28 src=192.168.1.1 dst=192.168.1.141 sport=32769 \
      dport=13 [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 sport=13 \
      dport=32769 use=1
```

Si noti che la voce è analoga a quelle relative alle connessioni TCP, con il nuovo nome (e relativo numero) del protocollo nei primi due campi, ed il tempo di vita restante alla voce nel terzo. Il resto della voce è analogo a quelle usate per il TCP, però non esiste più nel quarto campo un valore dello stato e vengono riportati di seguito direttamente indirizzo sorgente e destinazione e porta sorgente e destinazione. Segue, non avendo ricevuto nessun altro pacchetto il flag della condizione `[UNREPLIED]` e indirizzi e porte (ovviamente invertite) dei pacchetti che si aspettano come risposta.

Nel caso di UDP, quando viene creata con l'invio di un nuovo pacchetto, a ciascuna voce viene dato un tempo di vita di 30 secondi, dei quali, nell'esempio precedente, ne sono restati solo 28 disponibili. Non appena si riceve un pacchetto di risposta sulla coppia di IP corrispondente e porta la condizione di `[UNREPLIED]` sparirà, e la connessione si potrà considerare stabilita, la voce nella tabella delle connessioni diventerà del tipo:

```
udp      17 25 src=192.168.1.1 dst=192.168.1.141 sport=32768 \
      dport=13 src=192.168.1.141 dst=192.168.1.1 sport=13 dport=32768 use=1
```

[16] un modo per ottenere questo è bloccare con `iptables` sulla macchina di destinazione i pacchetti di risposta al collegamento via UDP.

ed il tempo di vita verrà riportato di nuovo a 30 secondi, facendo ripartire il conto alla rovescia.

Si noti però che ancora la connessione non è considerata stabile, lo diverrà non appena sarà osservata una certa quantità di traffico, al che comparirà il flag di `[ASSURED]` e la voce diverrà:

```
udp      17 177 src=192.168.1.1 dst=192.168.1.141 sport=32768 \
      dport=13 src=192.168.1.141 dst=192.168.1.1 sport=13
      dport=32768 [ASSURED] use=1
```

ed il tempo di vita viene portato ad 180 secondi (nell'esempio ne sono passati tre).

Dato che anche il protocollo ICMP non supporta connessioni, anche questo viene classificato in base ad una euristica dei pacchetti. Inoltre questo protocollo viene usato principalmente per l'invio di messaggi di controllo relativi ad errori della rete, e pertanto non è neanche classificabile in una logica di connessioni e stati. Esistono però alcuni specifici pacchetti ICMP che richiedono una risposta, ed allora di nuovo diventa possibile classificare queste risposte in una logica di connessione.

Per questo motivo il sistema del *conntrack* classifica soltanto quattro coppie di *tipi*[17] di pacchetti ICMP, *echo request* e *echo reply*, *netmask request* e *netmask reply*, *timestamp request* e *timestamp reply*, *information request* e *information reply*. In questo caso l'euristica di classificazione delle connessioni è identica a quella di UDP, dove al posto della porta viene usato il tipo di pacchetto, che deve corrispondere alla precedente richiesta. Ad esempio pacchetti di *echo reply* provenienti da un IP a cui si era inviati un *echo request* vengono considerati facenti parte di una connessione.

Così se effettuiamo un `ping` verso una macchina all'invio del pacchetto *echo request* nella tabella del *conntrack* apparirà una voce del tipo:[18]

```
icmp     1 22 src=192.168.1.1 dst=192.168.1.141 type=8 code=0 id=29957 \
    [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 type=0 code=0 id=29957 use=1
```

che ha un tempo di vita di 30 secondi. Al solito la voce riporta indirizzi IP sorgente e di destinazione del pacchetto e dell'eventuale risposta attesa. In questo caso viene anche riportato sia il tipo di pacchetto ICMP che ha creato la voce (8, essendo un *echo request*) seguita poi da quello (0, essendo un *echo reply*) che ci si attende come risposta, mentre i valori del campi `code` e `id` degli stessi.

Nel caso di ICMP non si ha invece nessuna voce relativa ad una connessione stabilita, infatti la ricezione del pacchetto di risposta esaurisce tutto il traffico previsto per questo tipo di operazioni, per cui la connessione viene immediatamente chiusa, e la voce nella tabella del *conntrack* cancellata.

3.2.4 Il funzionamento del filtro degli stati

Una delle principali innovazioni portate dal *netfilter* con la serie dei kernel 2.4.x è stata quella della possibilità di usare Linux come firewall *stateful*. Come spiegato in sez. 3.1.2 un firewall *stateful* è un firewall in grado di classificare i pacchetti in un insieme di *stati*, per poi selezionarli in base a questi ultimi.

[17] i pacchetti ICMP vengono classificati in base al valore di un apposito campo nella loro intestazione, il *tipo*, per maggiori informazioni si può fare riferimento a quanto illustrato in sez. 3.4.4 e a fig. 3.8.

[18] al solito la si può ottenere bloccando i pacchetti di risposta provenienti dalla macchina che si sta "*pingando*".

Nel caso di Linux questo viene realizzato attraverso l'infrastruttura del *connection tracking* descritto in sez. 3.2.3. Benché le modalità in cui vengono classificati i pacchetti all'interno del *connection tracking* siano abbastanza complesse, il filtro degli stati di Linux riclassifica tutti i pacchetti in solo 4 stati, che si sono riportati in tab. 3.2. Nella tabella si è menzionato anche `UNTRACKED` che in realtà non è uno stato ma identifica semplicemente i pacchetti che si sono esclusi dal *connection tracking*, e quindi anche dalla classificazione degli stati, tramite il target `NOTRAK` nella catena di `raw`.

Stato	Descrizione
`NEW`	Identifica i pacchetti usati per dare origine ad una connessione.
`ESTABLISHED`	Identifica i pacchetti riconosciuti far parte o essere associati ad una connessione già stabilita.
`RELATED`	Identifica i pacchetti che danno origine ad una nuova connessione che è però correlata ad una connessione esistente.
`INVALID`	Identifica i pacchetti che non vengono associati a nessuna connessione esistente o hanno dati o intestazioni riconosciuti come non validi.
`UNTRACKED`	Identifica i pacchetti che si sono forzatamente esclusi dal *connection tracking* e con questo anche dalla classificazione negli stati tramite il target `NOTRAK` nella tabella di `raw`.

Tabella 3.2: Gli stati delle connessioni definiti nel *netfilter*.

Come per la classificazione delle connessioni anche quella degli stati viene eseguita nella catena di `PREROUTING`, con l'eccezione dei pacchetti creati localmente che vengono analizzati su `OUTPUT`. Quando un pacchetto arriva sulla catena di `PREROUTING` prima viene eseguito un eventuale DNAT e le manipolazioni previste da eventuali regole nella tabella di `mangle`, infine viene calcolato lo stato. Lo stesso avviene per i pacchetti uscenti dalla catena di `OUTPUT`.

Quando si incontra un pacchetto che non si è mai visto prima e che può essere considerato come pacchetto di inizio di una nuova connessione, questo verrà inserito nello stato `NEW`. Il caso più comune è quello di un pacchetto TCP con il flag SYN, ma può essere anche il primo pacchetto di una sessione su UDP. Si tenga presente inoltre che possono essere classificati in questo stato anche pacchetti TCP diversi da un SYN, il che può comportare problemi in certi casi, ma può essere di aiuto quando ci sono connessioni andate in timeout ma non chiuse.

Se un pacchetto corrisponde ad una voce nella tabella delle connessioni allora verrà classificato come parte di una connessione già stabilita ed assumerà lo stato di `ESTABLISHED`. Perché un pacchetto sia classificato come tale il *conntrack* deve aver rilevato del traffico in entrambe le direzioni, si arriva allo stato `ESTABLISHED` solo dopo che una connessione è passata dallo stato `NEW`, quando arriva un pacchetto interpretabile come risposta. Questo significa che anche dei pacchetti ICMP possono entrare in questo stato, qualora generati come risposta ad un pacchetto che inizia una connessione (come l'*echo reply* in risposta ad un *echo request*).

Lo stato più complesso da realizzare, anche se è molto semplice da descrivere, è `RELATED`; in breve si può dire che se il traffico può essere collegato in qualche modo ad una connessione già stabilita (ad esempio è un pacchetto ICMP relativo ad una connessione TCP, o il traffico dati di una sessione FTP) questo verrà classificato nello stato `RELATED`. Questo significa che per potere avere un pacchetto nello stato `RELATED` deve esistere già una connessione in stato `ESTABLISHED` cui questo è collegato.

La difficoltà sta tutta nel meccanismo con cui *conntrack* è in grado di stabilire in che modo un pacchetto è collegato ad una connessione stabilita. Degli esempi per questo tipo di pacchetti sono le risposte ICMP relative ad una connessione,[19] il traffico dati FTP collegato alla presenza di una connessione di controllo, o le connessioni DCC eseguite tramite IRC. In alcuni di questi casi il meccanismo per rilevare il collegamento può essere molto complesso, e richiedere anche l'analisi (parziale) del contenuto dei dati trasmessi all'interno dei vari protocolli, nel qual caso diventa necessario utilizzare degli opportuni moduli di supporto che estendono le capacità di classificazione del *conntrack*.

Questo è ad esempio quello che serve per poter utilizzare i citati protocolli per FTP e IRC, per i quali nella distribuzione standard del kernel sono previsti due appositi moduli, che quando inclusi forniscono le estensioni necessarie; altri moduli relativi ad altri protocolli (H323, SIP) sono disponibili sia come patch esterne che nei kernel di default.

Infine tutto il traffico che non rientra nelle tre categorie precedenti, che cioè non può essere identificato o a cui non è possibile assegnare uno stato, viene classificato come `INVALID`. Si tenga presente che ciò può avvenire sia perché effettivamente si tratta di pacchetti malformati, o di risposte a connessioni inesistenti, sia perché si è esaurito lo spazio nella tabella degli stati, e non è pertanto più possibile classificare il traffico ulteriore.

Come si vede il meccanismo del filtro sugli stati è piuttosto complesso, anche se semplifica notevolmente il lavoro di gestione di un firewall. Ad esempio la classificazione dei pacchetti consente di bloccare in maniera pulita tutti i pacchetti sospetti classificati nello stato `INVALID`. Inoltre con le combinazioni degli stati `NEW` e `ESTABLISHED` consente di gestire in maniera flessibile la creazione nuove "*connessioni*" anche per protocolli come UDP e ICMP che non supportano nativamente questo concetto, e per i quali sarebbe impossibile, senza la presenza del *conntrack* che traccia il flusso dei pacchetti, identificare flussi di dati collegati fra loro con regole basate esclusivamente sul contenuto delle intestazioni dei singoli protocolli.

Inoltre l'uso dello stato `RELATED` permette di ricostruire collegamenti anche fra pacchetti, connessioni e flussi di dati che, fermandosi al punto di vista dei protocolli di livello inferiore, sarebbero completamente scorrelati fra di loro, rendendo possibile filtrare in maniera semplice su protocolli complessi a livello di applicazione, che non sarebbe possibile controllare con un firewall non *stateful*.

La semplificazione della complessità dei protocolli di rete in quattro stati diversi comporta come sempre la necessità di capire bene cosa succede sotto la superficie, onde evitare effetti indesiderati non previsti. Per questo esamineremo in dettaglio il funzionamento del filtro degli stati per ciascuno dei protocolli principali su cui esso opera.

Nel caso del protocollo TCP il punto più critico sta nella definizione di cosa si intende per un pacchetto in stato `NEW`. Infatti dal punto di vista del protocollo l'unico pacchetto che può iniziare una connessione è quello che ha attivo il flag SYN e disattivi i flag ACK e RST. La classificazione del filtro degli stati in certi casi però classifica come `NEW` anche pacchetti che non hanno questa caratteristica. Questo viene fatto per consentire al traffico di una connessione già stabilita (fatta passare da un altro firewall cui si è subentrati, o autorizzata prima di aver caricato in memoria il *conntrack*) di proseguire.

Questo però non è quello che appunto si intende normalmente, per cui se si vuole che passino strettamente soltanto i pacchetti che danno origine ad una nuova connessione TCP, occorrerà

[19] al contrario di quanto veniva affermato nell'*Iptables HOWTO*, i messaggi di errore relativi ad una connessione non sono classificati allo stato `ESTABLISHED`, ma nello stato `RELATED`.

scartare i pacchetti classificati in stato NEW ma che non hanno il flag SYN opportunamente impostato.

Un altro punto da capire è che dal punto di vista della classificazione dei pacchetti vengono considerati come facenti parte dello stato ESTABLISHED tutti i pacchetti successivi al primo (come abbiamo visto in sez. 3.2.3), anche se ancora dal punto di vista del TCP la connessione non ha concluso il *three way handshake* e non può essere considerata come stabilita. Questo avviene perché deve essere possibile consentire ai pacchetti in stato NEW e ESTABLISHED di uscire dalla nostra macchina ed accettare in risposta solo i pacchetti di stato ESTABLISHED: se si considerasse la risposta del SYN+ACK come facente parte dello stato NEW questo non sarebbe possibile.

Nel caso di UDP il meccanismo è molto più semplice, e si considera in stato NEW il primo pacchetto che non corrisponde ad una voce già presente nella tabella delle connessioni. Come sempre controllo viene fatto sulla tabella di OUTPUT per le connessioni uscenti e su quella di PREROUTING per le connessioni entranti. Ogni pacchetto successivo corrispondente viene considerato essere nello stato ESTABLISHED. Dato che UDP non prevede nessun meccanismo di conclusione della connessione questa viene effettuata tramite la scadenza della relativa voce nella tabella delle connessioni, che avviene dopo 3 minuti di assenza di traffico.

Infine nel caso di ICMP si considerano come facenti parte dello stato NEW i pacchetti di richiesta (cioè *echo request* e affini), mentre si considerano come ESTABLISHED tutti quelli di risposta (cioè *echo reply* e affini). Come già accennato dato che alla ricezione del pacchetto di risposta non può esserci alcun altro traffico legale, la relativa voce nella tabella delle connessioni viene immediatamente cancellata.

I pacchetti ICMP però vengono utilizzati anche per comunicare messaggi di controllo relativi ad errori e problemi della rete. In questo caso allora gran parte di questi pacchetti vengono classificati in stato RELATED facendo riferimento ai pacchetti della connessione in risposta ai quali sono stati creati.[20] Ad esempio due dei pacchetti ICMP più comuni sono *host unreachable* e *network unreachable*, generati dal router che riceve il nostro traffico quando questo si accorge che la macchina di destinazione o un altro router che dà accesso alla rete su cui essa si trova non è più raggiungibile.

3.3 Il comando iptables

Affronteremo in questa sezione la sintassi di uso del comando iptables, e le modalità con cui questo viene impiegato per gestire il funzionamento delle regole utilizzate dal *netfilter* per eseguire i compiti di filtraggio e manipolazione dei pacchetti.

3.3.1 La sintassi generale del comando

Come illustrato in sez. 3.2.2 il *netfilter* mette a disposizione dei punti di aggancio nei quali è possibile intercettare i pacchetti, esaminarli, e decidere cosa farne. Usando l'infrastruttura di *IP Tables* questo si può fare inserendo delle *regole* all'interno delle varie *catene* presenti nelle varie *tabelle*.

[20] questo avviene solo quando i dati riportati nel payload del pacchetto coincidono; in particolare vengono inseriti in questo stato i pacchetti ICMP di tipo: *destination-unreachable*, *source-quench*, *time-exceeded*, *paremeter-problem* e *redirect*.

Il comando che permette di selezionare ed esaminare le tabelle, controllare, creare e cancellare le catene, gestirne le proprietà, definire e creare le varie regole è appunto `iptables`. Data le molteplici funzionalità che fornisce il comando è piuttosto complesso e presenta una grande quantità di opzioni, queste si possono classificare secondo una schematizzazione generale per cui l'invocazione del comando ha la forma:

```
iptables controllo [selezione] [azione]
```

in cui le opzioni sono raggruppate in tre classi generiche, quelle per il controllo di tabelle e catene, quelle per la selezione dei pacchetti, e quelle per la decisione delle azioni da compiere sui pacchetti selezionati. Le parti indicate tra parentesi sono opzionali, nel senso che ci sono sintassi valide del comando in cui non compaiono opzioni di selezione dei pacchetti o azioni da compiere su questi ultimi.

In realtà non è necessario specificare le varie opzioni di `iptables` nell'ordine appena mostrato, né raggruppate nelle tre classi ivi elencate; esse possono essere invocate in qualunque ordine, ma usare questa schematizzazione facilita senz'altro la leggibilità del comando. Dato poi che in genere un firewall si costruisce inserendo tutte le relative regole di filtraggio all'interno del *netfilter* con una serie di chiamate successive ad `iptables` (in genere effettuate all'interno di uno script di shell), scrivere in maniera ordinata facilita la comprensione dello stesso.

Unica eccezione alla struttura appena mostrata è l'invocazione del comando senza opzioni, che si limita a stampare una riga di aiuto che invita ad usare le due opzioni `--help` o `-h`. Usando solo queste ultime è possibile ottenere la stampa a video di una breve sinossi delle opzioni principali. L'unica altra opzione utilizzabile singolarmente è, come per tutti i comandi che seguono lo standard GNU, `--version`, che stampa a video la versione del comando.

3.3.2 Le opzioni per il controllo di tabelle e catene

Cominciamo allora con il prendere in esame le varie opzioni che vengono utilizzate per scegliere su quale tabella o catena si va ad operare, per controllare i contenuti, per impostare le proprietà delle catene predefinite, per gestire la selezione, creazione, cancellazione e modifica delle catene create dall'utente stesso.

La prima opzione di `iptables` è `-t` (o `--table`) che permette di selezionare su quale tabella operare, usando come parametro il nome della stessa come indicato in sez. 3.2.2. Dato che il comando è usato principalmente per la realizzazione di firewall, se non si specifica nessuna tabella viene utilizzata di default la tabella `filter`, se invece si vogliono inserire delle regole in catene presenti sulle altre tabelle occorrerà sempre selezionarle esplicitamente specificandole come parametro per l'opzione `-t`.

Come accennato in sez. 3.2.2 oltre alle catene predefinite si possono creare (con l'opzione `-N`) delle nuove catene definite dall'utente, inoltre `iptables` consente di analizzare i contenuti di una catena, e di leggerne i contatori associati (che indicano il numero di pacchetti che l'hanno attraversata) con l'opzione `-L`. Si possono cancellare le regole di una catena con `-F` o impostarne la politica (solo per quelle predefinite) con `-P`. L'elenco completo delle opzioni per operare sulle catene è riportato in tab. 3.3.

Inoltre usando l'opzione `-v` si può aumentare la quantità di informazione fornita dal comando; l'uso di questa opzione è di particolare importanza quando usata insieme a `-L`, dato che permette di mostrare tutti i dettagli delle regole di selezione comprese quelle sulle interfacce, sul campo

Opzione		Significato
-N	--new-chain	Crea una nuova catena, il cui nome deve essere passato come parametro, e non deve corrispondere ad una catena già esistente.
-X	--delete-chain	Cancella una catena, il cui nome deve essere passato come parametro; la catena deve essere vuota e le catene predefinite non possono essere cancellate.
-E	--rename-chain	Rinomina una catena, il vecchio nome ed il nuovo devono essere passati come parametri, le catena predefinite non possono essere rinominate.
-L	--list	Elenca le regole presenti nella catena; se non si passa nessun parametro elenca le regole di tutte le catene nella tabella.
-F	--flush	Cancella tutte le regole presenti nella catena; se non si passa nessun parametro cancella le regole di tutte la catene nella tabella.
-Z	--zero	Azzera i contatori di una catena; se non si passa nessun parametro azzera i contatori di tutte le catene nella tabella.
-P	--policy	Imposta la politica di default per la catena, prende come parametro addizionale la destinazione da usare per tutti i pacchetti che arrivano alla fine della catena senza incontrare una corrispondenza in alcuna regola.

Tabella 3.3: Opzioni per la gestione delle catene.

TOS di IP, e le varie opzioni delle regole. Sempre con `-L` si possono usare anche due altre opzioni, `-n` che disabilita la risoluzione degli indirizzi e delle porte, che verranno stampati in forma numerica, e `--line-numbers` che stampa un numero progressivo per le regole presenti nella catena, in modo da poterlo utilizzare con le opzioni `-D`, `-R` e `-I`.

Dato che il funzionamento del *netfilter* prevede la scansione sequenziale delle catene, il comando `iptables` prevede diverse opzioni, riportate in tab. 3.4, per inserire o cancellare una regola all'interno di una catena in una posizione definita. Per ciascuna di questa opzioni deve essere specificata come parametro la catena da utilizzare, che può essere sia una di quelle predefinite, che una di quelle create con i comandi di tab. 3.3.

Opzione		Significato
-A	--append	Inserisce la regola in cosa alla catena. In questo modo la regola sarà sempre l'ultima della catena, fintanto che non se ne aggiunge un'altra.
-D	--delete	Cancella una regola nella catena, sia indicandola per posizione numerica nella catena, che esprimendola integralmente.
-R	--replace	Rimpiazza la regola nella catena, identificata con il numero che ne indica la posizione, con quella specificata nel comando.
-I	--insert	Inserisce una regola in una posizione specifica nella catena.

Tabella 3.4: Opzioni per l'inserimento di una regola all'interno di una catena.

Infine le opzioni `-R`, `-I` e `-D` prevedono un secondo parametro che specifica la posizione nella catena in cui rimpiazzare, inserire o cancellare una regola; questo deve essere indicato nella forma del numero progressivo della regola, che si può ottenere combinando l'opzione `--line-numbers` con quella per la stampa delle regole `-L`.

3.3.3 I criteri di selezione dei pacchetti

Come illustrato in sez. 3.3.1 oltre ad indicare tabelle e catene da usare, il comando `iptables` serve principalmente a creare le regole da inserire al loro interno. Per creare una regola allora, oltre alla catena su cui inserirla, occorre anche specificare un criterio di selezione dei pacchetti.

Per questo motivo la seconda classe di opzioni, la più numerosa, è quella che permette di definire i criteri per la selezione dei pacchetti, il cuore di ogni regola. Questi criteri possono essere di diversi tipi, i primi che affronteremo sono i criteri generici, che sono sempre disponibili perché forniti dall'infrastruttura stessa del *netfilter*, indipendentemente dal protocollo dei pacchetti o delle estensioni che si possono usare.

Si tenga presente poi che per quanto in seguito si elencheranno, per esigenza di chiarezza nell'esposizione, i vari criteri di selezione uno per uno, con le opzioni relative a ciascuno di essi, normalmente quando si definisce una regola da inserire in una catena, si possono specificare insieme più criteri di selezione. In tal caso il significato della regola è che un pacchetto, per avere una corrispondenza, deve soddisfarli tutti quanti; vale cioè una modalità di *AND* logico dei criteri di selezione, ad esempio una regola del tipo "`-s 192.168.1.24 -d 192.168.0.4`" richiede che il pacchetto abbia un certo indirizzo sorgente *e* un certo indirizzo di destinazione.

I criteri generici sono sostanzialmente i criteri che operano direttamente sui parametri più comuni contenuti nella intestazione dei pacchetti IP, e sulle interfacce da cui i pacchetti entrano o escono, e sono disponibili nativamente nel *netfilter* di Linux. Le relative opzioni sono le seguenti:

`-p, --protocol`
: Seleziona il pacchetto in base al protocollo del livello di trasporto, che deve essere specificato come parametro. I valori possibili sono `tcp`, `udp`, `icmp` o `all` per indicare tutti e tre. L'uso di questi valori speciali consente di attivare le opzioni specifiche relative a tali protocolli su cui torneremo più avanti, per cui in realtà quando si usano questi valori si va oltre i criteri generici. Oltre a questi nomi si può usare direttamente il valore numerico del protocollo, o il nome corrispondente che deve essere però indicato nel file `/etc/protocols`. Inoltre si può usare il carattere "`!`" per invertire la selezione. Possibili esempi sono:

  ```
  -p tcp
  --protocol ! udp
  ```

`-s, --src, --source`
: Seleziona il pacchetto in base all'indirizzo sorgente. L'indirizzo sorgente può essere specificato sia in forma numerica che con un nome a dominio; è però una pessima idea usare nomi che devono essere risolti attraverso una interrogazione al DNS, data la vulnerabilità di questo protocollo che può anche fornire risposte false. Si può anche indicare una intera rete, specificandola nella forma `indirizzo/rete`, dove la parte di rete può essere specificata sia come netmask, che con la notazione CIDR. Anche in questo caso si può invertire la selezione apponendo il carattere "`!`". Possibili esempi sono:

  ```
  -s 192.168.1.1
  -s 192.168.2.0/24
  --src 10.0.0.0/255.0.0.0
  --source ! 172.16.1.2
  ```

-d, --dst, --destination
Seleziona il pacchetto in base all'indirizzo di destinazione. Come per l'indirizzo sorgente si può specificare (con la stessa sintassi) sia un indirizzo che una intera rete. Possibili esempi sono:

```
-d 192.168.1.1
-d 192.168.2.0/24
--dst 10.0.0.0/255.0.0.0
--destination ! 172.16.1.2
```

-i, --in-interface
Seleziona il pacchetto sulla base dell'interfaccia di rete da cui proviene. L'opzione richiede come parametro il nome dell'interfaccia, e può essere usata solo sulle catene `INPUT`, `FORWARD` e `PREROUTING`. Oltre al nome esatto si può indicare una intera classe di interfacce usando il carattere "`+`", se ad esempio si vogliono selezionare tutte le interfacce Ethernet si potrà usare il parametro `eth+`. L'opzione supporta pure l'inversione della soluzione con l'uso del carattere "`!`". Possibili esempi sono:

```
-i ppp0
-i ! eth1
--in-interface tun+
```

-o, --out-interface
Seleziona il pacchetto sulla base dell'interfaccia di rete in uscita a cui è destinato. L'opzione richiede come parametro il nome dell'interfaccia, e può essere usata solo sulle catene `FORWARD`, `OUTPUT` e `POSTROUTING`. Supporta le stesse estensioni per indicare classi di interfacce e la negazione della selezione viste per l'analoga precedente. Possibili esempi sono:

```
-o ppp0
-o ! eth0
--out-interface tun+
```

-f, --fragment
Seleziona i pacchetti IP frammentati (si veda quanto detto a proposito in sez. 3.2.3 e sez. 3.4.3). Dato che solo il primo frammento contiene l'intestazione del protocollo IP i criteri di selezione basati su indirizzi sorgente o destinazione o altri dati desumibili dall'intestazione, non potranno mai corrispondere per dei frammenti successivi al primo che allora vengono selezionati tramite questa opzione. L'opzione può essere preceduta da un carattere "`!`" che inverte la selezione. Possibili esempi sono:

```
-f
! --fragment
```

Come appena accennato, qualora si richieda una corrispondenza per protocollo che coinvolge uno dei tre protocolli principali del TCP/IP (vale a dire quando si specifica uno dei criteri con `-p tcp`, `-p udp` o `-p icmp`) vengono attivate automaticamente una serie di ulteriori possibili criteri di selezione basati sulle caratteristiche particolari di questi stessi protocolli.

In particolare quando si usano sia `tcp` che `udp` si potranno effettuare selezioni ulteriori sui valori delle porte utilizzate da detti protocolli, secondo i criteri specificabili tramite le opzioni:

--sport, --source-port
Seleziona il pacchetto in base alla porta sorgente. La porta può essere specificata sia per numero che per il nome utilizzato in `/etc/services`.[21] Usando i numeri si può specificare un intervallo di porte usando il carattere "`:`" come separatore, omettendo uno dei due estremi di un intervallo si sottintende l'uso del relativo valore massimo (65535) o minimo (0). Anche in questo caso si può invertire una selezione apponendo ad essa il carattere "`!`". Possibili esempi sono:

```
--sport 25
--sport ! smtp
--source-port 0:1024
--source-port 25000:
```

--dport, --destination-port
Seleziona il pacchetto in base alla porta di destinazione. La porta può essere specificata sia per numero che utilizzando il nome utilizzato in `/etc/services`, usando la stessa sintassi vista in precedenza per `--sport`. Anche in questo caso si può invertire la selezione apponendo il carattere "`!`". Possibili esempi sono:

```
--dport 80
--dport ! ssh
--destination-port 25000:
```

Quando si effettua il filtraggio sul protocollo `udp` le precedenti sono le sole opzioni aggiuntive disponibili, nel caso invece si filtri su `tcp` invece diverranno utilizzabili anche le seguenti ulteriori opzioni:

--tcp-flags
Seleziona il pacchetto in base allo stato dei flag del protocollo TCP, questi sono una serie di bit nell'intestazione dei pacchetti utilizzati nella gestione delle connessioni. L'opzione richiede due parametri, il primo è la lista dei flag da tenere sotto controllo (separati da virgole e senza spazi) ed il secondo la lista di tutti e soli i flag che fra i precedenti si richiede siano attivi. I valori usati per specificare i flag sono quelli dei rispettivi nomi, come indicati nella definizione del protocollo,[22] cioè `SYN`, `ACK`, `FIN`, `RST`, `URG`, `PSH`, a cui si aggiungono i due valori `ALL` e `NONE`, il cui significato è ovvio. Possibili esempi sono:

```
--tcp-flags SYN,ACK,FIN,RST SYN
--tcp-flags ! SYN,ACK,RST SYN
```

--syn Seleziona i pacchetti che hanno il flag SYN attivo e i flag ACK e RST disattivi, che sono la caratteristica che identifica il pacchetto TCP usato per iniziare una connessione. Pertanto bloccare questi pacchetti significa bloccare ogni connessione TCP. È una abbreviazione della regola `--tcp-flags SYN,RST,ACK SYN`. Al solito apponendo ad essa il carattere "`!`" si inverte la selezione. Possibili esempi sono:

[21] si tenga presente però che in questo caso la scrittura delle regole viene ritardata dalla risoluzione del nome, e quando si scrive un gran numero di regole, il ritardo può essere sensibile.
[22] si faccia riferimento all'RFC 793.

```
--syn
! --syn
```

--tcp-option
Seleziona il pacchetto in base al valore della sezione dell'intestazione di un pacchetto TCP chiamata appunto *TCP option*. Questa indica delle sezioni appunto opzionali che si possono trovare nell'intestazione, queste hanno sempre una struttura divisa in tre campi in cui il primo indica il tipo di opzione, ed è un numero a 8 bit selezionabile con questa opzione. I restanti campi sono la lunghezza dell'opzione[23] ed i relativi dati. Usando il carattere "`!`" si inverte la selezione. Possibili esempi sono:

```
--tcp-option 16
! --tcp-option 16
```

--mss Seleziona i pacchetti TCP con flag SYN o SYN/ACK che abbiano il valore specificato per la MSS *Maximum Segment Size*, un parametro che controlla la dimensione massima dei pacchetti per quella connessione. Si può inoltre specificare un intervallo di valori usando il carattere "`:`". Possibili esempi sono:

```
--mss 100
--mss 150:200
```

Infine quando si specifica il protocollo ICMP con `-p icmp` diventa disponibile una ulteriore selezione tramite l'opzione `--icmp-type` che seleziona il tipo di pacchetti; si possono selezionare i vari tipi di pacchetti sia secondo il valore numerico, come specificato nella definizione del protocollo, (si veda l'RFC 792) che utilizzando uno dei nomi ottenibili con il comando `iptables -p icmp -h`, che si sono riportati in tab. 3.10.

3.3.4 Le estensioni dei criteri di selezione

I precedenti criteri di selezione sono quelli più elementari, che supportano la selezione sulla base dei principali parametri dei protocolli più importanti. La flessibilità del *netfilter* è che diventa possibile, scrivendo degli opportuni moduli per il kernel, estendere le capacità di selezione dei pacchetti.

Per questo esiste l'opzione `-m` (o `--match`) che permette di attivare queste estensioni, ciascuna delle quali ha un suo nome (di norma corrispondente al nome del modulo usato dal kernel). La sintassi comune in questi casi è richiedere l'uso dell'estensione con il comando `-m estensione` per poi utilizzarla usando le nuove opzioni ad essa relative, che diventano disponibili una volta che la si è attivata.

Grazie a questi moduli le capacità di selezione dei pacchetti possono essere ampliate in maniera incredibilmente estesa e complessa, mettendo a disposizione, oltre a criteri basati sulle proprietà specifiche di ciascun pacchetto (in genere relativi a dati presenti nelle intestazioni dei protocolli) altri criteri più astratti, relativi a proprietà più ampie e non legate al singolo pacchetto, come quelle usate dal filtro sugli stati il cui funzionamento è illustrato in sez. 3.2.3.

[23] essendo opzionale non è necessario che tutti i tipi di opzione siano riconosciuti, per questo se uno non lo è deve essere possibile scartare i dati relativi passando alla sezione successiva.

Un elenco delle principali estensioni dei criteri di selezione è quello seguente, in cui per ciascuna estensione si riportano le principali caratteristiche e le ulteriori opzioni di controllo abilitate:

`-m limit`
L'estensione permette di utilizzare uno dei criteri di selezione speciale che non dipende dal contenuto di un singolo pacchetto, ma da proprietà "*collettive*" degli stessi. Questa estensione permette di creare una selezione in base al flusso (nel senso di numero per unità di tempo) di pacchetti che corrispondono ad un determinato criterio. Si può così limitare il flusso di pacchetti accettati, in modo da rispondere ad eventuali attacchi di DoS (vedi sez. 1.1.3) basati su tecniche di *flood*.[24]

Il funzionamento di questa estensione non è banale, dato che non è semplice calcolare un flusso medio di pacchetti, in quanto il valore di una media temporale dipende strettamente dal periodo di tempo su cui la si calcola. Se si contasse semplicemente il numero di pacchetti che arrivano durante un certo intervallo di tempo, una raffica concentrata di pacchetti potrebbe dare un flusso elevatissimo se l'intervallo su cui si media è piccolo e coincide con il picco, e nullo nei successivi intervalli, con variazioni enormi dei possibili valori della media a seconda della dimensione dell'intervallo in cui la si calcola.

Per evitare questo problema e garantire comunque un flusso medio affidabile il meccanismo di selezione è assimilabile a quello di un *serbatoio* di pacchetti. Fintanto che nel serbatoio c'è spazio tutti i pacchetti vengono accettati, e il criterio di selezione corrisponde. Quando lo spazio si esaurisce ogni ulteriore pacchetto sarà scartato. Il serbatoio però viene svuotato ad un ritmo costante, cosicché dopo un certo tempo si libera dello spazio e un nuovo pacchetto potrà essere accettato. In questo modo si può garantire su una media di lungo periodo (a serbatoio vuoto si può avere un picco dovuto ad una raffica di pacchetti) un flusso costante.

Per controllare questo meccanismo l'estensione introduce due opzioni, la prima delle quali, `--limit-burst`, è quella che ci permette di stabilire la dimensione in numero di pacchetti del nostro serbatoio. Il suo valore di default è di 5 pacchetti; questo significa che una volta inserita la regola i primi 5 pacchetti saranno sempre accettati, mentre per i successivi si dovrà aspettare che il *serbatoio* si sia svuotato a sufficienza per accettarne di altri. Questo comportamento ci fa anche capire il significato del nome dell'opzione: con questa possiamo decidere fino a che punto vogliamo rispondere a dei flussi concentrati (i *burst*) di pacchetti.

Invece l'opzione che controlla lo *svuotamento* del nostro serbatoio, e definisce pertanto il flusso medio di pacchetti che accetteremo, è `--limit` che permette di definire la frequenza con cui si libera uno spazio per un nuovo pacchetto nel serbatoio (il valore di default è 3 all'ora). L'opzione prende un parametro del tipo `N/tempo` dove `N` indica il numero di spazi da liberare, e `tempo` l'intervallo di tempo in cui farlo; quest'ultimo deve essere specificato tramite una opportuna parola chiave, l'opzione accetta i valori `second`, `minute`, `hour`, o `day`. Possibili esempi sono:

[24] si chiamano così le tecniche che mirano a creare una "*inondazione*" di pacchetti nei confronti di una macchina bersaglio, in modo da poterla bloccare nel tentativo di rispondere.

```
--limit 3/second
--limit-burst 10
```

-m connbytes

L'estensione consente di effettuare una selezione in base al numero di pacchetti/bytes passati su una connessione, sia nelle singole direzioni che in totale, e permette così di individuare quelle che consumano più banda in modo da poterle adeguatamente abbassare di priorità. Perché l'estensione possa funzionare è necessario abilitare la contabilità dei pacchetti in transito sulle connessioni scrivendo un valore diverso da zero in `/proc/sys/net/netfilter/nf_conntrack_acct`, ma si tenga presente che questa sarà presente solo per le nuove connessioni create dopo l'abilitazione. Se l'informazione non è presente il criterio non corrisponderà mai.

L'estensione abilita tre nuove opzioni; la prima è `--connbytes` che indica l'intervallo dei valori in pacchetti o bytes su cui effettuare la selezione e deve essere specificata con valori numerici nella forma `min:max` dove la seconda parte (`:max`) può essere omessa per impostare solo un valore minimo, la selezione inoltre può essere invertita anteponendo il carattere "`!`". La seconda opzione è `--connbytes-dir` che indica quali direzioni del flusso di pacchetti prendere in esame nella selezione e richiede come argomento uno fra i valori `original` (i pacchetti in uscita che han creato la connessione) `reply` (le risposte) e `both` (entrambe le direzioni). La terza opzione è `--connbytes-mode` che invece consente di indicare se i valori specificati per `--connbytes` devono essere considerati come numero di pacchetti (`packets`), numero di byte (`bytes`) o dimensione media del pacchetto `avgpkt`. Un possibile esempio è:

```
--connbytes 1000000 --connbytes-dir both --connbytes-mode bytes
--connbytes 10000 --connbytes-dir reply --connbytes-mode packets
```

-m mac

L'estensione permette di selezionare, attraverso l'opzione `--mac-source`, il pacchetto sulla base del MAC address sorgente del pacchetto (in genere Ethernet) che arriva sull'interfaccia di rete. Questo criterio è disponibile solo sulle catene di `PREROUTING`, `FORWARD` e `INPUT`. Al solito è possibile invertire la selezione con "`!`". Possibili esempi sono:

```
--mac-source 00:0A:95:74:C3:D4
--mac-source ! 00:0A:95:74:C3:D4
```

-m mark

L'estensione permette di selezionare, attraverso l'opzione `--mark`, i pacchetti che attraversano il *netfilter* che sono stati marcati con un opportuno valore tramite l'uso dell'azione `MARK` (vedi sez. 3.3.5 per la spiegazione del meccanismo). L'opzione prende come parametro il valore numerico impostato per il relativo pacchetto. Questo può essere specificato anche nella forma `valore/maschera`, dove la maschera viene usata per eseguire un AND binario del valore del `MARK` del pacchetto prima di eseguire il confronto. Possibili esempi sono:

```
--mark 10
--mark 1/7
```

-m multiport
L'estensione permette di utilizzare le opzioni di selezione per le porte sorgente e destinazione `--source-ports` e `--destination-ports`. Queste sono molto simili (presentano solo una `s` aggiunta al nome) a quelle illustrate in sez. 3.3.4 per i protocolli TCP e UDP e come in quel caso possono essere usate solo se si è richiesta anche una selezione sui suddetti protocolli (con `-p tcp` o `-p udp`). L'estensione permette di usare una sintassi in cui diventa possibile selezionare le porte con una lista di numeri (separati da virgole e senza spazi, con un limite massimo di 15 porte diverse) invece che singolarmente o come intervalli. L'estensione definisce inoltre l'opzione `--ports` che permette di applicare la stessa selezione alla lista di numeri indicati richiedendo che porta destinazione e sorgente siano identiche, e corrispondenti ad una delle porte della lista. Possibili esempi sono:

```
--source-ports 25,80,111
--destination-ports 25,80,111
--ports 5000,5001
```

-m owner
L'estensione permette di filtrare usando alcune caratteristiche specifiche del processo che ha creato i pacchetti, ed è utilizzabile soltanto sulla catena di `OUTPUT`. Dato che alcuni pacchetti (ad esempio quelli ICMP) vengono creati dal kernel e non sono mai associabili ad un processo, essi non potranno mai essere selezionati da questa estensione. L'elenco delle opzioni possibili è il seguente:

- **--uid-owner** seleziona i pacchetti creati da un processo con user-ID effettivo uguale a quello specificato come parametro.
- **--gid-owner** seleziona i pacchetti creati da un processo con group-ID effettivo uguale a quello specificato come parametro.
- **--pid-owner** seleziona i pacchetti creati dal processo il cui *pid* è uguale a quello specificato.
- **--sid-owner** seleziona i pacchetti creati da un processo il cui session-ID è uguale a quello specificato come parametro.
- **--cmd-owner** seleziona i pacchetti creati da un processo invocato con il comando specificato.[25]

-m state
Questa è l'estensione che permette di utilizzare il filtro degli stati, la cui descrizione dettagliata è in sez. 3.2.4, usando le informazioni contenute nel sistema di *connection tracking* (anche questo descritto in sez. 3.2.3). L'estensione abilita l'opzione `--state`, cui si passa la lista (separata da virgole e senza spazi) degli stati che si vogliono selezionare, secondo i nomi riportati in tab. 3.2.

-m tos
Questa estensione permette di utilizzare l'opzione `--tos` per selezionare i pacchetti in base al valore del campo TOS (*Type of Service*) dell'intestazione dei pacchetti IP (vedi

[25] questa opzione è presente solo quando `iptables` è stato compilato con un kernel che supporta questa funzionalità.

fig. 3.7). Questo è un campo di 8 bit usato[26] per differenziare i flussi di dati, che serve a caratterizzarne la tipologia; ad esempio per indicare che i pacchetti devono essere trasmessi il più rapidamente possibile, o che invece occorre ottimizzare la quantità dei dati trasmessi, più che la rapidità o la latenza. L'opzione prende come parametro il valore numerico del TOS espresso in cifra esadecimale, o tramite uno dei valori simbolici illustrati in tab. 3.5.

Valore		Significato
`Minimize-Delay`	`0x10`	Ottimizzare la trasmissione per rendere più veloce possibile la ritrasmissione dei pacchetti (usato per traffico interattivo di controllo come SSH).
`Maximize-Throughput`	`0x8`	Ottimizzare la trasmissione per rendere il più elevato possibile il flusso netto di dati (usato su traffico dati, come quello di FTP).
`Maximize-Reliability`	`0x4`	Ottimizzare la trasmissione per ridurre al massimo le perdite di pacchetti (usato su traffico come TFTP o BOOTP).
`Minimize-Cost`	`0x2`	Ottimizzare la trasmissione per utilizzare i collegamenti con minor costo (usato per i protocolli di streaming).
`Normal-Service`	`0x0`	Nessuna richiesta specifica.

Tabella 3.5: Valori del campo TOS utilizzabili con l'opzione `--tos`.

`-m ttl`
: Questa estensione permette di utilizzare l'opzione `--ttl` per selezionare i pacchetti in base al valore del campo TTL (*Time to Live*) dell'intestazione dei pacchetti IP (vedi fig. 3.7). Il campo indica il numero di router che il pacchetto può ancora attraversare prima di essere scartato.

`-m unclean`
: L'estensione non definisce nessuna opzione ulteriore e si limita a selezionare tutti i pacchetti che sembrano essere malformati o avere caratteristiche non usuali. In genere la si usa per identificare il traffico sospetto. L'estensione è considerata ancora sperimentale e l'uso della stessa per scartare i pacchetti può risultare pericoloso e portare anche alla perdita di pacchetti facenti parti di connessioni perfettamente regolari.

3.3.5 Le azioni sui pacchetti

Il terzo ed ultimo componente di ogni regola di `iptables` è l'azione da applicare ad ogni pacchetto selezionato nella regola stessa, quello che nel vocabolario del comando è chiamato *target*. L'opzione che controlla questa azione, che decide il destino del pacchetto quando questo corrisponde alla regola di selezione, è `-j` (o `--jump`), seguita dall'identificatore dell'azione da applicare (il nome del *target*).

Come accennato in sez. 3.2.1 nell'infrastruttura del *netfilter*, a livello del kernel, sono già previste alcune azioni elementari possibili, che si riflettono in quelli che in `iptables` sono chiamati gli *special target*, cioè una serie di azioni che sono sempre definite e che determinano la conclusione della scansione delle regole all'interno di una catena predefinita (o in una eventuale ulteriore

[26] non molto, in realtà, dato che non tutti i sistemi operativi lo supportano e molti amministratori ignorano completamente questa proprietà nella configurazione dei router.

catena definita dall'utente su cui lo si poteva esser mandato). Una di queste è sempre quella di inviare il pacchetto su una catena definita dall'utente (da questo deriva il nome *jump*), le altre sono riportate in tab. 3.6.

Opzione	Significato
`ACCEPT`	Accetta il pacchetto che viene fatto immediatamente proseguire per le successive catene (secondo lo schema di attraversamento di fig. 3.4) all'interno nel *netfilter*.
`DROP`	Il pacchetto viene scartato immediatamente, ed ogni ulteriore controllo non è più necessario. Nessuna altra regola verrà presa in considerazione.
`QUEUE`	Il pacchetto viene passato in user space (dove può essere letto dagli opportuni programmi).
`RETURN`	Blocca l'attraversamento della catena corrente e torna a quella precedente, se usato su una catena predefinita applica la politica di default della stessa.

Tabella 3.6: Le azioni elementari (*special target*) del comando `iptables`.

Una delle caratteristiche di `iptables` è che oltre ai criteri di selezione è possibile ampliare anche le possibili azioni che si possono eseguire sui pacchetti, al di là di quelle elementari di tab. 3.6, attraverso quelle che vengono chiamate le "*target extension*", sempre grazie all'utilizzo di ulteriori moduli nel kernel che definiscono nuovi parametri (corrispondenti a nuovi *target*) da utilizzare con l'opzione `-j`, che a loro volta possono richiedere ulteriori opzioni.

Le principali estensioni dei *target* disponibili come azioni da compiere sui pacchetti è data dalla seguente lista (non completa, dato che molte estensioni non è detto siano incluse nei sorgenti ufficiali del kernel):

`DNAT` Questa è una azione valida solo nella tabella di `nat` e nelle catene `PREROUTING` e `OUTPUT`, e nelle eventuali catene definite dall'utente se vi si *salta* a partire da una di queste.

Questa azione permette di riscrivere l'indirizzo di destinazione di un pacchetto. Come già accennato in sez. 3.2.3 la regola di selezione viene applicata solo sul primo pacchetto di una connessione (necessita quindi dell'uso del sistema del *conntrack*) e poi viene automaticamente ripetuta per tutti i pacchetti facenti parte della stessa connessione. Una volta modificato il pacchetto questo verrà instradato secondo il nuovo indirizzo.

Si tenga presente inoltre che siccome il *Destination NAT* è operato nella catena di `PREROUTING`, nel successivo attraversamento del *netfilter* i pacchetti modificati con `DNAT` si presenteranno con l'indirizzo modificato, per cui tutte le eventuali regole di firewall che si vogliono mettere nelle catene successive dovranno usare quest'ultimo. Lo stesso vale per i pacchetti modificati sulla catena di `OUTPUT`, in quanto la catena di `nat` è attraversata prima di quella di `filter` (si riveda lo schema di fig. 3.4). In caso di DNAT le sole catene che vedono l'indirizzo originale del pacchetto sono quella di `PREROUTING` e quella di `OUTPUT` ma solo nella tabella di `mangle`.

La traslazione degli indirizzi viene specificata attraverso l'opzione `--to-destination`, questa può prendere come parametro un singolo indirizzo di destinazione, od un intervallo che va indicato con gli indirizzi degli estremi separati da un "`-`" senza spazi interposti. In questo caso la traslazione sceglierà a turno uno dei valori dell'intervallo

(dato che la scelta è fatta solo sul primo pacchetto di una connessione, ed i successivi pacchetti subiscono la stessa traslazione, questo garantisce che tutto il traffico vada verso lo stesso IP finale). In questo modo si può eseguire una forma semplificata (*round-robin*) di bilanciamento di carico.

Qualora si effettui la traslazione per pacchetti TCP o UDP, si potrà effettuare la traslazione anche della porta di destinazione, da specificare singolarmente facendola seguire all'indirizzo di destinazione separata con un ":". Si potrà anche specificare un intervallo di porte indicando i due estremi separati da un "-". Questo è possibile solo avendo abilitato con `-p` la relativa selezione sui suddetti protocolli. Possibili esempi sono:

```
--to-destination 192.168.2.2
--to-destination 192.168.2.2:8080
--to-destination 192.168.2.16-192.168.2.32
--to-destination 192.168.2.16:6000-6010
```

LOG Questa azione abilita la registrazione sul *syslog* dei dati dei pacchetti selezionati. Dato che la registrazione viene eseguita dal kernel la facility del *syslog* utilizzata è `kern`; questo significa che i messaggi vengono mostrati anche dal comando `dmesg` e inviati, a meno di non ridurne la priorità, sulla console. Una regola con criterio di selezione che usi questa azione come *target* non termina la scansione della catena, che proseguirà dalla regola successiva. Per questo se si vuole eseguire il logging di pacchetti che poi si desidera scartare occorre sempre usare una azione di `LOG` prima di quella di `DROP`. Si tenga presente inoltre che i log contengono informazioni dettagliate riguardo il vostro traffico di rete, che possono essere lette dagli utenti e costituire un possibile rischio di sicurezza.

Con l'uso di questa azione vengono attivate una serie di opzioni che permettono di definire le modalità in cui il logging viene eseguito, il loro elenco è il seguente:

`--log-level`
: imposta la priorità dei messaggi, prende come parametro il valore numerico della priorità o il corrispondente identificativo, come si può ottenere dalla pagina di manuale del *syslog*, accessibile con `man syslog.conf`. Si può usare questo valore e le impostazioni del *syslog* per evitare che i log prodotti da `iptables` siano letti dagli utenti. Possibili esempi sono:

  ```
  --log-level 3
  --log-level debug
  ```

`--log-prefix`
: Permette di aggiungere una stringa (di un massimo di 29 caratteri) da premettere ai messaggi di log, una caratteristica utile per aumentarne la leggibilità. Possibili esempi sono:

  ```
  --log-prefix "INPUT"
  ```

--log-tcp-sequence
: Permette di inserire nei log i *numeri di sequenza*[27] dei pacchetti TCP. Questa opzione non ha parametri.

--log-tcp-options
: Permette di inserire nei log il valore delle opzioni del protocollo TCP. Questa opzione non ha parametri.

--log-ip-options
: Permette di inserire nei log il valore delle opzioni del protocollo IP. Questa opzione non ha parametri.

MARK Questa azione è valida solo nella tabella di `mangle`, e serve per marcare i pacchetti con un valore numerico, questo non va a modificare niente nel contenuto del pacchetto stesso, ma è una sorta di etichetta che viene attaccata al pacchetto che resterà associata al pacchetto all'interno del kernel. Così sarà possibile filtrare i pacchetti sulla base di tale valore con l'estensione `--mark` vista in sez. 3.3.4, ma lo stesso valore sarà accessibile anche dal sistema di routing avanzato ed in questo modo si potranno impostare politiche di routing diverse (ad esempio una limitazione sul traffico) per i pacchetti "*marcati*" tramite questa azione.

Il valore da usare per marcare il pacchetto deve essere specificato attraverso l'opzione `--set-mark` che prende come parametro il valore numerico da associare al pacchetto. Si ricordi che questo valore è puramente interno al kernel e pertanto non sarà disponibile quando il pacchetto transiterà su altre macchine. Possibili esempi sono:

```
--set-mark 4
```

MASQUERADE
Questa azione è valida soltanto nella tabella di `nat` e nella catena di `POSTROUTING`. Sostanzialmente è una abbreviazione di `SNAT` e viene usata principalmente per condividere le connessioni temporanee ad Internet (in genere quelle via modem gestite con `pppd` dove viene assegnato un IP dinamico). L'azione è quella di fare un *Source Network Address Translation* usando l'IP assegnato dinamicamente (ricavato da quello assegnato all'interfaccia da cui i pacchetti devono uscire). In tutti gli altri casi è meglio usare `SNAT` in quanto la risoluzione dell'indirizzo dall'interfaccia comporta un certo degrado delle prestazioni.

L'uso di `MASQUERADE` ha inoltre l'effetto di cancellare tutte le connessioni presenti quando l'interfaccia va giù, dato che è poco probabile avere lo stesso indirizzo quando l'interfaccia sarà riutilizzata e che in ogni caso si saranno dovuti bloccare tutti i programmi che le usavano. In questo modo si evita che restino ad occupare spazio nella tabella delle connessioni voci ormai inutilizzabili, e che si possa ricevere del traffico spurio corrispondente a connessioni che in realtà non esistono più.

Se si opera su pacchetti TCP e UDP (specificando l'uso di detti protocolli con `-p`) con l'uso di questa azione viene abilitata anche l'opzione `--to-ports` che può essere

[27] i *numeri di sequenza* sono delle informazioni caratteristiche del protocollo TCP (vedi fig. 3.9), mantenute nelle intestazioni dei pacchetti, che identificano ciascun pacchetto e permettono di inserirlo al posto giusto nel flusso dei dati quando questo viene riassemblato.

usata per soprassedere l'assegnazione automatica delle porte sorgente fatta dal kernel (torneremo su questo nella spiegazione di SNAT) con un intervallo da specificare come parametro indicandone i due estremi separati con un "-". Si può anche indicare una sola porta. Possibili esempi sono:

```
--to-ports 10000
--to-ports 10000-15000
```

REDIRECT
Questa azione è valida soltanto nella tabella di nat e nelle catene di PREROUTING e OUTPUT (o nelle catene definite dall'utente che derivino da queste) e serve a modificare l'IP di destinazione dei pacchetti selezionati perché questi vengano inviati alla macchina stessa (i pacchetti generati localmente vengono mappati sul localhost). Qualora si siano selezionati (con -p) dei pacchetti TCP o UDP diventa disponibile l'opzione --to-ports che permette anche di specificare su quale porta effettuare la redirezione (normalmente viene usata quella originaria del pacchetto). L'opzione permette anche di specificare un intervallo, indicato tramite gli estremi estremi separati con un "-". Possibili esempi sono:

```
--to-ports 8080
--to-ports 8000-8010
```

REJECT Questa azione è sostanzialmente equivalente a DROP, ma oltre ad eliminare il pacchetto fermando ogni ulteriore scansione delle regole, si incarica anche di inviare indietro un opportuno messaggio di errore. L'azione è utilizzabile solo nelle catene di INPUT, FORWARD e OUTPUT. L'azione abilita l'uso dell'opzione --reject-with che permette di specificare il pacchetto inviato come messaggio di errore, e prende come parametro uno dei valori riportati in tab. 3.7.

Opzione	Significato
icmp-net-unreachable	Restituisce un pacchetto ICMP di tipo *net unreachable*.
icmp-host-unreachable	Restituisce un pacchetto ICMP di tipo *host unreachable*.
icmp-port-unreachable	Restituisce un pacchetto ICMP di tipo *port unreachable*.
icmp-proto-unreachable	Restituisce un pacchetto ICMP di tipo *proto unreachable*.
icmp-net-prohibited	Restituisce un pacchetto ICMP di tipo *net prohibited*.
icmp-host-prohibited	Restituisce un pacchetto ICMP di tipo *host prohibited*.
icmp-admin-prohibited	Restituisce un pacchetto ICMP di tipo *admin prohibited*.
tcp-reset	Restituisce un pacchetto TCP con il flag RST attivo.

Tabella 3.7: I vari tipi di messaggi di errore utilizzabili con l'opzione --reject-with.

Il valore di default è icmp-port-unreachable, degli altri icmp-admin-prohibited può essere usato solo con i kernel che lo supportano, mentre tcp-reset può essere usato solo con regole che selezionano pacchetti TCP, e serve a chiudere in maniera pulita una connessione TCP.

SNAT Questa azione è valida solo nella tabella di nat e nella catena di POSTROUTING. Come le altre regole che operano nella tabella di nat essa si applica solo al primo pacchetto di una connessione e poi viene automaticamente ripetuta per tutti i pacchetti facenti parte

della stessa connessione. L'azione modifica l'indirizzo sorgente dei pacchetti eseguendo il *Source NAT*,[28] e si usa di solito per condividere una connessione ad internet, facendo in modo che tutti i pacchetti in uscita figurino come originanti dalla macchina corrente; il meccanismo del *connection tracking* si incarica di effettuare automaticamente la traslazione inversa (sull'indirizzo di destinazione) per i pacchetti ottenuti in risposta ai precedenti.

La traslazione degli indirizzi viene specificata attraverso l'opzione `--to-source`, questa può prendere come parametro un singolo indirizzo sorgente, od un intervallo che va indicato con gli indirizzi degli estremi separati da un "`-`" (senza spazi interposti). In questo caso la traslazione sceglierà a turno uno dei valori dell'intervallo, ottenendo una forma semplificata (*round-robin*) di bilanciamento.

Qualora si effettui la traslazione per pacchetti TCP o UDP (selezionati in precedenza con `-p`) si può anche specificare (con sintassi analoga a quella di `DNAT`) una porta sorgente o un intervallo di porte da utilizzare per la traslazione. In tal caso i pacchetti uscenti avranno come porta sorgente un valore compreso nell'intervallo specificato. Possibili esempi sono:

```
--to-source 192.168.2.2
--to-source 192.168.2.16-192.168.2.20
--to-source 192.168.2.16:10000-30000
```

In realtà per poter gestire più connessioni contemporanee l'utilizzo di questa opzione prevede che possano essere modificati, oltre l'indirizzo sorgente, anche altre caratteristiche dei pacchetti. Il kernel cerca di modificare il meno possibile i pacchetti limitandosi al solo indirizzo sorgente, ma qualora le modifiche siano necessarie perché si ha un conflitto (ad esempio due connessioni allo stesso server sullo stesso servizio che usano la stessa porta sorgente) il kernel deve intervenire per consentire una identificazione univoca della connessione. Questo nel caso dei pacchetti TCP e UDP significa che deve essere modificata anche la porta sorgente, nel qual caso la regola seguita è che (sempre che non si sia specificato in un intervallo definito) se la porta originale è inferiore a 512 vengono usate altre porte sotto 512, se è fra 512 a 1024 vengono usate porte inferiori a 1024 e per tutte le altre porte sono usate quelle a partire da 1024.

Nel caso di ICMP invece, non esistendo i numeri di porta, in caso di più pacchetti inviati allo stesso IP, viene modificato il campo di codice.

`NOTRACK` Questa azione è valida solo nella tabella di `raw` e consente di disabilitare il *connection tracking* per tutti i pacchetti che corrispondono al criterio di selezione usato nella regola che la usa come `target`.

`TOS` Questa azione è valida solo nella tabella di `mangle`, ed abilita l'uso dell'opzione `--set-tos` che permette di modificare il valore del campo TOS nell'intestazione di IP,[29] usato per

[28] questo è il caso più comune di NAT, che viene implementato di default nella gran parte dei router usati per connettersi alle linee DSL, tanto che spesso si usa NAT come indicativo di questo particolare tipo di traslazione.

[29] come accennato in sez. 3.3.3 per l'omonimo criterio di selezione, il *Type of Service* è un campo usato per classificare il traffico IP in una serie di classi in modo che possa essere trattato diversamente dai router.

dare ai router indicazioni sul tipo di traffico di cui il pacchetto fa parte. L'opzione prende come parametro il nuovo valore del TOS, che può essere specificato sia con il valore numerico esadecimale che con uno degli identificativi già visti in tab. 3.5 per l'opzione relativa all'omonimo criterio di selezione. Possibili esempi sono:

```
--set-tos 0x10
--set-tos Normal-Service
```

TTL Questa azione è valida solo nella tabella di `mangle`, ed abilita l'uso di una serie di opzioni che permettono di modificare il valore del campo TTL di IP, usato per limitare il numero di volte che un pacchetto può transitare attraverso un router, in modo da evitare l'intrappolamento permanente in eventuali *routing loop*.[30] Questa azione può servire a riportare allo stesso valore di TTL tutti i pacchetti uscenti dalla macchina, in modo da mascherare all'esterno i valori diversi che indicherebbero la presenza di una operazione di `SNAT`. Le opzioni abilitate con l'uso di questa azione sono:

- **`--ttl-set`** imposta un valore per il TTL dei pacchetti selezionati, da specificare come parametro per l'opzione (il valore di default usato dal kernel è 64).
- **`--ttl-dec`** permette di decrementare il valore del TTL dei pacchetti selezionati del valore specificato come parametro. Questo può essere utilizzato per bloccare il raggiungimento di certi servizi (di norma il DNS) da macchine troppo lontane, per costringerle ad usare server più vicini.
- **`--ttl-inc`** permette di incrementare il valore del TTL dei pacchetti selezionati del valore specificato come parametro, dato che al passaggio da un router questo valore viene decrementato di uno, con questa opzione lo si può reincrementare in modo da non far rilevare la presenza del nostro firewall in programmi come `traceroute`.

Possibili esempi sono:

```
--ttl-set 64
--ttl-dec 5
--ttl-inc 1
```

3.3.6 Programmi di gestione

Abbiamo fin qui trattato le varie capacità del *netfilter* di Linux, e come queste possono essere utilizzate attraverso l'uso del comando `iptables`; esso però consente di eseguire sulle tabelle del kernel solo una operazione alla volta, per questo sono stati creati degli opportuni programmi per permettere la gestione di *gruppi* di regole.

Il comando `iptables` infatti permette di inserire le regole una alla volta, per questo in genere si suole costruire un firewall tramite uno script di shell che si incarica di invocare `iptables` più

[30] si chiama così quella situazione in cui, per un qualche malfunzionamento della rete, i pacchetti si trovano ad essere reinviati ad un router da cui sono provenuti, innescando così un circolo vizioso; per questo il valore di questo campo viene decrementato di una unità ogni volta che un pacchetto attraversa un router e quando si annulla il pacchetto viene scartato e viene inviato in messaggio di errore (*ICMP time expired*).

volte. Quando però le regole diventano molte questo meccanismo comporta una penalizzazione nelle prestazioni, infatti si deve ripetere il comando continuamente, e questo comporta che ogni volta deve essere estratto dal kernel l'intero insieme delle regole, che deve essere opportunamente modificato secondo quanto specificato dal comando, e poi reinserito nel kernel, compiendo così un gran numero volte il trasferimento dei dati da kernel space ad user space.

Per velocizzare queste operazioni insieme ad `iptables` vengono forniti altri due programmi che permettono il trasferimento di un gruppo di regole in un blocco unico. Il primo è `iptables-save`, che serve a scrivere (sullo standard output) l'insieme di regole correntemente attive, utilizzando un apposito formato di facile scansione. Il comando permette anche, con l'opzione `-c` (o `--counters`) di salvare i valori dei contatori associati a ciascuna regola,[31] mentre con `-t` (o `--table`) si può richiedere il salvataggio delle regole relative ad una singola tabella, invece che a tutte e tre. Un esempio del formato generato da questo comando è il seguente:

iptables-save

```
# Generated by iptables-save v1.2.9 on Sun Dec 14 19:52:50 2003
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -s 127.0.0.1 -d 127.0.0.1 -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
...
COMMIT
# Completed on Sun Dec 14 19:52:50 2003
```

Il secondo comando, `iptables-restore`, permette di caricare nel kernel un insieme di regole letto dallo standard input,[32] nello stesso formato usato da `iptables-save`. A meno di non specificare l'opzione `-n` (o `--noflush`) le regole presenti vengono cancellate, mentre con `-c` (o `--counters`) si può richiedere il ripristino dei relativi contatori.

Il problema con `iptables-restore` è che esso accetta, come è ovvio, solo un contenuto statico, ed è pertanto impossibile utilizzarlo con una qualche forma di scripting per permettergli ad esempio di usare un indirizzo dinamico. In casi come questi l'unica soluzione possibile è quella di scrivere uno script per modificare direttamente il file su cui si sono salvate le regole da utilizzare, prima di darlo in pasto al comando stesso.

3.4 Criteri per la costruzione di un firewall

Una volta esaminata la struttura del *netfilter* e le funzionalità dei comandi che permettono di utilizzarlo, passiamo a definire i criteri generali per il filtraggio dei pacchetti, e le regole da utilizzare per la realizzazione pratica di un firewall, entrando nei dettagli di come questo poi possa essere realizzato con Linux.

[31] e può servire quando non si vogliono perdere questi valori, che potrebbero essere utilizzati a fini statistici, tutte le volte che si modificano le regole del firewall.

[32] ovviamente, grazie alle normali capacità di redirezione presenti in qualunque shell, è sempre possibile utilizzare un file qualsiasi.

3.4.1 Le funzionalità dirette del kernel

Le regole del *netfilter* non sono il solo componente da usare nella realizzazione di un firewall; oltre al sistema del *netfilter* infatti il kernel fornisce direttamente anche una serie di funzionalità che, pur non essendo direttamente legate al filtraggio dei pacchetti, sono di grande utilità nella realizzazione di un firewall. Queste sono controllate dall'interfaccia del *sysctl*,[33] che permette di impostare i valori di tutta una serie di parametri interni al kernel.

L'accesso a questi parametri è di norma effettuabile attraverso il filesystem `/proc`;[34] infatti tutti i parametri sono visibili (e modificabili) come file all'interno della directory `/proc/sys`. Nel nostro caso quelli che ci interessano sono i parametri relativi alla rete, mantenuti nella sottodirectory `net` ed in particolare quelli usati per controllare il comportamento dello stack TCP/IP del kernel, a loro volta disponibili nell'ulteriore sottodirectory `ipv4`.

In tab. 3.8 si sono riportati, insieme ad una breve spiegazione del loro significato, ed al tipo di valore utilizzato, i principali parametri del kernel relativi allo stack TCP/IP rilevanti per la costruzione di un firewall, modificabili scrivendo direttamente negli omonimi file presenti nella directory `/proc/sys/net/ipv4`. Una descrizione più completa dei parametri presenti in questa directory (relativi alla gestione dello stack TCP/IP) si trova nella documentazione allegata al kernel, nel file `Documentation/networking/ip-sysctl.txt`.

Parametro	Tipo	Significato
`ip_forward`	`bool`	Abilita il kernel all'inoltro dei pacchetti da una interfaccia ad un'altra, di default è disabilitato (valore nullo). Un valore non nullo lo abilita.
`icmp_echo_ignore_all`	`int`	Istruisce il kernel a ignorare tutte le richieste di pacchetti ICMP di tipo *echo reply* dirette alla macchina.
`icmp_echo_ignore_broadcast`	`bool`	Istruisce il kernel a ignorare tutte le richieste di pacchetti ICMP di tipo *echo reply* dirette a indirizzi di broadcast o multicast.
`icmp_ratelimit`	`int`	Istruisce il kernel a limitare il numero di pacchetti ICMP che possono essere inviati per numero. Un valore nullo disabilita la limitazione, un valore positivo indica il numero di pacchetti al millisecondo.
`icmp_ratemask`	`int`	Imposta la maschera binaria dei pacchetti ICMP per i quali vale la limitazione di flusso impostata con il precedente parametro `icmp_ratelimit`. Viene attivata la limitazione per ciascun tipo di pacchetto per cui il bit nella posizione corrispondente al valore del tipo è attivo.
`tcp_syncookies`	`bool`	Abilita la protezione dei *syncookies* nei confronti di attacchi di *syn flood*.

Tabella 3.8: I parametri del kernel che controllano il comportamento dello stack TCP/IP di rilevanza per la costruzione di firewall, con relativo tipo di dato e spiegazione del significato.

In realtà il primo parametro, `ip_forward`, è usato nella costruzione dei firewall solo per attivare l'inoltro dei pacchetti da una interfaccia ad un'altra, una funzionalità che in realtà non

[33] per una trattazione di questa interfaccia si può fare riferimento al capitolo 6 di [GaPiL], *Guida alla Programmazione in Linux*, disponibile su `http://gapil.gnulinux.it`.

[34] si può usare anche il comando `sysctl`, ma in genere l'interfaccia è più macchinosa; molte distribuzioni però consentono di impostare detti parametri all'avvio attraverso il file `/etc/sysctl.conf`, per i dettagli si possono consultare le relative pagine di manuale.

ha niente a che fare con il filtraggio dei pacchetti, ma che è necessaria quando si usano più interfacce.

Gli altri parametri permettono di bloccare o limitare direttamente (senza l'uso di specifiche regole) alcuni pacchetti ICMP, infine il parametro `tcp_syncookies` richiede un kernel che sia stato compilato con la relativa opzione, ed abilita una protezione specifica contro i cosiddetti *syn flood*, attacchi DoS in cui la propria macchina viene *inondata* di pacchetti di inizio connessione (i SYN appunto). Per bloccare questo tipo di attacchi quando la coda delle connessioni per un socket viene esaurita inizia l'emissione di speciali pacchetti (i cosiddetti *syncookies*) e riprende l'accettazione di ulteriori SYN solo in corrispondenza di una risposta. Questo però modifica il funzionamento usuale del protocollo TCP, ed è da usare solo in caso di emergenza in quanto degrada comunque notevolmente alcuni servizi.

Oltre ai parametri di tab. 3.8, che controllano il comportamento dello stack TCP/IP del kernel nel suo insieme, esistono una serie di altri parametri che permettono di configurarne il comportamento per ciascuna interfaccia. Questi sono accessibili all'interno della directory `/proc/sys/net/ipv4/conf`, che contiene a sua volta delle ulteriori directory (ad esempio `lo`, `eth0`, ecc.) per ciascuna delle interfacce di rete presenti nel sistema, e la directory `default` per indicare l'interfaccia usata per la *default route*.

Oltre a queste è inoltre presente la directory speciale `all` che contiene gli stessi parametri delle altre, in questo caso però essi assumono un significato diverso e possono essere usati o per attivare la relativa funzionalità contemporaneamente su tutte le interfacce, o per abilitare la possibilità di usarla; in quest'ultimo caso occorrerà attivare esplicitamente la funzionalità anche nella directory relative a ciascuna interfaccia (per quelle su cui la si vuole). In tab. 3.9 si sono riportati i parametri rilevanti per la costruzione di un firewall.

Parametro	Tipo	Significato
`forwarding`	`bool`	Abilita il kernel all'inoltro dei pacchetti sull'interfaccia.
`rp_filter`	`bool`	Abilita la protezione *antispoofing* sull'interfaccia.

Tabella 3.9: I parametri del kernel di rilevanza per la costruzione di firewall, che controllano il comportamento dello stack TCP/IP per le singole interfacce.

Il primo parametro può essere usato per restringere il reinstradamento dei pacchetti fra le sole interfacce selezionate attraverso di esso, evitando l'attivazione generale che si ottiene con il precedente `ip_forward`.

Il secondo invece permette di attivare una protezione *antispoofing*[35] direttamente a livello di instradamento. Una interfaccia per cui si attiva questa protezione non farà passare nessun pacchetto che arrivi su di essa che presenti un indirizzo sorgente non corrispondente alla rete su cui l'interfaccia stessa si affaccia.

[35] in questo contesto si parla di *spoofing* quando si falsificano gli indirizzi sorgenti dei pacchetti, per farli apparire come provenienti da altre macchine; in genere questa tecnica viene usata per attacchi di tipo *Distributed Denial of Service* (detti DDoS) in cui si inviano pacchetti di questo tipo, con l'indirizzo sorgente della macchina bersaglio, ad un gran numero di macchine su internet, in modo che la loro risposta *allaghi* il bersaglio, saturando la banda.

3.4.2 Regole generali e politiche di gestione

Prima di entrare nei dettagli relativi all'uso dei vari criteri di selezione e filtraggio dei pacchetti, è opportuno prendere in esame alcune regole generali da applicare nella costruzione di un firewall. Si tratta in sostanza di chiarirsi quale metodo usare per la politica di gestione dei pacchetti, che in sostanza si traduce nella decisione riguardo alla *policy* di default da applicare alle catene (quella che si imposta con l'opzione `-P` di `iptables`).

Esistono di fatto due politiche di gestione, che si riflettono anche nei due possibili valori della *policy* delle catene, quella in cui si lascia passare tutto (`ACCEPT` nel linguaggio di `iptables`) per poi selezionare quello che si vuole bloccare, e quella in cui si blocca tutto di default (`DROP` nel linguaggio di `iptables`) per poi abilitare solo il traffico consentito.

La prima politica ha il vantaggio che funziona tutto di default; ma questo è l'unico vantaggio, infatti richiede una pianificazione attenta delle regole per individuare tutti i possibili traffici anomali che si vogliono bloccare, e che proprio per la loro caratteristica di essere anomali non sono affatto facili da individuare. Il grosso problema di questa politica, che la rende estremamente debole sul piano della sicurezza, è che tutto quello che non avete previsto, ma che un attaccante può inventarsi, passa. Pertanto non la prenderemo ulteriormente in considerazione.

Se invece si usa la politica di bloccare tutto per default, e poi far passare solo il traffico consentito, avremo l'inconveniente che all'inizio non funziona nulla, ma il grande vantaggio che dovremo solo individuare il traffico che ci interessa far passare, dato che tutto il resto sarà comunque bloccato. Questo ovviamente rende impossibile costruire da zero una simile politica da remoto, ma permettere l'accesso da remoto ad un firewall, sia pure per un servizio controllato come SSH, non è comunque una buona idea. Nel caso comunque abbiate questa necessità dovrete per forza partire consentendo almeno il traffico necessario alla connessione di controllo.[36]

Da questo si potrebbe evincere l'indicazione di cambiare la politica di default di tutte le catene a `DROP`, ma in realtà questo non è necessario e comporta il rischio di chiudersi fuori (se per esempio si esegue un reset delle regole con `-F`). Basterà invece mettere come ultima regola di ciascuna catena un `-j DROP`, senza nessun criterio di selezione, così che comunque tutto quello che verrebbe sottoposto alla politica di default (che può restare ad `ACCEPT`) viene bloccato.

In questo modo un firewall ben scritto dovrà contenere soltanto (a parte quella finale) delle regole terminanti con `ACCEPT`, con la sola eccezione (che sarebbe comunque meglio riformulare adeguatamente) in cui si usa `DROP` per escludere preventivamente una selezione di pacchetti fatti passare da un successivo `ACCEPT` troppo ampio.

3.4.3 I criteri di filtraggio per IP

In sez. 3.3.3 abbiamo descritto in dettaglio le varie opzioni di `iptables` che permettono di selezionare pacchetti, il problema adesso è quello di capirne i possibili criteri di applicazione. Cominceremo iniziando a esaminare come utilizzare i criteri di selezione per i pacchetti IP, ma per fare questo occorre prima capire il significato dei vari campi che possono essere filtrati e l'uso che ne viene fatto all'interno del funzionamento del protocollo.

In fig. 3.7 si è riportato lo schema della testata di un pacchetto IP; i due dati fondamentali sono l'IP destinazione, che indica la macchina cui il pacchetto è diretto, e l'IP sorgente, che

[36] il rischio di *chiudersi fuori* in questo caso è molto alto, per cui almeno all'inizio, le prove conviene farle da una console, e non da remoto.

indica la macchina da cui proviene (e a cui pertanto saranno inviate le risposte). Il *netfilter* è in grado di selezionare i pacchetti in base a questi campi (con le opzioni `-d` e `-s` di `iptables`).

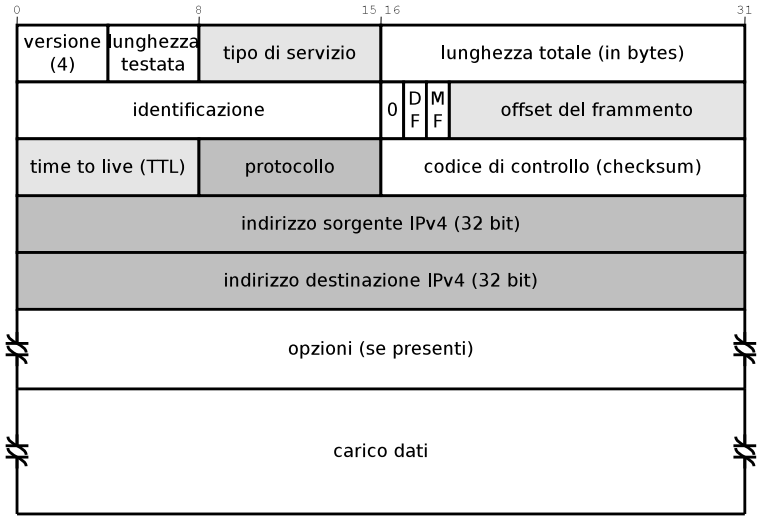


Figura 3.7: Lo schema della testata di un pacchetto IP, con evidenziati i campi controllabili con `iptables`.

Il filtraggio in base ad indirizzi destinazione e sorgente è la funzionalità più basilare di un firewall e permette di selezionare i pacchetti in base alle reti da cui provengono e sono destinati. É con questi criteri di selezione che si possono abilitare o disabilitare gli accessi a reti o singole macchine.

Si tenga presente però che per una qualunque connessione i pacchetti con i quali inviamo dati verso un server avranno come sorgente il nostro IP e destinazione quello del server, mentre quelli di risposta avranno come sorgente l'IP del server (è lui infatti che ce li spedisce) e destinazione il nostro. Pertanto per selezionare tutto il traffico fra due macchine usando solo i valori dei relativi indirizzi, occorrono due regole, una per il traffico in entrata e l'altra per quello in uscita.

Se si vogliono selezionare le connessioni in uscita verso un qualunque indirizzo di destinazione, senza però accettare quelle in ingresso, non si può pertanto usare un criterio basato soltanto sugli indirizzi in gioco,[37] in quanto per accettare il traffico di ritorno si dovrebbe garantire l'accesso da qualunque IP, consentendo con questo anche la possibilità di connessioni in ingresso. Per questa operazione è necessario l'uso del filtro sugli stati (con la sola eccezione del protocollo TCP, che vedremo in sez. 3.4.5).

Il terzo criterio di base per la selezione dei pacchetti (con l'opzione `-p` di `iptables`), è quello sul campo che contiene il numero identificativo del protocollo. Questo campo identifica il contenuto del carico di dati contenuto nel resto del pacchetto, che è sempre un pacchetto di un altro protocollo trasportato da IP. Vedremo in seguito quali altri possibilità di filtraggio si attivano qualora questo protocollo sia uno dei tre protocolli principali (TCP, UDP e ICMP). In generale

[37] il trucco delle due regole vale solo per le connessioni ad una singola macchina, e comunque una regola di questo tipo accetta pacchetti di qualunque connessione verso di noi provenienti da essa, non solo le risposte ad una nostra connessione su di essa.

il *netfilter* permette di selezionare i pacchetti in base a qualunque valore, e questo può essere usato ad esempio per redirigere il traffico relativo a certi protocolli (con il DNAT) a macchine dedicate al loro trattamento.

L'ultimo criterio di base è quello che permette di selezionare i pacchetti frammentati (con l'opzione `-f` di `iptables`). La frammentazione è una caratteristica del protocollo IPv4, che permette ai router di spezzare pacchetti troppo grandi per poter essere trasmessi su un segmento di rete. Se un pacchetto eccede la MTU di un tratto di rete non potrà essere trasmesso integralmente su quel tratto, pertanto il router lo suddividerà in varie parti. Questo significa che il campo dati sarà diviso in varie parti di dimensione opportuna, cui sarà aggiunta una nuova intestazione che contiene nel campo *offset* l'indicazione di quale parte dei dati del pacchetto originale è contenuta.

Questo comporta che tutti i pacchetti generati dallo stesso frammento (eccettuato il primo) non conterranno l'intestazione del protocollo successivo, per cui tali pacchetti non passeranno nessun criterio di selezione basato su valori relativi a questi. Attivando il sistema del *conntrack* i pacchetti vengono deframmentati all'ingresso nel *netfilter*, e quindi il criterio diventa inutile, ma se non lo si usa si possono selezionare tutti i frammenti successivi al primo (sulla base di un valore non nullo del capo *offset*).

Come accennato non si usano quasi mai dei criteri di filtraggio basati solo su IP, l'unica eccezione può essere quella del filtro relativo al *localhost*, in tal caso infatti, dato che tutto il traffico deve provenire ed arrivare dallo stesso indirizzo, non c'è il problema di dover consentire traffico verso il localhost da un indirizzo qualsiasi,[38] e sono perfettamente legittime (e necessarie, se si vuole poterlo usare) regole come:

```
iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

Sebbene i criteri di selezione basati su IP non siano mai usati da soli, sono però quelli usati più comunemente in tutte le altre regole, sia per limitare gli accessi da e per sottoreti definite (in genere in combinazione con il filtro degli stati), che per selezionare i protocolli su cui applicare le ulteriori regole di selezione.

3.4.4 I criteri di filtraggio per ICMP

Il protocollo ICMP, come indica il nome (*Internet Control Message Protocol*) è un protocollo speciale che viene utilizzato per l'invio di messaggi di controllo, e che è fondamentale per il funzionamento di IP. Con `iptables` si possono selezionare i pacchetti ICMP con `-p icmp`. I pacchetti ICMP vengono trasmessi direttamente su IP, il protocollo prevede una intestazione molto semplice il cui formato è illustrato in fig. 3.8.

Il protocollo classifica i messaggi in base ai valori dei due campi *tipo* e *codice*. Ciascun tipo di messaggio ha un suo preciso significato e viene impiegato per un certo compito ben definito. Un elenco dei vari tipi di pacchetti, insieme all'identificativo da passare come argomento all'opzione `--icmp-type` e ad una breve descrizione del significato, si trova in tab. 3.10.

In alcuni casi (la cosa vale soltanto per i pacchetti di tipo *destination unreachable*, *redirect*, *time-exceeded* e *parameter problem*) per lo stesso tipo di pacchetto possono esistere diverse

[38] un tale traffico, tra l'altro, sarebbe oltremodo sospetto.

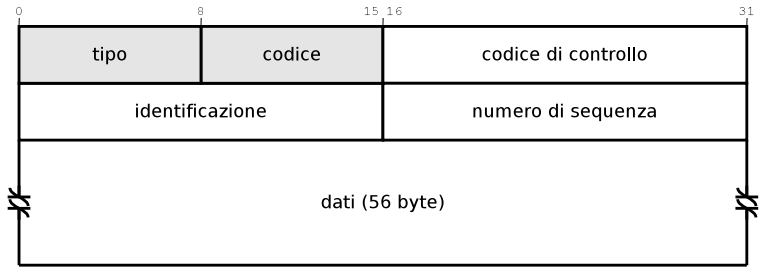


Figura 3.8: Lo schema della testata di un pacchetto ICMP, con evidenziati i campi controllabili con `iptables`.

condizioni di errore o diversi significati che il messaggio deve poter portare, questi allora vengono ulteriormente specificati attraverso un diverso valore del campo *codice*.

La selezione `iptables` consente di scegliere anche questi specifici pacchetti (i valori di tab. 3.10 selezionano solo sulla base del tipo, e il valore del codice viene ignorato), attraverso i valori riportati in tab. 3.11, nella quale si è riportato anche (dividendo la tabella in sezioni diverse per ciascuno dei quattro tipi di pacchetto precedentemente citati), il valore numerico del campo codice.

Valore	**Tipo**	**Significato**
`any`	–	Seleziona tutti i possibili valori
`echo-reply`	0	Inviato in risposta ad un ICMP *echo-request*
`destination-unreachable`	3	Segnala una destinazione irraggiungibile, viene inviato all'IP sorgente di un pacchetto quando un router realizza che questo non può essere inviato a destinazione.
`source-quench`	4	Inviato in caso di congestione della rete per indicare all'IP sorgente di diminuire il traffico inviato.
`redirect`	5	Inviato per segnalare un errore di routing, richiede che la macchina sorgente rediriga il traffico ad un altro router da esso specificato.
`echo-request`	8	Richiede l'invio in risposta di un `echo-reply`.
`time-exceeded`	11	Inviato quando il TTL di un pacchetto viene azzerato.
`parameter-problem`	12	Inviato da un router che rileva dei problemi con l'intestazione di un pacchetto.
`timestamp-request`	13	Richiede l'invio in risposta di un `timestamp-reply`.
`timestamp-reply`	14	inviato in risposta di un `timestamp-request`.
`info-request`	15	Richiede l'invio in risposta di un `info-reply`.
`info-reply`	16	Inviato in risposta di un `info-request`.
`address-mask-request`	17	Richiede l'invio in risposta di un `address-mask-reply`.
`address-mask-reply`	18	Inviato in risposta di un `address-mask-request`.

Tabella 3.10: Tipi di pacchetti ICMP selezionabili con `--icmp-type`.

In sez. 3.2.3 abbiamo visto come alcuni pacchetti ICMP, *echo request*, *netmask request*, *timestamp request* e *information request* (ed i corrispondenti pacchetti di risposta) vengano classificati dal sistema del *conntrack*. Di questi le ultime tre coppie riguardano delle funzionalità poco usate, e possono essere tranquillamente ignorati e venire bloccati da un firewall. Tra l'altro la risposta ad un *timestamp request* permette di conoscere il valore dell'orologio di sistema di una macchina, e questo può essere utile in certi attacchi.

Valore	Codice
network-unreachable	0
host-unreachable	1
protocol-unreachable	2
port-unreachable	3
fragmentation-needed	4
source-route-failed	5
network-unknown	6
host-unknown	7
host-isolated	8
network-prohibited	9
host-prohibited	10
TOS-network-unreachable	11
TOS-host-unreachable	12
communication-prohibited	13
host-precedence-violation	14
precedence-cutoff	15
network-redirect	0
host-redirect	1
TOS-network-redirect	2
TOS-host-redirect	3
ttl-zero-during-transit	0
ttl-zero-during-reassembly	1
ip-header-bad	0
required-option-missing	1

Tabella 3.11: Specifici pacchetti ICMP classificati in base ai valori del campo *codice*, suddivisi per tipo: *destination unreachable*, *redirect*, *time-exceeded* e *parameter problem*.

I pacchetti *echo request* e *echo reply* costituiscono invece la base del comando `ping` e sono molto utilizzati per scopi diagnostici, per questo non è il caso di scartarli a priori. Il problema è che molti attacchi di tipo *Distributed Denial of Service* si basano appunto sull'uso di questi pacchetti. In genere il trucco è quello di inviare (da macchine compromesse) una serie di *echo request* falsificando l'indirizzo sorgente in quello della macchina da attaccare. Questo comporta che essa riceverà tutte le risposte. Se si aggiunge che in molti casi i router lasciano passare degli *echo request* in *broadcast*, questo consente un grosso effetto moltiplicativo (un solo pacchetto scatena la risposta di tutta una rete) che conduce in genere ad una *alluvione* di *echo reply* (questo tipo di attacchi infatti viene spesso chiamato *flood*) sulla macchina vittima dell'attacco.

Per questo motivo alcuni amministratori poco competenti tendono a filtrare completamente tutto ICMP, invece di provvedere a risolvere il problema seriamente. In tal caso infatti quello che va fatto è bloccare le risposte a richieste in *broadcast* (questo può essere fatto con Linux con i parametri `icmp-ignore-broadcast` del kernel illustrati in sez. 3.4.1), e inserire (con l'estensione `limit`) un controllo sul flusso massimo di questo tipo di pacchetti.

Inoltre per evitare di essere usati come *complici* di un tale attacco occorre abilitare sui *router di frontiera*[39] la protezione *antispoofing* (sempre descritta in sez. 3.4.1, con il parametro `rp_filter`) che blocca i pacchetti uscenti da una interfaccia il cui indirizzo sorgente non corrisponda alla rete che sta su quell'interfaccia.

[39] si chiamano così (*border router*) i router che stanno ai bordi di Internet, che hanno cioè dietro di loro dei tratti di rete chiusi, dai quali non si rientra in Internet; in sostanza quelli che non stanno sulle dorsali.

Una volta fatto questo occorre poi compiere una scelta relativa ai pacchetti ICMP che si vogliono far passare, tenendo presente che alcuni di essi sono importanti per il buon funzionamento della rete, e che bloccarli tutti danneggia anzitutto noi stessi.

Un pacchetto pericoloso, e da filtrare senz'altro, è il `redirect`, in quanto esso comporta la possibilità di riscrivere da remoto la tabella di routing di una macchina modificando la rotta per la destinazione dei pacchetti, con possibilità enormi di attacchi di tipo DoS. Il pacchetto infatti veniva utilizzato quando un router identificava una rotta migliore per un pacchetto arrivato a lui, con esso si indicava alla macchina sorgente di redirigere il traffico direttamente sulla nuova rotta; il problema è che un pacchetto appositamente forgiato potrebbe ad esempio farvi redirigere il traffico sul localhost, con effetti tutt'altro che gradevoli per la vostra rete.

Lo stesso problema, anche se in maniera molto minore, può derivare dall'accettare i pacchetti `source-quench` che vengono usati dai router per comunicare uno stato di congestione della linea, richiedendo appunto uno *smorzamento* del traffico. In genere la loro ricezione fa eseguire al kernel un rallentamento del traffico in uscita verso quella destinazione, in modo da diminuire la congestione. Di nuovo un uso *malizioso* potrebbe portarvi a ridurre il traffico anche quando non ce n'è bisogno, ma il problema è senz'altro minore rispetto al `redirect`.

I pacchetti `time-exceeded` e `destination-unreachable` sono invece pacchetti utili alla rilevazione di problemi sulla rete, e sono di norma da accettare, è in particolare molto importante accettare il pacchetto `fragmentation-needed` in quanto viene usato per la determinazione della MTU (la dimensione massima) dei pacchetti da inviare su una connessione, la cosiddetta *Path MTU*.

Come accennato in sez. 3.4.3 quando un router individua un pacchetto la cui dimensione eccede la capacità del tratto di rete su cui deve inviarlo, lo frammenta. Questa operazione rallenta sia le operazioni del router, e aumenta (per la necessità di header aggiuntivi) il consumo di banda, nuocendo alla velocità di trasmissione; dato inoltre che la *Path MTU* può variare, è stato previsto il meccanismo del *Path MTU Discovery* (descritto nell'RFC 1191) che permette di riconoscere quale è il valore massimo delle dimensioni di un pacchetto da inviare su una collegamento, senza che questo debba essere frammentato. Il meccanismo sfrutta il flag *Don't Fragment* del protocollo IP (vedi fig. 3.7) che richiede al router di non eseguire la frammentazione di pacchetti di dimensione eccessiva, ma di generare in ritorno un pacchetto ICMP di tipo `destination-unreachable` e codice `fragmentation-needed`, ricevendo il quale si può riconoscere quando un pacchetto ha una dimensione eccessiva e determinare quella corretta.

Un elenco tipico di regole riguardanti ICMP è pertanto il seguente:

```
iptables -A INPUT -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -p icmp --icmp-type fragmentation-needed -j ACCEPT
```

3.4.5 I criteri di filtraggio per TCP

Il protocollo TCP è il più utilizzato dei protocolli trasportati su IP, ed è l'unico dei protocolli della suite TCP/IP che supporta nativamente connessioni e stati. Realizzare tutto questo però ha un suo costo, e come si può vedere dalla quanto mostrato in fig. 3.9, la testata dei pacchetti TCP è una delle più complesse.

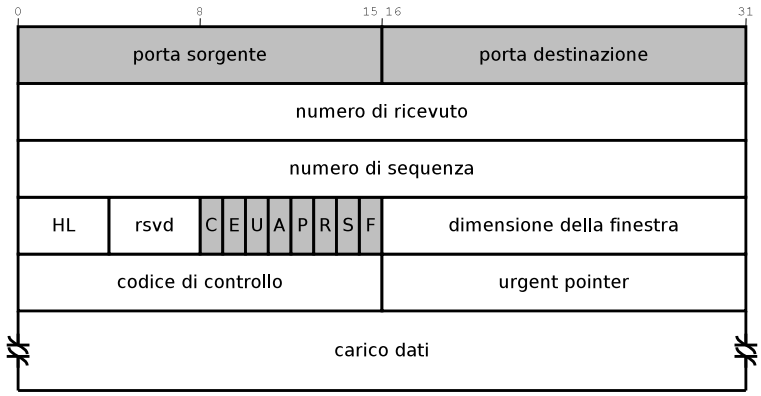


Figura 3.9: Lo schema della testata di un pacchetto TCP, con evidenziati i campi controllabili con `iptables`.

La caratteristica principale di TCP, comune con UDP come protocollo di trasporto, è la presenza di due campi di 16 bit usati per mantenere il numero di porta (sorgente e destinazione), che vengono utilizzati per distinguere, fra tutti i pacchetti che transitano fra due macchine, quelli appartenenti ad una specifica connessione. Come abbiamo visto queste possono essere selezionate attraverso l'uso delle opzioni `--sport` e `--dport`.

Usare solo i valori delle porte per identificare una connessione non è però pratico. Come abbiamo visto in sez. 3.2.3 infatti le due direzioni del flusso di dati all'interno di una connessione hanno destinazione e sorgente invertita. Allora anche se è possibile selezionare tutti i pacchetti di una connessione qualora siano note le porte (destinazione e sorgente) usate, il problema sorge dal fatto che queste non sono determinabili a priori. Infatti in una connessione solo una porta è sempre ben stabilita (quella relativa al servizio cui ci si vuole collegare), l'altra, che viene detta per questo *effimera*, viene scelta dal kernel della macchina di chi inizia la connessione con un valore dinamico, preso fra quelli disponibili, e può essere qualunque. Pertanto, se si vuole ricevere in maniera generica il traffico di ritorno di una connessione, si dovrà consentire l'accesso a tutto un insieme di porte, e benché questo sia possibile, è poco pratico (e fa passare qualunque pacchetto proveniente dalla stessa origine, anche se non collegato a nessuna connessione).

Per questo l'uso classico della selezione sulle porte TCP è quello sulla porta destinazione combinato con il filtro degli stati; questo permetterà di selezionare per quali porte di destinazione si accettano pacchetti in stato `NEW`, consentendo di effettuare connessioni verso di esse, sia in ingresso (quali server locali possono essere contattati dall'esterno) che in uscita (a quali servizi esterni si consente l'accesso).

In generale perciò i criteri di filtraggio basati su TCP si riducono sostanzialmente alla scelta dei servizi cui di vuole dare accesso. È usuale consentire l'accesso via SSH alle macchine sulla DMZ, purtroppo spesso questo viene fatto anche per il firewall nonostante il rischio che tutto ciò comporta. In genere, oltre a SSH, i servizi più comuni sono il web, e la posta elettronica, che vanno posti nella DMZ. Un elenco tipico di regole che consentono l'accesso a questi servizi su un server è il seguente:

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -m state --state NEW -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
```

In genere è da evitare di fornire accesso ad eventuali database (anche se posti nella DMZ) dall'esterno, limitando questo solo alle macchine nella stessa rete. Inoltre è assolutamente da escludere l'accesso a servizi tipici di un rete interna, come i file server (Samba) o i servizi per le autenticazioni centralizzate (LDAP o NIS). Qualora alcuni dati di questi servizi dovessero risultare utili per le macchine presenti in DMZ si dovrà piuttosto predisporre un meccanismo di replicazione delle informazioni, (e solo di quelle indispensabili), a partire dalla rete interna e verso l'esterna (e mai il viceversa).

Per quanto riguarda l'accesso verso l'esterno tutto dipende da eventuali politiche di restrizione degli accessi che si intendono implementare, ad esempio si può consentire soltanto l'accesso a web e posta bloccando tutto il resto. In genere poi, per alcuni servizi, si può provvedere alla impostazione di un *proxy trasparente*, utilizzando le capacità di redirezione del *netfilter*.

3.4.6 I criteri di filtraggio per UDP

Il secondo protocollo più utilizzato su internet dopo il TCP è UDP, questo protocollo non è altro che un semplice involucro (un *wrapper*) per utilizzare le caratteristiche di IP a livello di trasporto. Pertanto l'intestazione del protocollo, illustrata in fig. 3.10, è estremamente semplice, e si limita ad aggiungere ad IP i campi relativi alle porte (per poter supportare diversi canali di comunicazione) e due campi ausiliari di controllo (la lunghezza del pacchetto e la somma di controllo).

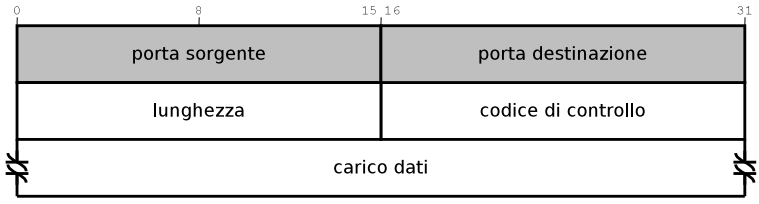


Figura 3.10: Lo schema della testata di un pacchetto UDP, con evidenziati i campi controllabili con `iptables`.

Benché nel caso di UDP non esistano connessioni (il protocollo si limita a garantire solo il massimo sforzo nella consegna dei dati) si possono comunque, come fa il filtro degli stati, identificare dei canali di comunicazione; questi hanno (in termini di porte coinvolte e flussi dei relativi pacchetti) la stessa caratteristiche di una connessione TCP, e pertanto valgono anche in questo caso le considerazioni precedenti relative al filtraggio in base al valore della porta di destinazione.

Come per il TCP si tratta di selezionare quali sono i servizi a cui si vuole consentire l'accesso dall'esterno, il principale servizio che opera su UDP è il DNS, che di nuovo deve essere posto in DMZ; un altro servizio importante che utilizza UDP è NFS, ma come per Samba (e qualunque altro meccanismo di condivisione dei file della rete interna) esso non deve mai reso visibile all'esterno. Un esempio di regole utilizzate per UDP è il seguente:

```
iptables -A INPUT -p udp --dport 53 -m state --state NEW -j ACCEPT
iptables -A INPUT -p udp --dport 500 -m state --state NEW -j ACCEPT
```

che forniscono accesso remoto al DNS ed al servizio IKE per IPSEC (che vedremo in sez. 4.1.2).

3.4.7 Un esempio di firewall

Vedremo adesso una serie di regole che possono essere utilizzate per costruire un firewall elementare; si è fatto la scelta di non creare e commentare uno script già pronto, perché lo scopo è quello di mettere in grado le persone di scrivere il proprio. Per questo useremo volutamente esempi particolari (senza definire variabili al posto delle varie interfacce e indirizzi) e regole spesso incomplete, in modo che ciascuno, una volta compresi i concetti che le sottendono, debba a sua volta creare un suo script.

Negli esempi usati per illustrare i criteri di filtraggio sui vari protocolli si è sempre fatto riferimento ad una singola macchina presa come origine o destinazione finale dei pacchetti, operando quindi sulle catene di `INPUT` e `OUTPUT`; nel caso si debba installare un firewall però si tratterà di operare principalmente sulla catena di `FORWARD`, dovendo filtrare i pacchetti in transito fra diverse interfacce di rete.

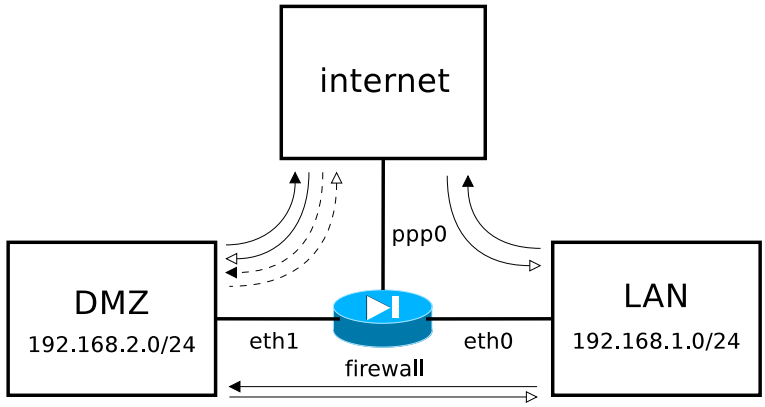


Figura 3.11: Schema delle interfacce per un firewall di esempio nella configurazione di base con DMZ, con illustrazione dei flussi di dati.

Il caso più comune, utilizzando una topologia come quella di fig. 3.2, è quello di consentire l'accesso ad internet dalla rete interna senza restrizioni, di impedire l'accesso da internet alla rete interna e di consentire l'accesso dall'esterno alla DMZ solo per i servizi pubblici che si sono posti su di essa.

Come esempio particolare consideriamo la disposizione delle interfacce e delle sottoreti illustrata in fig. 3.11, in cui si sono anche riportati i flussi dei pacchetti da consentire secondo quanto detto in sez. 3.1.3. In figura si è indicato un flusso di dati che permette la connessione con una freccia piena ed il flusso dei pacchetti di ritorno con la freccia vuota; inoltre si è usata una linea continua per indicare un flusso generico non filtrato e una discontinua per quello consentito solo per alcuni servizi.

Il primo passo è allora quello di definire quali pacchetti vogliamo comunque far passare. Useremo il filtro degli stati, per cui in generale questi saranno tutti quelli in stato `ESTABLISHED` o `RELATED`, in quanto risposte a connessioni consentite, più i pacchetti ICMP di controllo di

cui abbiamo parlato in sez. 3.4.4; metteremo il tutto su una catena `allowed` utilizzando una definizione del tipo:

firewall.sh
```
iptables -N allowed

iptables -A allowed -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A allowed -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A allowed -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A allowed -p icmp --icmp-type echo-request -j ACCEPT
#iptables -A allowed -p icmp --icmp-type fragmentation-needed -j ACCEPT
```

dove prima si crea la catena, e poi vi si inseriscono le regole per i pacchetti appena citati.

Attenendoci rigidamente al criterio per cui non deve essere possibile arrivare da internet alla rete interna, bloccheremo anche ogni accesso diretto al firewall dall'esterno. La soluzione più sicura è quella di bloccare qualunque connessione remota diretta al firewall, per comodità si può sacrificare un po' di sicurezza concedendo un accesso limitato solo dalla rete interna. Questo significa che per quanto riguarda le due catene di `INPUT` e `OUTPUT` (che si ricordi riguardano solo i pacchetti diretti o provenienti dal firewall) basteranno una serie di regole del tipo:

firewall.sh
```
#
# INPUT
#
iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A INPUT -j allowed
# iptables -A INPUT -i eth0 -s 192.168.1.0/24 --state NEW -j ACCEPT
iptables -A INPUT -j DROP
#
# OUTPUT
#
iptables -A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A OUTPUT -m state --state NEW -j ACCEPT
iptables -A OUTPUT -j allowed
iptables -A OUTPUT -j DROP
```

Nell'esempio si abilita sia in ingresso che in uscita tutto il traffico sul `localhost`, e si accettano i pacchetti delle connessioni già stabilite e quelli ICMP di controllo secondo quanto già specificato nella catena `allowed`; poi si permettono le nuove connessioni in uscita dal server. Volendo abilitare delle connessioni in ingresso si può ad esempio scommentare la riga in cui si abilitano le connessioni dirette al firewall solo se provenienti dalla corretta interfaccia e dalla relativa sottorete (sarebbe opportuno comunque essere più restrittivi e limitarle sulla base di singoli indirizzi, ad esempio quello della macchina dell'amministratore).

Una volta definito cosa fare dei pacchetti destinati direttamente alla macchina che fa da firewall la configurazione di quest'ultimo riguarda sostanzialmente i pacchetti in transito, e cioè la catena di `FORWARD`. Per consentire i flussi illustrati in fig. 3.11 si dovranno definire delle regole del tipo:

firewall.sh
```
iptables -A FORWARD -j allowed
iptables -A FORWARD -i eth0 -o ppp0 -m state --state NEW -j ACCEPT
```

```
iptables -A FORWARD -i eth0 -o eth1 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i eth1 -o ppp0 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i ppp0 -o eth1 -m state --state NEW -j services
iptables -A FORWARD -j DROP
```

In questo caso prima si fanno passare i soliti pacchetti sempre consentiti tramite la catena `allowed`, poi si abilita l'accesso a partire dalla LAN sia per la DMZ che per internet; volendo essere pignoli, il che quando si tratta di sicurezza è sempre un pregio, occorrerebbe specificare anche le reti di origine e quelle di destinazione, lo lasciamo per esercizio. Si dà poi accesso ad internet dalla DMZ, ed infine per le connessioni provenienti da internet e dirette sulla DMZ si userà una catena `services` opportunamente creata.

Il grosso del lavoro di filtraggio è da fare su `services`, qui si dovranno definire a quali servizi e su quali server si potrà accedere da internet; di nuovo prendiamo un esempio in cui nella DMZ ci sono sulla solo due macchine, la `192.168.2.2` che fa da server di posta e DNS, e la `192.168.2.3` che fa da server web, in questo caso potremo usare delle regole del tipo:

firewall.sh

```
iptables -N services
iptables -A services -d 192.168.2.2 -p tcp --dport 25  -j ACCEPT
iptables -A services -d 192.168.2.2 -p udp --dport 53  -j ACCEPT
iptables -A services -d 192.168.2.3 -p tcp --dport 80  -j ACCEPT
```

Fin qui si sono trattate solo le regole destinate alla gestione del firewall, ma avendo usato nell'esempio un caso di reti private sia per la DMZ che per la LAN queste non sono in realtà sufficienti. L'ipotesi dell'esempio di fig. 3.11 è allora quella che il firewall faccia anche da router per una connessione con un IP pubblico solo per `ppp0`.[40] In questo caso è necessario usare le funzionalità della tabella di `nat` per compiere le opportune trasformazioni degli indirizzi.

Il primo passo è quello di eseguire un *Source NAT* per i pacchetti uscenti su internet in modo che questi assumano l'indirizzo pubblico del nostro *router/firewall*; questo si ottiene andando ad operare sulla catena di `POSTROUTING` della tabella di `nat` con una regola del tipo:

firewall.sh

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

che esegue il *Source NAT* su tutti i pacchetti uscenti su `ppp0`.

Si tenga presente che `MASQUERADE` è una azione speciale, che esegue il *Source NAT* andando a vedere quale è l'IP assegnato sulla relativa interfaccia; per questo in genere la si usa solo per le interfacce su cui l'IP è assegnato dinamicamente, dato che è in grado di riconoscere al volo un eventuale cambiamento dello stesso (ed è quindi tipica delle connessioni *dial-in*). Questo processo comporta un piccolo *overhead* per la determinazione dell'indirizzo, per cui se l'IP è statico è più opportuno usare direttamente il *Source NAT* su quest'ultimo, con qualcosa del tipo:

firewall.sh

```
iptables -t nat -A POSTROUTING -o ppp0 -j SNAT --to-source 62.94.208.119
```

[40] è il caso tipico di una connessione via modem, sia su linea telefonica analogica, che via ADSL.

Il passo successivo è quello di redirigere le connessioni provenienti da internet per i servizi cui si vuole dare accesso sui rispettivi server nella DMZ; questo significa dover eseguire un *Destination NAT* relativo a detti servizi, cosa che si otterrà operando sulla catena di `PREROUTING` della tabella di `nat` con un qualcosa del tipo:

firewall.sh
```
iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 80 -j DNAT \
                --to-destination 192.168.2.3
iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 25 -j DNAT \
                --to-destination 192.168.2.2
iptables -t nat -A PREROUTING -i ppp0 -p udp --dport 53 -j DNAT \
                --to-destination 192.168.2.2
```

Si noti come in questo caso si sia utilizzata la selezione sull'interfaccia di provenienza per identificare le connessioni entranti da internet, e selezionati i pacchetti solo sulla base della porta di destinazione. Questo perché nell'esempio in questione c'è un unico indirizzo pubblico possibile, quello associato con `ppp0`. Nel caso invece si abbiano più indirizzi esterni, con una configurazione in cui la macchina fa solo da firewall (come in fig. 3.2 e non anche da router per la connessione, come in fig. 3.11) e si vogliano usare per i server nella DMZ degli IP diversi,[41] bisognerà selezionare i pacchetti in arrivo anche in base agli indirizzi pubblici cui sono rivolti e non solo in base all'interfaccia.

Infine si noti come in tutti gli esempi precedenti le regole di firewall sono state espresse usando direttamente indirizzi presenti sulle reti locali, anche per i servizi pubblici messi nella DMZ; questo è corretto perché la traslazione degli indirizzi avviene sulle due catene di `PREROUTING` e `POSTROUTING`, cioè rispettivamente prima dell'entrata e dopo l'uscita dal firewall e pertanto a livello della tabella di `filter` si può senza problemi fare riferimento agli indirizzi come li si vedono all'interno della propria rete locale.

[41] una tecnica usata comunemente è quella di assegnare comunque tutti gli IP pubblici (con l'*IP aliasing*) all'interfaccia esterna del firewall, e poi eseguire comunque un DNAT su altrettanti diversi IP privati, messi all'interno della DMZ.

Capitolo 4

Virtual Private Network

4.1 Cenni di teoria delle VPN

Prima di parlare della implementazione pratica e delle modalità di installazione, uso e configurazione delle VPN, faremo una introduzione su cosa sono le VPN, quali caratteristiche hanno, cosa le contraddistingue, e sulle basi teoriche (algoritmi e protocolli) sulle quali queste vengono costruite.

4.1.1 Cos'è una VPN

Con la sigla VPN (*Virtual Private Network*) si intende in realtà una classe di tecnologie che permettono di mettere in comunicazione appunto delle *reti private* attraverso internet. Con reti private si intendono in genere reti non accessibili direttamente da internet, che di norma utilizzano gli indirizzi non instradabili dell'RFC 1918,[1] e condividono la connessione ad internet attraverso un router che effettua il mascheramento di detti indirizzi.[2]

Dato che questi indirizzi non vengono instradati dai router è ovviamente impossibile collegare direttamente attraverso Internet due reti di questo tipo, anche se non usano gli stessi indirizzi. Se però si trovasse il modo di far arrivare in una qualche maniera i pacchetti che le due reti vogliono scambiarsi ai due relativi router, questi non avrebbero alcun problema a consegnarli ai relativi destinatari.

Una VPN è questo canale di comunicazione alternativa che permette di trasmettere i pacchetti fra due router secondo lo schema di fig. 4.1, canale che di norma, come misura di ulteriore sicurezza, viene opportunamente cifrato in modo che nel transito da internet il traffico trasmesso non possa essere osservato; una caratteristica che rafforza l'uso del nome di *private*.

In generale non è necessario, per avere una VPN, che i due tratti di rete agli estremi della stessa usino indirizzi privati; e di per sé per collegare due reti private non è necessario che il canale di comunicazione sia cifrato, ma questa è una delle caratteristiche più tipiche di una

[1] cioè gli indirizzi della rete di classe A `10.0.0.0/8`, delle 16 reti di classe B da `172.16.0.0/16` a `172.31.0.0/16` e delle 256 reti di classe C da `192.168.0.0/24` a `192.168.255.0/24`.

[2] abbiamo visto in sez. 3.3.5 come eseguire questa operazione con il *netfilter*, attraverso il target `SNAT` nella tabella di `nat`.

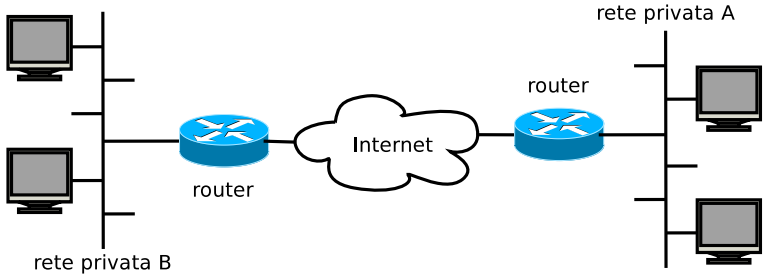


Figura 4.1: Lo schema generico di una VPN che collega due reti private.

VPN, senza la quale si avrebbe un semplice *tunnel*. Per questo normalmente il cuore di una VPN è un algoritmo crittografico che costituisce la base su cui è possibile costruire un canale di comunicazione sicuro attraverso cui passeranno i nostri dati.

Inoltre quello di fig. 4.1 è solo lo schema più comune di utilizzo di una VPN, in realtà si possono usare le VPN anche dare accesso ad una singola macchina (nella configurazione detta *road warrior*) all'interno di una rete altrimenti invisibile dall'esterno, o per mettere in contatto due macchine singole (configurazione limite della precedente, in cui la rete si riduce ad una sola macchina).

Infine è sempre possibile, usando più canali, mettere in comunicazione fra loro un numero arbitrario di reti, purché ovviamente queste mantengano indirizzi non in conflitto fra di loro. In questo caso avremo ovviamente una struttura più complessa, anche se alla base resta un incrocio di tante configurazioni del tipo di quella mostrata in fig. 4.1.

4.1.2 Il protocollo IPSEC

Benché spesso si preferisca utilizzare altre soluzioni (ed in particolare tunnel basati sui protocolli di trasporto, normalmente UDP) dal punto di vista della standardizzazione la soluzione più appropriata per realizzare delle VPN è quella di usare IPSEC, cioè una reimplementazione del protocollo IP, supportata nativamente in IPv6, ed incapsulata in IPv4, che fornisce autenticazione e crittografia direttamente al livello di rete (appunto quello di IP).

L'uso di IPSEC al posto di IP permette infatti, fra le altre cose, di cifrare completamente tutto il traffico fra due macchine su internet, risolvendo così in maniera naturale il problema di comunicare in maniera privata. Dato poi che il protocollo supporta una comunicazione in modalità tunnel fra due router che cifra pure gli indirizzi iniziale e finale dei pacchetti, è altrettanto naturale ottenere una VPN. Il protocollo IPSEC è basato su tre componenti:

AH definito nell'RFC 2405, è l'acronimo di *Authentication Header*, e fornisce un servizio di autenticazione dei pacchetti.

ESP definito nell'RFC 2406, è l'acronimo di *Encapsulating Security Payload*, e fornisce un servizio di autenticazione e cifratura dei dati.

IKE definito nell'RFC 2409 per la versione originaria (IKEv1) ed aggiornato con una seconda versione (IKEv2), definita nell'RFC 5996. è l'acronimo di *Internet Key Exchange*, e

fornisce un servizio per la negoziazione dei parametri necessari al funzionamento dei precedenti.

Di queste tre componenti quella di gran lunga più complessa è IKE, che nasce dalla combinazione di una serie di altri protocolli. In sostanza IKE combina il protocollo ISAKMP (*Internet Security Association and Key Management Protocol* definito dall'RFC 2408) che gestisce la negoziazione delle connessioni e definisce le cosiddette *Security Association*. A questo si aggiunge il DOI (*Domain Of Interpretation*, definito dall'RFC 2407) per IPSEC che completa ISAKMP nel caso specifico di IPSEC, e il protocollo *Oakley* di scelta delle chiavi (definito nell'RFC 2412), che è quello che permette l'impostazione iniziale dei parametri necessari alla comunicazione. Il protocollo è stato in seguito rivisto con la creazione di una seconda versione, la cui specificazione finale (che riprende la definizione originale dell'RFC 4306 ed alcuni chiarimenti successivi dell'RFC 7418) è fornita nell'RFC 5996.

È a questo livello che vengono negoziate le chiavi e gli algoritmi crittografici utilizzati, i tempi di vita delle connessioni, implementando in pratica quel meccanismo di crittografia ibrido descritto alla fine di sez. 1.2.3.

Nel funzionamento di IPSEC giocano un ruolo cruciale le *Security Association* (in breve SA), queste identificano un canale di comunicazione definendone destinazione, protocollo utilizzato (AH o ESP) ed un *Security Parameter Index* (SPI) che distingue tutti i possibili diversi canali che hanno la stessa destinazione e usano lo stesso protocollo. Per quanto abbiamo appena detto è evidente che una SA è unidirezionale (è infatti sempre definita a partire dalla macchina corrente verso una certa destinazione) per cui ne occorrono sempre due (una per direzione) per ogni canale di comunicazione.

Il primo passo per poter utilizzare IPSEC è quello di eseguire IKE per la negoziazione dei parametri della connessione, questo in genere avviene in due fasi; nella prima fase viene negoziata una coppia di SA di ISAKMP per la comunicazione fra due *gateway*, in modo da poter gestire più canali indipendenti; con queste nella seconda fase vengono generate le ulteriori SA per IPSEC (ad uso di ESP o AH) per le singole comunicazioni. Il tutto avviene usando il protocollo UDP sulla porta 500. Pertanto è essenziale al funzionamento di IPSEC che un firewall lasci passare il traffico su questa porta.

IKE è progettato per la massima flessibilità e permette di negoziare separatamente per ciascuna SA un tempo di vita, l'algoritmo crittografico (simmetrico) da utilizzare per la cifratura dei pacchetti, l'algoritmo per l'autenticazione, e l'algoritmo di crittografia asimmetrica usato per lo scambio delle chiavi. Con la versione 2, oltre a vari miglioramenti relativi ai meccanismi di negoziazione, all'affidabilità delle connessione, e alla resistenza ad attacchi di DoS, è stato inserito all'interno del protocollo la gestione del cosiddetto *NAT Traversal* (su cui torneremo più avanti) con l'incapsulazione di sia di IKE che ESP in pacchetti UDP, per i quali è stata riservata la porta 4500. Se pertanto si utilizza IKEv2 è necessario che il firewall consenta il traffico anche su questa porta.

Una volta completata l'impostazione iniziale tramite IKE, tutto il resto della comunicazione avviene direttamente attraverso quello che più propriamente si indica con IPSEC, facendo riferimento solo ai due protocolli AH ed ESP, che effettivamente operano a livello di rete. Infatti IKE è un protocollo di livello superiore che serve a negoziare i parametri usati dagli altri due, l'effettiva comunicazione non avviene attraverso di esso. Da qui in avanti perciò riferendoci ad IPSEC intenderemo soltanto la parte relativa alla comunicazione al livello di rete.

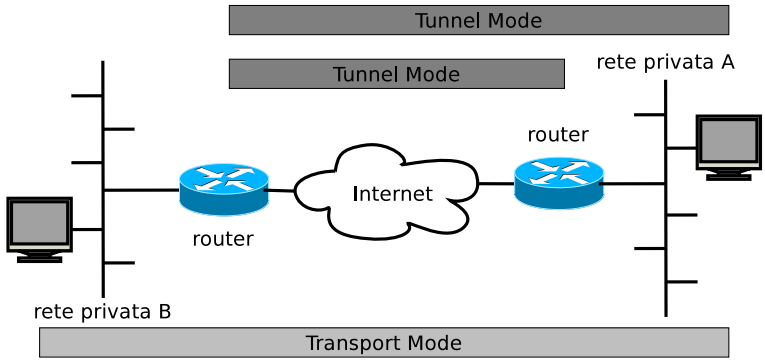


Figura 4.2: Modalità di trasporto dei dati realizzate tramite IPSEC.

Che sia incapsulato in IPv4, o implementato come estensione di IPv6, IPSEC supporta due modalità native, illustrate in fig. 4.2. La prima, detta modalità di trasporto (o *Transport Mode*) prevede la comunicazione diretta da stazione a stazione, ciascuna delle quali tratta i suoi pacchetti e li spedisce tramite IPSEC. La seconda modalità, detta tunnel (o *Tunnel Mode*) prevede invece la presenza di almeno un router (il cosiddetto *security gateway*) che faccia da tramite per la comunicazione con le macchine poste nella rete per la quale agisce appunto da *gateway*.

La modalità tunnel è appunto la modalità tipica con cui si realizza una VPN, e presenta le due possibilità mostrate nella parte alta di fig. 4.2, in cui si può realizzare la VPN per comunicare da una macchina esterna (il *road warrior*) verso il *security gateway* per accedere alla rete dietro di questi, o direttamente fra due *security gateway*, per connettere due reti private.

In generale IPSEC supporta due tipi di comunicazione, autenticata o cifrata, che possono essere usate indipendentemente l'una dall'altra. Però l'uso di una comunicazione cifrata non autenticata è insicuro e soggetto ad attacchi che possono permettere di decifrare i messaggi, per cui non viene mai usata. La comunicazione autenticata avviene attraverso l'uso di AH, mentre la comunicazione cifrata attraverso ESP, che supporta anche, ove richiesto (e come detto questo deve essere sempre fatto) l'autenticazione.

Dei due protocolli usati per la comunicazione effettiva AH serve solo ad autenticare i pacchetti; il meccanismo di funzionamento di AH, nei due casi di modalità trasporto o tunnel, è illustrato in fig. 4.3. Nel caso si usi la modalità trasporto ci si limita a frapporre fra l'intestazione di IP e quella dei protocolli di livello superiore (nell'esempio TCP) l'intestazione di AH, che contiene tutti i dati relativi all'autenticazione.

Se invece si opera in modalità tunnel si crea una nuova intestazione per il pacchetto, che conterrà solo gli indirizzi degli estremi del tunnel, e si autentica tutto il pacchetto originale, replicandolo integralmente dopo l'intestazione di AH. Alla ricezione del pacchetto il router che fa da *security gateway* rimuoverà l'intestazione usata per la trasmissione, verificherà il contenuto e ricostruirà il pacchetto originale, inviandolo a destinazione (in genere nella rete privata posta dietro il router stesso). In questo modo si possono inviare pacchetti attraverso internet anche usando gli indirizzi riservati per le reti private illustrati in sez. 4.1.1.

La procedura di autenticazione cerca di garantire l'autenticità del pacchetto nella massima

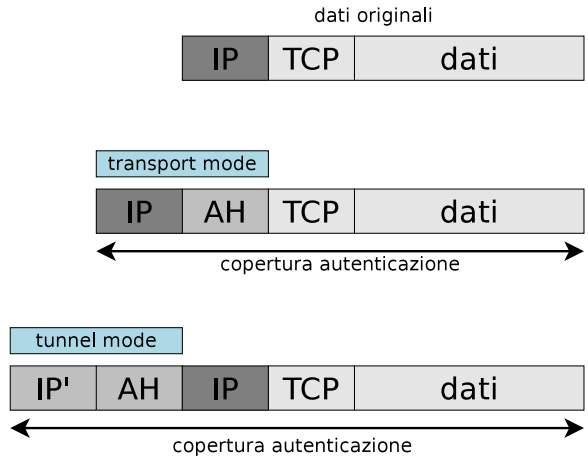


Figura 4.3: Schema di un pacchetto IPSEC autenticato l'uso di AH.

estensione possibile, ma dato che alcuni campi dell'intestazione di IP possono variare in maniera impredicibile alla sorgente, il loro valore non può essere protetto dall'autenticazione. Il calcolo dei dati di autenticazione viene allora effettuato alla sorgente su una versione speciale del pacchetto inviato in cui il TTL nell'intestazione principale è impostato a zero, così come le opzioni che possono essere modificate nella trasmissione. La copertura dell'autenticazione del pacchetto è mostrata sempre in fig. 4.3.

Qualora si debba garantire una trasmissione riservata dei dati, occorre invece usare ESP, lo schema di un pacchetto secondo questo protocollo è illustrato in fig. 4.4; di nuovo se si opera in modalità di trasporto ci si limita a cifrare il contenuto del pacchetto IP, interponendo il risultato fra l'intestazione di ESP ed una coda che chiude la sezione cifrata; a questo si aggiunge poi un'eventuale altra sezione finale che contiene le informazioni di autenticazione; le sezioni del pacchetto coperte da autenticazione e crittografia sono sempre riportate in fig.4.4.

Se invece si opera in modalità tunnel come nel caso di AH si usa una nuova intestazione per IP, cifrando completamente tutto il pacchetto originale. In questo modo non solo si potranno usare nel pacchetto originale indirizzi di reti private, ma in generale si potrà impedire ad un osservatore di conoscere gli indirizzi originali di sorgente e destinazione, dato che rimangono nella parte non cifrata solo quelli dei due *security gateway* che effettuano lo scambio.

Come accennato IPSEC è un protocollo standard ed è quello che viene utilizzato da più tempo per la creazione di VPN; nonostante questo esso non risulta essere molto diffuso e nel tempo sono stati proposti parecchi approcci alternativi. Tutto questo è dovuto al fatto che i vari protocolli che compongono IPSEC hanno una pessima interazione il NAT, per cui la versione originale di IPSEC non è utilizzabile quando i pacchetti devono attraversare un router o un firewall che eseguono un NAT.[3]

[3]il caso può sembrare irrilevante rispetto alle situazioni illustrate in fig. 4.2, dato che basterebbe utilizzare IPSEC sul firewall o sul router dopo aver eseguito il NAT; il problema è che solo una piccola percentuale di router (quelli più evoluti) supportano IPSEC, ma anche così non si risolverebbe il problema, molto comune, di tutti quei client che si vorrebbero collegare come *road warrior* partendo da una rete privata, dato che in tal caso l'indirizzo di partenza dei pacchetti IPSEC deve stare sul client.

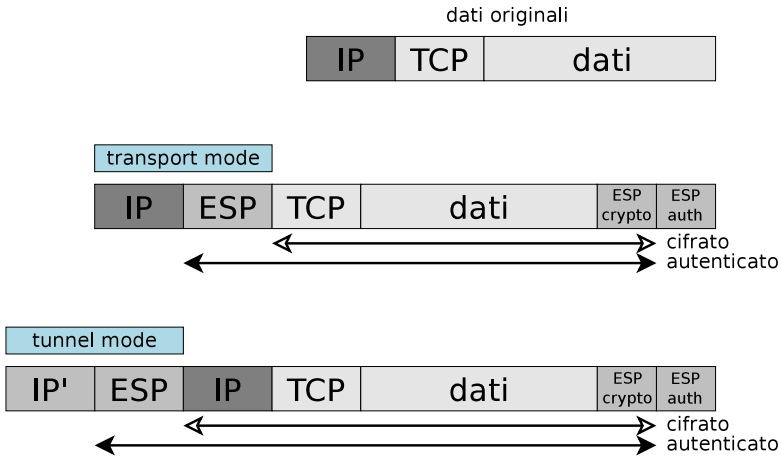


Figura 4.4: Schema di pacchetto IPSEC cifrato con l'uso di ESP.

In particolare AH semplicemente non può di funzionare in caso di NAT. Esso infatti autentica (si riveda fig. 4.3) l'intero pacchetto, compresi gli indirizzi sorgente e destinazione, che verrebbero modificati nell'attraversamento di un NAT, pertanto una volta passato il NAT il pacchetto non risulterebbe più originale. Dato che AH nasce appunto per eliminare la possibilità di ricevere pacchetti modificati, ed il NAT funziona modificando i pacchetti, i due semplicemente non possono lavorare insieme.

Con ESP, anche quando il contenuto è autenticato, il problema è più sottile in quanto in questo caso gli indirizzi esterni (vedi fig. 4.4) non sono inclusi nell'autenticazione, quindi possono essere modificati nel NAT. Restano però una serie di altri problemi; il primo è che quando il pacchetto originale contiene dei pacchetti UDP o TCP la checksum che ne verifica l'integrità è calcolata sugli IP originali, e questa, se il pacchetto è cifrato con IPSEC, non può essere aggiornata, per cui a destinazione essa non corrisponderà; inoltre si pone il problema di come reinviare all'indietro i pacchetti di risposta, dato non ci sono dati permettano di associare in maniera adeguata un certo pacchetto IPSEC ad uno specifico indirizzo interno.[4]

Oltre a questi ci sono poi un'altra serie di problemi legati all'uso di IKE che alla fine rendono impossibile l'uso della versione originale di IPSEC attraverso un NAT. Per questo motivo è stata realizzata una estensione del protocollo chiamata NAT-T (da *NAT Traversal*) che prevede l'incapsulamento dei pacchetti ESP in pacchetti UDP con l'uso della porta 4500, così che a questo punto diventa possibile attraversare normalmente un NAT come avviene per tutti i pacchetti UDP. Questa però è una estensione successiva, e compresa come parte dello standard solo con IKEv2. Inoltre alla fin fine usare questo metodo alla fine rende IPSEC equivalente alle modalità di funzionamento di altre soluzioni (come OpenVPN, che vedremo in sez. 4.3) che lavorano nativamente su UDP.

[4]nel caso di TCP e UDP questo viene fatto tenendo in conto (ed eventualmente modificando) il numero di porta, ma ESP non ha nulla di equivalente, l'unica possibilità potrebbe essere l'uso del campo SPI, che però non può essere modificato; essendo un numero 32 bit la probabilità di collisione sarebbe bassa, ma il punto è che questo è associato ad una SA, ed è quindi diverso a seconda della direzione del pacchetto e non esiste un modo semplice per associare un valore in uscita con uno in ingresso.

4.1.3 Le VPN in *user-space*

Il problema maggiore di FreeS/WAN e di tutti i derivati successivi, come *Openswan*, così come dell'implementazione nativa del kernel 2.6 ed in generale di tutte le VPN basate su IPSEC, è che il loro funzionamento, appoggiandosi su un apposito protocollo del livello di rete, è realizzato all'interno del kernel. Questo comporta in genere una procedura di installazione molto più complessa (in caso di necessità di modifiche o aggiornamenti si tratta di reinstallare il kernel e riavviare la macchina), ed inoltre in caso di vulnerabilità fornisce ad un attaccante il massimo dei privilegi possibili, ed il massimo del rischio, dato che un problema può far crollare l'intero sistema.

Se a questo si aggiunge che né FreeS/WAN né *Openswan* brillavano per semplicità di installazione e configurazione, si comprende come si siano cercate delle alternative che permettessero di implementare una VPN in *user-space*, utilizzando dei normali programmi che implementassero il tunnel attraverso una connessione cifrata appoggiandosi ad un normale protocollo del livello di trasporto (in genere tramite socket UDP), risolvendo così anche i problemi relativi all'attraversamento di firewall e NAT che sono presenti con l'uso di IPSEC.

Una tale alternativa presenta una lunga serie di innegabili vantaggi: gli aggiornamenti sono molto più semplici (basta reinstallare un programma e riavviare il servizio), è possibile far girare il servizio per conto di utenti non privilegiati (riducendo così i rischi in caso di compromissione o di bug), si evitano completamente le problematiche relative all'attraversamento di un NAT che avviene come per qualunque altro servizio basato su socket.

In genere questo tipo di prodotti si basano sull'uso di SSL, che è un protocollo ampiamente noto e ben controllato, e che non presenta le problematiche di gestione di IPSEC, che è un protocollo molto complesso che se non è configurato correttamente rischia di perdere le sue caratteristiche di sicurezza. In questo caso si tratta in genere di fare passare i dati della VPN attraverso una connessione protetta con SSL.

Capita spesso però che i vantaggi in semplicità si traducano in una diminuzione del livello di sicurezza; questo è vero in particolare per tutte quelle VPN in *user-space* che sono realizzate con un programma che permette di aprire una connessione verso un server centrale, limitandosi semplicemente a cifrare la comunicazione con SSL. Questo può rendere la configurazione anche molto semplice, ma si perde così però uno dei cardini di IPSEC, che è quello di poter identificare con certezza l'uno con l'altro i due capi della connessione.

In altri casi invece si spaccia per VPN quello che è un banalissimo tunnel relativo ad un singolo protocollo, cioè al fatto di far passare il traffico verso certi servizi per una connessione SSL, redirigendo il traffico di una o più porte su un canale cifrato come visto in sez. 2.1.3 per `stunnel`. In questo caso manca completamente la capacità, propria di una vera VPN, di dare un accesso all'intera gamma dei servizi di rete.

È pertanto essenziale nella scelta di una VPN in *user-space*, capire quali sono le modalità in cui questa viene realizzata, e quale può essere la sua utilizzabilità su diverse piattaforme, dato che in questo caso non esiste una standardizzazione di un protocollo come quella di IPSEC. Come vedremo in sez. 4.3 OpenVPN rappresenta una valida soluzione per la realizzazione di VPN in *user-space*, con funzionalità che non hanno nulla da invidiare rispetto a nessun altro prodotto, tanto è diventata praticamente uno standard presente in ogni distribuzione GNU/Linux.

4.2 La gestione di VPN con *Openswan* e *strongSwan*

Vedremo in questa sezione come realizzare una VPN con IPSEC, sia per la parte di comunicazione di rete tramite i protocolli AH ed ESP, che per la gestione di tunnel cifrati con IKE, facendo riferimento ad una delle diverse possibili soluzioni, quella realizzata dal pacchetto *Openswan*. Dato che la realizzazione di IPSEC su Linux ha avuto una storia travagliata inizieremo con una panoramica generale sull'argomento.

4.2.1 Quale IPSEC per Linux

Linux ha acquisito una implementazione ufficiale, (presente cioè nella versione standard del kernel) di IPSEC (o meglio dei protocolli AH e ESP) a partire dai kernel della serie 2.6, di questa esiste anche un *backport* non ufficiale per i kernel della serie 2.4. Questa implementazione, denominata *NETKEY*, è stata realizzata rifacendosi alla architettura della analoga implementazione realizzata dal progetto KAME per i vari FreeBSD, OpenBSD, ecc.

In precedenza però era già presente una implementazione del protocollo IPSEC realizzata dal progetto FreeS/WAN. Questo progetto, oltre ad una implementazione dei protocolli AH ed ESP, realizzata come patch al kernel ufficiale e denominata *KLIPS*, forniva anche un demone per l'uso di IKE ed il supporto per una modalità di funzionamento, detta *Opportunistic Encription* (abbreviata in OE) che fosse in grado di far utilizzare automaticamente IPSEC per ogni comunicazione con altre macchine in grado di utilizzare detto protocollo.[5]

Gli scopi del progetto erano essenzialmente di protezione della privacy (da cui derivava lo sviluppo della parte relativa alla *Opportunistic Encription*), con una forte attenzione agli aspetti politici. All'epoca il software FreeS/WAN veniva sviluppato al di fuori dagli Stati Uniti in quanto quel paese non consentiva l'esportazione di software crittografico. Nel Gennaio del 2000 i regolamenti vennero cambiati e le esportazioni (eccettuato quelle verso alcuni paesi considerati terroristi) consentite.

Dato però che non vi era certezza legislativa (il permesso resta di natura amministrativa e può essere revocato in qualunque momento) i leader del progetto FreeS/WAN continuarono a rifiutarsi di accettare contributi da cittadini statunitensi per essere assolutamente certi di evitare problemi di natura legale. Questo, insieme ad altre considerazioni di natura tecnica, e agli immancabili attriti personali, creò un conflitto con gli sviluppatori del kernel (che ritenevano accettabili contributi da cittadini statunitensi e troppo pesanti i cambiamenti apportati allo stack TCP/IP del kernel) che finirono per reimplementare indipendentemente i protocolli AH ed ESP.

In ragione di questa situazione e della percezione di uno scarso successo dell'obiettivo principale del progetto, che rimaneva quello della diffusione capillare della *Opportunistic Encription*, nel Marzo del 2004 il team originale decise di cessare lo sviluppo di FreeS/WAN. Trattandosi di software libero il codice venne immediatamente ripreso e con la nascita di altri progetti come *strongSwan* e *Openswan* che ne hanno proseguito lo sviluppo, aggiungendo al contempo un gran numero di funzionalità, come il *NAT traversal* e l'autenticazione tramite certificati, che fino ad allora erano state sviluppate esternamente.

Così dal lato del kernel ci si è trovati così di fronte a diverse implementazioni dei protocolli di rete necessari. Però, come illustrato in sez. 4.1.2, per la creazione di una VPN con IPSEC

[5] l'idea era di inserire opportuni record con chiavi pubbliche nel DNS che se ottenuti avrebbero consentito la creazione automatica di un canale cifrato con la destinazione.

non è sufficiente disporre di una implementazione dei protocolli AH ed ESP nel kernel, ma occorre anche utilizzare una implementazione del protocollo IKE per garantire autenticazione dei capi della comunicazione ed uno scambio sicuro delle chiavi crittografiche. Benché questo tipo di supporto sia stato parzialmente introdotto al di fuori di FreeS/WAN portando su Linux i programmi di gestione ripresi dal progetto KAME (che fanno parte dei cosiddetti `ipsec-tools`) l'implementazione presente in FreeS/WAN era nettamente più evoluta ed è stata ereditata dai suoi successori.

Pertanto al momento sono disponibili una larga serie di combinazioni di elementi con cui è possibile creare una VPN basata su IPSEC; nel seguito si è preferito fare riferimento alle implementazioni derivate da FreeS/WAN (*Openswan* e *strongSwan*),[6] per le maggiori funzionalità da essi fornite (in particolare per *strongSwan* che supporta pienamente anche la versione 2 del protocollo IKE) rispetto agli strumenti di `ipsec-tools`.

Inizialmente i due progetti citati erano sostanzialmente equivalenti, differenziandosi in seguito sulle funzionalità aggiunte e sul focus dello sviluppo, più centrato sulle funzionalità di autenticazione e identificazione il primo, che ha aggiunto numerose modalità di autenticazione delle VPN e diversi algoritmi di cifratura, più sulle tecnologie di rete il secondo, che continua a sviluppare per il 2.6 anche l'implementazione originale di IPSEC (*KLIPS*) del progetto FreeS/WAN.

4.2.2 Configurazione iniziale di *strongSwan* e *Openswan*

La principale differenza nelle versioni recenti di *Openswan* e *strongSwan* è che il primo continua a supportare anche per i kernel della serie 2.6 l'implementazione originaria (KLIPS) del protocollo IPSEC realizzata dal progetto FreeS/WAN, mentre *strongSwan* usa soltanto il supporto della implementazione standard del kernel anche se è in grado di usare KLIPS su un kernel 2.4.

Dato che lo sviluppo di KLIPS è comunque destinato a seguire sempre con un certo ritardo l'evoluzione del kernel, problema che non si pone rispetto all'implementazione nativa, non prenderemo in considerazione le problematiche relative alla sua installazione facendo riferimento al supporto standard (NETKEY) che è presente su tutti i kernel delle principali distribuzioni.[7] Per questo il resto di questo paragrafo tratterà solo gli aspetti del supporto IPSEC in user-space, vale a dire dei programmi che implementano il protocollo IKE.

Fra le due alternative *strongSwan* sembra quella sviluppata in maniera più attiva: ha il supporto completo per IKEv2, fornisce una documentazione molto più aggiornata e sembra orientata verso nuove direzioni di sviluppo. Tratteremo pertanto principalmente questa, illustrando brevemente, dove possibile, le eventuali differenze con *Openswan*, anche se, a causa dell'origine comune, i nomi dei file di configurazione e la relativa sintassi, molte direttive, i comandi di gestione e le relative opzioni, sono gli stessi anche per *Openswan*.

Essendo disponibili i pacchetti per le principali distribuzioni non staremo a trattare le modalità di installazione; nel caso di Debian i pacchetti forniscono i programmi per la gestione di IKE, lo script di avvio (`/etc/init.d/ipsec`), ed il programma di controllo `ipsec`. In entrambi i

[6] che nel caso di Debian vengono distribuite direttamente negli omonimi pacchetti `openswan` e `strongswan`.

[7] nel caso di Debian comunque con *Openswan* viene distribuito il supporto di KLIPS con i due pacchetti `openswan-modules-source` e `openswan-modules-dkms` che consentono una ricompilazione dello stesso come modulo in maniera relativamente semplice, mentre il kernel 2.4 contiene già integrato il backport di NETKEY.

casi con il pacchetto viene installato uno scheletro di file di configurazione.[8] La configurazione dipende infatti da quali sono i canali IPSEC che si vogliono stabilire e dalle modalità di autenticazione che si vogliono usare per la loro creazione (che tratteremo a parte in sez. 4.2.3), e dovrà essere completata manualmente.

Nel caso di *strongSwan*, che come accennato fornisce anche una implementazione completa della versione 2 del protocollo IKE, oltre al classico demone `pluto` ereditato da FreeS/WAN, che gestisce la versione 1 del protocollo, viene installato anche un secondo demone, `charon`, sviluppato da zero per questo scopo, che prevede alcune configurazioni specifiche non contemplate da *Openswan*. Questo non è disponibile per *Openswan* che usa una sua versione modificata di `pluto` che supporta alcune funzionalità di base di IKEv2.

Il file di configurazione principale è `/etc/ipsec.conf`, la cui sintassi è descritta in dettaglio nella relativa pagina di manuale, accessibile con `man ipsec.conf`. Qualora si usi una versione di *Openswan* superiore alla 2.0 la prima riga non commentata del file deve essere `version 2.0`, in quanto con il passaggio a tale versione sono cambiate alcune opzioni di configurazione,[9] ed occorre perciò segnalare esplicitamente quale sintassi si sta utilizzando. Tutto ciò non è necessario per *strongSwan* che non supporta versioni precedenti del file di configurazione.

Il file contiene tutte le informazioni di configurazione necessarie al funzionamento di una VPN, ad eccezione delle informazioni di autenticazione reciproca che, come vedremo in sez. 4.2.3 sono mantenute in `ipsec.secrets`. Per questo di norma il file non contiene informazioni riservate, e può essere tenuto accessibile in lettura.[10] Si possono comunque raccogliere eventuali configurazioni che si vogliono mantenere private in singoli file separati (senza permesso di lettura) che possono essere inclusi in `ipsec.conf` tramite la direttiva `include`, questa richiede che si specifichi un nome di file e supporta anche inclusioni multiple con l'uso del carattere jolly "`*`".

Il formato di `ipsec.conf` segue le convezioni standard per cui righe vuote e quanto segue il carattere "`#`" viene ignorato. Il suo contenuto è suddiviso in in sezioni, che sono introdotte da una riga nella forma "`tipo nome`", dove `tipo` è una parola chiave riservata che indica il tipo della sezione, e `nome` è un identificativo scelto dall'utente per distinguere sezioni dello stesso tipo.

Si tenga presente che la dichiarazione di una sezione, così come la direttiva `include`, deve sempre iniziare all'inizio di una riga; il contenuto della sezione infatti viene identificato dal fatto che le righe che ne fanno parte iniziano almeno con uno spazio; questo consente di avere una visione immediata del contenuto delle varie sezioni usando semplicemente l'indentazione.

I tipi di sezione disponibili sono tre; il primo tipo è `config`, che viene usata per impostare la configurazione di avvio dei vari demoni ed al momento questa sezione ha un unico nome possibile che è `setup`. Il secondo tipo di sezione è `conn` che identifica una connessione fra i due capi di una VPN, ed il cui nome è un qualunque identificativo associato alla stessa. Il terzo (disponibile solo per *strongSwan*) è `ca`, che permette di specificare le proprietà di una *Certification Authority*. Si è riassunta la situazione in tab. 4.1 dove si sono riportate (nella seconda parte) anche le altre direttive che pur non definendo sezioni devono essere utilizzate in maniera simile (cioè scritte senza indentazione).

[8] nel caso di *Openswan debconf* presenta una schermata in cui si chiede se usare certificati per l'identificazione (con la possibilità successiva di crearne o importarne uno) o lasciare la cosa non configurata.

[9] questo fa differenza solo nel caso di vecchie Debian, in quanto in *woody* è presente la versione 1.96 di FreeS/WAN, che usa la vecchia sintassi, tutte le distribuzioni recenti usano la nuova sintassi e specificano questo valore.

[10] l'unico caso in cui questo non è vero era con il cosiddetto *manual keying*, vedi quanto detto all'inizio di sez. 4.2.3.

Tipo	Significato
`config`	Indica una sezione di configurazione, prevede obbligatoriamente come identificativo il valore `setup`, e viene utilizzata per le impostazioni di avvio dei demoni.
`conn`	Identifica una sezione di configurazione di una connessione VPN, richiede un identificativo arbitrario purché univoco.
`ca`	Identifica una sezione che consente di applicare valori specifici ad una *Certification Authority* (valida solo per *strongSwan*).
`include`	Consente di includere il contenuto del file passato come identificativo (pertanto non è propriamente un identificatore di sezione).
`version`	Indica la versione della sintassi del file di configurazione (valida solo, e da specificare obbligatoriamente, per *Openswan*).

Tabella 4.1: Parole chiave identificanti i tipi di sezione utilizzabili in `ipsec.conf`.

Il contenuto di ogni sezione è poi costituito da una serie di direttive di assegnazione dei valori per i valori dei parametri che sono definiti in quella sezione, da esprimere nella forma `parametro=valore`, esiste però un parametro speciale, `also`, valido per tutti i tipi di sezione, che quando viene assegnato al nome di un'altra sezione consente di aggiungere le definizioni dei parametri in essa presenti alla sezione in cui viene utilizzato, come se questi venissero scritti al suo posto. L'uso di `also` può essere nidificato, e lo si può anche utilizzare più volte purché non ci sia mai un riferimento multiplo alla stessa sezione. La sezione a cui si fa riferimento deve ovviamente esistere, ed essere specificata *dopo* quella in cui lo si utilizza.

Per le sezioni di tipo `ca` e `conn` esiste inoltre il nome riservato `%default` che serve a configurare i valori di default che saranno automaticamente assegnati (qualora non sovrascritti da altre assegnazioni esplicite) ai parametri di quel tipo di sezione.[11] Queste devono essere comunque specificate prima delle sezioni normali, inoltre non si può usare il parametro `also` in una di queste sezioni.

La prima parte della configurazione è sempre costituita dalla sezione `config`, un esempio del suo contenuto, ripreso dal file `ipsec.conf` installato con *strongSwan* su una Debian Squeeze come scheletro di configurazione è il seguente:

ipsec.conf

```
config setup
        # plutodebug=all
        # crlcheckinterval=600
        # strictcrlpolicy=yes
        # cachecrls=yes
        # nat_traversal=yes
        charonstart=yes
        plutostart=yes
```

Come accennato la sezione ha come unico nome utilizzabile `setup`, e vi si richiede semplicemente l'avvio dei demoni `charon` e `pluto` con i due parametri `charonstart` e `plutostart`, tenendo

[11] questo nome veniva usato per impostare correttamente le connessioni con le versioni di FreeS/WAN precedenti la 2.0, a partire da questa versione infatti i valori di default sono stati corretti e questo tipo di dichiarazione non viene sostanzialmente più usato.

per buoni i valori di default di tutti gli altri parametri.[12] Nello scheletro resta commentata la predisposizione di alcuni dei parametri di controllo, si sono riportati in tab. 4.2 quelli ritenuti più rilevanti, per un elenco completo si deve fare riferimento alla pagina di manuale di `ipsec.conf`.[13]

Parametro	Significato
`plutodebug`	Stabilisce quale livello di messaggi di debug di `pluto` deve essere registrato. Se vuoto o con il valore `none` indica nessun messaggio, `all` indica la massima prolissità; altri valori sono illustrati nella pagina di manuale accessibile con `man ipsec_pluto` (o `man pluto`).
`plutostart`	Richiede l'avvio del demone `pluto` per IKEv1, può assumere i valori `yes` e `no`.
`charonstart`	Richiede l'avvio del demone `charon` per IKEv2, può assumere i valori `yes` e `no`.
`nat_traversal`	Indica se accettare e offrire il supporto per il *NAT Traversal*. Prende come valori `yes` o `no` (che è il default). Questo parametro è rispecificabile anche per le singole connessioni. Nel caso di *strongSwan* si applica solo a `pluto` per IKEv1, il supporto per il *NAT Traversal* è automatico con IKEv2.
`keep_alive`	Indica in secondi l'intervallo di tempo da usare per l'invio di pacchetti che mantengano attiva la comunicazione sul canale UDP usato dal *NAT Traversal* onde evitare di perdere lo stato della stessa, il default è 20.
`cachecrls`	Indica se deve essere mantenuta in cache la lista dei certificati revocati (CRL), applicabile solo a `pluto` (per IKEv2 la cache è sempre attiva), i valori possibili sono `yes` o `no` (che è il default).
`crlcheckinterval`	Specifica il tempo, in secondi in cui viene rieseguita la acquisizione della lista dei certificati revocati (applicabile solo a `pluto`).

Tabella 4.2: I principali parametri disponibili nella sezione `config` di `ipsec.conf`.

Il grosso della configurazione si effettua comunque all'interno di sezioni di tipo `conn`; ciascuna di esse infatti indica una connessione di rete che deve essere stabilita attraverso IPSEC, ed è con questa direttiva che si imposta una VPN, specificando tutti i parametri necessari alla comunicazione fra i due capi della stessa. Questo significa anche che per ognuna di queste sezioni presente su un capo della VPN, ci deve una sezione corrispondente nel file `ipsec.conf` che sta sulla macchina all'altro capo della connessione.[14]

Un esempio di un paio di sezioni di configurazione per delle connessioni è il seguente, che si è ottenuto dalla versione di `ipsec.conf` installata come scheletro di configurazione di *strongSwan* su una Debian Squeeze, si sono rimossi i commenti che sono presenti nell'esempio per una maggiore leggibilità:

ipsec.conf

```
# Sample VPN connections
```

[12] nel caso di *Openswan* questa sezione è totalmente diversa; non sono previste le direttive di avvio dei singoli demoni in quanto esiste solo `pluto`, mentre va specificato quale implementazione dei protocolli di rete usare con il parametro `protostack`, impostato ad `auto` per indicare al programma di provare prima con NETKEY e poi con KLIPS.

[13] si tenga presente che i parametri sono diversi per *Openswan* e *strongSwan*.

[14] questo ovviamente nell'ipotesi che si usi lo stesso software da ambo le parti.

```
conn sample-self-signed
      left=%defaultroute
      leftsubnet=10.1.0.0/16
      leftcert=selfCert.der
      leftsendcert=never
      right=vpn.truelite.it
      rightsubnet=10.2.0.0/16
      rightcert=peerCert.der
      auto=start

conn sample-with-ca-cert
      left=%defaultroute
      leftsubnet=10.1.0.0/16
      leftcert=myCert.pem
      right=vpn2.truelite.it
      rightsubnet=10.2.0.0/16
      rightid="C=CH, O=Linux strongSwan CN=peer name"
      keyexchange=ikev2
      auto=start
```

Dato che la configurazione deve essere equivalente sui due capi della connessione, all'interno di una sezione `conn` questi non vengono identificati in termini di locale e remoto (altrimenti occorrerebbe invertire i ruoli passando da un capo all'altro), ma in termini di *destra* e *sinistra*. I due parametri principali (ed obbligatori) da impostare nella creazione di una VPN sono infatti `left` e `right`, che identificano gli indirizzi IP pubblici dei due *security gateway* (vedi fig. 4.1) fra cui si crea la VPN. Questi devono essere specificati in notazione dotted decimal oppure con il nome a dominio completo (FQDN).

Quale dei due capi si usi per indicare la macchina locale e quale la macchina remota in genere non ha importanza, all'avvio i programmi controllano il proprio indirizzo IP e sono in grado di identificare nella configurazione quale fra *destra* e *sinistra* è la loro parte. Solo quando questo non è possibile viene considerato locale `left` (seguendo la convenzione mnemonica delle iniziali di *locale* e *remoto*).

Si può così configurare la connessione su uno dei due capi e poi copiare la relativa sezione nel file di configurazione all'altro estremo. Questo comporta che un gran numero di parametri di configurazione sono doppi, esistono cioè nella versione il cui nome inizia con `left` ed in quella in cui inizia con `right`, il loro significato è identico, ma si applica soltanto al rispettivo capo della connessione.

Oltre alla indicazione esplicita degli indirizzi IP, si possono usare per `left` e `right` anche dei valori speciali che permettono un'assegnazione dinamica; ad esempio con il valore `%defaultroute` si assegna automaticamente come indirizzo quello associato all'interfaccia da cui si esce verso internet. Ovviamente questo può essere usato solo su uno due capi della VPN, nella configurazione dell'altro si dovrà per forza indicare l'indirizzo IP pubblico.[15]

Con il valore `%any` si indica invece che l'indirizzo IP di quel capo (sottintendendo che si tratta del capo remoto) sarà scelto automaticamente in fase di connessione. Si usa in genere questo valore per identificare l'indirizzo di un *road warrior* che si collega verso un concentratore di VPN, che in genere eseguirà la connessione da IP assegnati dinamicamente da provider e non

[15] se lo si usa cioè non si avrà più la simmetria dei due file di configurazione presenti sui due capi della configurazione.

noti a priori; in tal caso nella fase di negoziazione delle chiavi sarà anche acquisito l'IP della macchina remota che verrà automaticamente utilizzato per l'invio dei pacchetti.[16]

Un secondo parametro fondamentale è `leftsubnet` (e l'equivalente `rightsubnet`) con cui si specifica la rete privata che sta dietro il *security gateway*, da indicare in notazione CIDR. Questo parametro consente di rendere accessibile detta rete dall'altro capo della VPN. Se non lo si specifica viene presa come rete il singolo IP specificato con `left`, restringendo soltanto ad esso le connessioni via VPN (normalmente questo è quanto si fa sul lato di un *road warrior*).

Il parametro `auto` serve invece a stabilire quali operazioni effettuare all'avvio di IPSEC e corrisponde ad alcune opzioni dell'uso del comando di controllo `ipsec auto` che è quello usato per avviare e fermare manualmente le connessioni; l'uso del valore `add` aggiunge le informazioni sulla connessione al database interno di `pluto`, ma non avvia la connessione, per farlo infatti occorre indicare il valore `start`, il valore di default è invece `ignore` che non avvia automaticamente nessuna operazione.

Una lista degli altri parametri più importanti delle sezioni di tipo `conn` è riportata in tab. 4.3; un elenco più dettagliato e completo, con la descrizione dei relativi significati, si trova al solito nella pagina di manuale di `ipsec.conf`. Per brevità si sono riportati solo i parametri "`left`", dando per scontato l'esistenza e l'equivalenza dei corrispettivi "`right`".

Si tenga presente inoltre che *strongSwan* a partire dalla versione 4.2.1 ha iniziato ad utilizzare un file di configurazione ausiliario, `/etc/strongswan.conf`, con una sintassi molto diversa. La configurazione per l'uso di IPSEC resta in `ipsec.conf`, ma in questo file verranno inserite le configurazioni delle varie estensioni fornite del programma. Al momento è usato principalmente per il demone `charon`, e ad esempio nella versione installata su una Debian Squeeze l'unico parametro impostato è il numero di thread con cui viene fatto partire questo programma.

4.2.3 Autenticazione e gestione delle credenziali con IPSEC

Come illustrato in sez. 4.1.2 la cifratura e la autenticazione dei pacchetti di rete del protocollo IPSEC si basa sull'uso di appropriate chiavi di cifratura con cui questi vengono generati. Se ci si limita a questo livello (cioè a ESP ed AH) tutto quello che è necessario è fornire al kernel una di queste chiavi in maniera opportuna, che è quello che può essere fatto, senza tirare in ballo l'uso di IKE (e quindi anche senza *strongSwan* o *Openswan*) con il cosiddetto *manual keying*.

In questo caso non si fa scegliere la chiave di sessione ad IKE, ma essa viene specificata manualmente dall'amministratore impostandola direttamente attraverso uno *shared secret*. Questa modalità è poco sicura in quanto per cifrare viene sempre usata la stessa chiave, la scoperta della quale compromette tutto il traffico presente e passato. Essa deve essere utilizzata solo se costretti da compatibilità con altre implementazioni di IPSEC che supportano unicamente questa

[16] la configurazione standard su un *road warrior* vede cioè l'uso di `%defaultroute` su un capo (quello locale) per ottenere l'IP dinamicamente, e dell'indirizzo del server sull'altro, quest'ultimo invece avrà `%any` sul capo che identifica il *road warrior* e potrà avere `%defaultroute` o direttamente l'indirizzo IP sul capo che identifica se stesso.

[17] questa funzionalità è fornita per compatibilità con le versioni precedenti di FreeS/WAN che non supportavano l'uso di certificati SSL ma solo di chiavi RSA; nel caso di *Openswan* questo valore può essere ottenuto tramite il comando `ipsec showhostkey`, non presente in *strongSwan*; vista la maggiore chiarezza e semplicità di configurazione è comunque consigliato sempre indicare un file contenente la chiave pubblica (per la generazione si possono usare i comandi `ssh-keygen` o `openssl rsa`).

Parametro	Significato
`left`	Indica l'indirizzo IP pubblico di uno dei capi della connessione.
`leftsubnet`	Indica la rete privata posta dietro uno dei capi della connessione.
`leftupdown`	Indica uno script da lanciare per modificare il routing o le regole di firewall quando la connessione cambia stato, il default usa `ipsec _updown`.
`leftcert`	Indica il file che contiene il certificato da usare per la propria identificazione.
`leftrsasigkey`	Indica la chiave pubblica RSA da usare per la propria identificazione, il valore di default è `%cert` che indica che deve essere estratta da un file (se ne può anche scrivere direttamente il valore).[17]
`leftsendcert`	Indica le modalità di invio di un certificato all'altro capo a scopo di identificazione, il default è `ifasked` che lo invia su richiesta, si possono usare anche `yes` o `no` (o gli equivalenti `always` e `never`), quest'ultimo viene in genere utilizzato quando si usa una identificazione senza *Certification Authority* in cui si installano esplicitamente le chiavi pubbliche ai due capi della connessione.
`leftid`	Indica la modalità con cui si accerta l'identità dell'altro capo, può essere l'indirizzo IP, un nome a dominio completo (se preceduto dal carattere "@"), o una stringa da confrontare con le informazioni presenti un certificato.
`authby`	Indica la modalità di autenticazione usata per l'identificazione dei capi della VPN, il valore di default è `pubkey`, che indica l'uso di certificati o chiavi asimmetriche, alternativamente si può usare `secret` (o `psk`) per l'uso di un segreto condiviso, le versioni recenti supportano ulteriori meccanismi, deprecato per IKEv2.
`leftauth`	Indica la modalità di autenticazione il cui uso è richiesto sul capo locale della connessione (usare `rightauth` per il capo remoto), prende gli stessi valori di `authby`, supportato solo da IKEv2.
`auto`	Indica le operazioni da effettuare all'avvio di IPSEC, i valori possibili sono `ignore` che ignora la connessione, `add` che la predispone senza avviarla, `route` che predispone anche il routing e `start` che la avvia sempre automaticamente.

Tabella 4.3: I principali parametri generali di una sezione `conn` di `ipsec.conf`.

modalità di funzionamento. Questa modalità non è supportata da *strongSwan* che suggerisce allo scopo l'uso del comando `setkey` degli `ipsec-tools`.[18]

L'uso di IKE consente invece di realizzare il cosiddetto *automatic keying*, vale a dire impostare in maniera automatica, e cambiare periodicamente (il cosiddetto *rekeying*), le chiavi di cifratura dei pacchetti, così da rendere più sicure le comunicazioni. Inoltre nel realizzare questo procedimento IKE supporta anche le modalità per identificare opportunamente l'altro capo della connessione, onde evitare la possibilità di un attacco del tipo *man-in-the-middle* (si riveda quando spiegato in sez. 1.2.3) ed evitare l'accesso di estranei alle proprie VPN.

[18] con *Openswan* il supporto per il *manual keying* è disponibile, ma non lo prenderemo neanche in considerazione, essendo questa una modalità di utilizzo insicura, scomoda e totalmente sconsigliata.

Inizialmente il supporto di IKE fornito da FreeS/WAN prevedeva soltanto due modi per effettuare questo riconoscimento, l'usi di coppie di chiavi RSA (analoghe a quelle usate da `ssh`) in cui è necessario scambiare le chiavi pubbliche fra i due capi della connessione o l'uso di una chiave condivisa (denominata *Pre-Shared Key* o PSK) che deve essere la stessa su tutti e due.[19] In entrambe le modalità tutti i dati privati relativi alla gestione delle VPN venivano mantenuti nel file `/etc/ipsec.secrets` che per questo deve essere proprietà dell'amministratore ed accessibile solo da questi. In particolare andavano inserite in questo file le chiavi private RSA,[20] o le *Pre-Shared Key*.

Quando con le implementazioni successive è stata effettuata l'integrazione delle estensioni che consentono di utilizzare certificati SSL, sia nella forma si certificati autofirmati (sostanzialmente equivalente all'uso di chiavi RSA, formato dei file a parte) che di certificati rilasciati da una *Certification Authority*, il ruolo di `/etc/ipsec.secrets` come unica fonte dei dati privati è finito. Visto infatti che chiavi e certificati vengono in genere mantenuti in file di opportuno formato (ad esempio *strongSwan* supporta sia il formato testuale PEM che il binario DER) ormai non è più necessario inserire direttamente i dati delle chiavi private all'interno di `/etc/ipsec.secrets`,[21] e si può semplicemente usare un riferimento ai file che le contengono.

Questo ha fatto sì che una parte dei dati di autenticazione sia stata spostata nella directory `/etc/ipsec.d/`, in particolare per quanto riguarda la possibilità di mantenere in opportune sottodirectory della stessa chiavi e certificati; quelli dei propri corrispondenti, in caso di certificati autofirmati, o quelli delle *Certification Authority*, se le si usano.

Oltre a queste modifiche nel caso di *strongSwan* sono stati introdotti anche ulteriori metodi di autenticazione, cosa che ha comportato altri cambiamenti. Il formato generale di `/etc/ipsec.secrets` resta comunque lo stesso, si tratta in sostanza di una tabella di voci ciascuna delle quali è composta da una lista di indicatori di selezione, seguiti dalla specificazione del segreto da usare per gestire le connessioni che li riguardano. Lista e segreto devono essere separati dal carattere "`:`" seguito o da uno spazio o da un ritorno a capo, poi dall'identificatore del tipo di segreto (si sono riportati i valori possibili in tab. 4.4) a sua volta seguito da uno spazio e dall'opportuna indicazione del segreto stesso.

Ogni voce deve iniziare al margine sinistro del file, ma può proseguire su più righe, nel qual caso però le righe successive la prima devono essere rientranti ed indentate. Si possono inserire nel file delle righe di commento facendole iniziare con il carattere "`#`", ed inoltre si può specificare una direttiva `include` che permette di includere il contenuto del file specificato, per il nome del quale si può usare anche il carattere jolly "`*`" che sarà espanso con le regole del *filename globbing* dalla shell.

Gli indicatori della prima parte di ciascuna voce sono usati per selezionare quale segreto deve essere usato per identificare i capi di una VPN. Il significato dei selettori dipende dal tipo di segreto,[22] ma per chiavi e certificati indicano gli indirizzi dei capi corrispondenti possono

[19] questa chiave non va confusa con lo *shared secret* di cui si è parlato a proposito del *manual keying*, in questo caso la *Pre-Shared Key* serve solo per determinare l'autenticità dei due capi della VPN con IKE e non viene usata per cifrare il traffico, per il quale viene comunque usato l'*automatic keying* e delle chiavi di sessione gestite dal protocollo che vengono continuamente rinnovate.

[20] in un opportuno formato testuale; per *Openswan* questo è ancora possibile.

[21] con *strongSwan* ad esempio questo non è più possibile, e pertanto non lo prenderemo neppure in considerazione, essendo molto più semplice ricorrere all'indicazione dei file.

[22] la sintassi purtroppo è ambigua e confusa e nasce del sovrapporsi di diversi metodi, ed è inoltre pure scarsamente documentata, se non per esempi; si può considerare questo file e la sua documentazione come un ottimo

Identificatore	Significato
`PSK`	Un segreto condiviso (*Pre-Shared Key*) da utilizzare come chiave simmetrica su entrambi i capi della connessione, da specificare come stringa di caratteri delimitate da virgolette (sono validi tutti i caratteri tranne il ritorno a capo e le virgolette stesse).
`RSA`	Una chiave RSA o la chiave di un certificato X.509, da specificare tramite pathname assoluto o come pathname relativo a `/etc/ipsec.d/certs/`, se la chiave è protetta da password la si può specificare come secondo argomento o richiederne l'immissione da terminale all'avvio con `%prompt`.
`ECDSA`	Specifica una chiave privata ECDSA, usa la stessa sintassi usata per `RSA`.
`EAP`	Definisce delle credenziali per l'autenticazione con il protocollo EAP; richiede di specificare una password di autenticazione come stringa di caratteri delimitate da virgolette, deve essere usato come indicatore il nome utente (utilizzabile solo con IKEv2).
`XAUTH`	Definisce delle credenziali per XAUTH; richiede di specificare una password di autenticazione come stringa di caratteri delimitate da virgolette, deve essere usato come indicatore il nome utente (utilizzabile solo con IKEv2).
`PIN`	Definisce un PIN da usare per una smartcard, non è previsto l'uso di un indicatore.

Tabella 4.4: Identificatori del tipo di segreto utilizzabili all'interno di `ipsec.secrets`.

essere espressi come numeri IP (in notazione *dotted decimal*) come nome di dominio qualificati (preceduti dal carattere "`@`"). Se si usano le *Pre-Shared Key* non è consigliabile fare riferimento a nomi a dominio, dato che non è detto che ci si possa sempre fidare delle risoluzioni del DNS, questo non vale quando si usano segreti a chiave asimmetrica in quanto in tal caso l'identificazione è basata direttamente sulla chiave o sul certificato. Per indicare un IP qualunque si può utilizzare la notazione `%any`, anche se per compatibilità viene supportato con lo stesso significato anche l'indirizzo `0.0.0.0`.

Per determinare quale segreto (e relativa modalità di autenticazione) verrà usato fra quelli indicati in `/etc/ipsec.secrets` viene cercata una corrispondenza fra gli indirizzi locale e remoto dei due capi della connessione con gli indicatori della prima parte di ogni voce, e viene scelta la corrispondenza più specifica. L'assenza di indicatori implica che la voce corrisponderà ad una combinazione qualsiasi per gli indirizzi dei capi della VPN. Un solo indicatore verrà confrontato con l'indirizzo del capo locale (il ricevente) senza nessun controllo sul capo remoto (il richiedente), qualora ve ne siano indicati più di uno devono trovare corrispondenza entrambi i capi della connessione, a meno che non si stia usando una autenticazione a chiave asimmetrica (RSA) nel qual caso la lista viene interpretata come elenco delle possibili identità del capo locale e basta che l'indirizzo di questo corrisponda con uno degli indicatori dell'elenco.

Per capire meglio il funzionamento delle varie modalità di identificazione dei capi di una VPN tratteremo più esplicitamente i metodi più usati per la gestione delle autenticazioni: *Pre-Shared Key*, chiavi RSA e certificati. Si ricordi che indipendentemente dalla configurazione di VPN scelta (*road-warrior*, fra reti o fra singole macchine), è comunque necessario che ciascun *security*

esempio di come *non* fare le cose.

gateway possa identificare l'altro e questo comporta sia la presenza delle proprie credenziali (che nel caso di una *Pre-Shared Key* coincidono con quelle dell'altro capo) in `ipsec.secrets` per farsi autenticare, che l'indicazione di come identificare l'altro in `ipsec.conf`.

Benché meno sicuro rispetto agli altri metodi, che non prevedono la necessità di trasmissione su un canale sicuro della chiave condivisa e la compromissione di entrambi i lati della comunicazione in caso di intercettazione, l'uso delle *Pre-Shared Key* è ancora abbastanza comune per la sua semplicità di configurazione, specie quando ci si deve collegare con altri apparati per i quali l'uso degli altri metodi potrebbe risultare o più complesso o non supportato. Per autenticare un corrispondente con questo metodo occorre indicarne la scelta nella sezione che definisce della connessione in `ipsec.conf` con il parametro `authby=secret`.

In questo caso occorre anche la chiave sia indicata in `ipsec.secrets` in una voce di tipo `PSK` come stringa delimitata da delle virgolette che può contenere qualunque carattere eccetto le virgolette stesse o un a capo.[23] Un esempio di questo tipo di voci è il seguente:

ipsec.secrets

```
# sample /etc/ipsec.secrets file for 10.1.0.1
10.1.0.1 10.2.0.1: PSK "secret shared by two hosts"

# an entry may be split across lines,
# but indentation matters
www.xs4all.nl @www.kremvax.ru
    10.6.0.1 10.7.0.1 1.8.0.1: PSK "secret shared by 5"

# shared key for road warrior
10.0.0.1 0.0.0.0 : PSK "jxTR1lnmSjuj33n4W51uW3kTR55luUmSmnlRUuW..."
```

Si tenga presente che, come nell'esempio, si possono avere diverse *Pre-Shared Key*, specificate in altrettante voci del file, da usare per VPN fra macchine diverse. In questo caso come accennato la chiave da usare per una certa connessione viene determinata in base agli IP delle due macchine ai capi della VPN, cercando una corrispondenza nella lista degli indici associati a ciascuna chiave.

Questo ha una conseguenza precisa quando si usano le *Pre-Shared Key* per i *road-warrior*. In tal caso infatti, dato che l'indirizzo che essi assumeranno non può essere noto a priori, si dovrà usare nell'indice l'indirizzo generico, e a questo punto non si potranno usare due chiavi diverse per due macchine diverse, in quanto non sarebbe più possibile la risoluzione univoca della chiave da usare. Ciò comporta che per tutti i *road warrior* è necessario usare la stessa chiave simmetrica, cosa che rende questa modalità di autenticazione piuttosto debole.

La seconda modalità di autenticazione è quella basata su chiavi RSA, identificate dall'omonimo identificatore illustrato in tab. 4.4. In questo caso il possesso della chiave privata serve soltanto ad identificare noi stessi presso gli altri capi di tutte le eventuali VPN che si vogliono creare, cui bisognerà distribuire la nostra chiave pubblica. L'autenticazione degli altri capi della connessione, utilizzando un meccanismo a chiave asimmetrica, non sarà più fatta in base al contenuto di `ipsec.secrets` (non serve più infatti un segreto condiviso), ma sarà effettuata sulla base della *loro* chiave pubblica, cui dovrà essere associata, dalla loro parte, la relativa chiave privata mantenuta nel *loro* file `ipsec.secrets`. Questo consente di utilizzare chiavi diverse per ciascun *road warrior*.

[23] in questo caso la chiave serve sia ad identificare l'altro capo che a farsi identificare.

Le chiavi possono essere specificate in maniera diversa, a seconda della modalità utilizzata per definirle. Il parametro principale da usare in `ipsec.conf` in questo caso è `leftrsasigkey` che indica la chiave pubblica dell'altro capo, inoltre in questo caso non è necessario assegnare `authby=pubkey` perché questo è il valore di default. Con FreeS/WAN occorreva scrivere direttamente la chiave nel file in un opportuno formato testuale,[24] con le versioni più recenti si usa il valore speciale `%cert` e si indica il file da cui leggerla con `leftcert`, per cui è sufficiente disporre della chiave in un file di formato standard.

Allo stesso tempo perché l'altro capo della connessione possa identificarci sarà necessario specificare la nostra chiave privata in `ipsec.secrets`; questo viene fatto con l'indicativo RSA seguito dal nome del file in cui questa è contenuta.[25] Questo può essere un pathname assoluto o relativo rispetto a `/etc/ipsec.d/private`. Opzionalmente si può indicare come secondo argomento una password da usare per sbloccare la chiave (se questa è protetta) o l'uso della parola riservata `%prompt` per far effettuare la richiesta della stessa sulla console all'avvio della VPN. Un esempio di questa configurazione potrebbe essere il seguente:

ipsec.secrets

```
: RSA /etc/ipsec.d/private/hollandKey.pem
```

Si tenga presente che la configurazione andrà eseguita per entrambi i capi della VPN, usando le chiavi pubbliche di ciascuno di essi. Di nuovo non importa quale è la macchina identificata come `left` e qual'è quella identificata come `right`, l'importante è che si assegnino le chiavi pubbliche in maniera coerente; un errore comune che si aveva quando le chiavi pubbliche erano scritte direttamente dentro `ipsec.conf` era quello di assegnare per sbaglio la chiave pubblica dell'altro capo al proprio capo e viceversa, con la conseguente impossibilità di stabilire la connessione per la non corrispondenza alle rispettive chiavi private.

Con l'uso delle chiavi RSA si ottiene una maggiore robustezza nella gestione delle singole VPN, in quanto basterà generare una sola di queste chiavi per ciascun capo di una VPN, e potremo usare detta chiave per tutte le connessioni con tutti gli altri, distribuendo la relativa chiave pubblica. In questo modo non sarà necessario distribuire segreti condivisi. L'altro vantaggio è che in questo caso le configurazioni *road warrior* possono mantenere chiavi RSA diverse in quanto la scelta di quale chiave usare nella comunicazione sarà fatta in base alla chiave pubblica che sta sull'altro capo.

Infine se si usa un certificato X.509 per identificare l'altro capo si danno in sostanza due possibilità. La prima è quella di usare certificati autofirmati,[26] per i quali occorrerà semplicemente specificare il nome del file nel parametro `leftcert`, in forma assoluta o relativa se il certificato è posto nella directory `/etc/ipsec.d/certs`. Come per le chiavi RSA si dovrà aver cura di copiare il proprio certificato anche sull'altro capo e viceversa, e per entrambi assegnare correttamente sia `leftcert` che `rightcert`.

La comodità dell'uso dei certificati X.509 sta però nella seconda possibilità che prevede di eseguire l'autenticazione usando una *Certification Authority* che garantisce l'autenticità dei

[24] con *Openswan* questo è ancora possibile e si può ottenere il valore da inserire all'altro capo usando il comando `ipsec showhostkey` dalla propria parte.

[25] con *Openswan* è ancora supportata l'indicazione diretta della chiave privata dentro questo file con un opportuno formato testuale, non la tratteremo essendo una configurazione inutilmente complessa e facilmente soggetta ad errori.

[26] che ad esempio su Debian nel caso di *Openswan* vengono generati direttamente dal sistema di *debconf*.

certificati senza doverli trasferire a mano. Se si vuole usare questa funzionalità però occorre definire quale è il certificato della *Certification Authority* che firma i certificati usati dalle singole macchine. Questo può essere fatto con il parametro `leftca` in `ipsec.conf`, il cui valore deve essere assegnato al nome del rispettivo file. In generale però questo non è necessario, perché se detto parametro non viene definito, verranno usati automaticamente tutti i certificati che si trovano nella directory `/etc/ipsec.d/cacerts`, per cui tutto quel che serve è copiare in tale directory il certificato della nostra CA.

Questa seconda modalità ha il vantaggio che, come per SSL, basterà distribuire un solo certificato, quello della CA che ha firmato i certificati dei vari capi delle connessioni, invece di dover distribuire i singoli certificati pubblici di ciascuno di essi. In ogni caso si dovrà comunque impostare (con `leftcert` o `rightcert`) quale è il proprio certificato ed indicare all'interno di `ipsec.secrets` la corrispondente chiave privata; cosa che, trattandosi in genere anche in questo caso di chiavi RSA, si fa esattamente come illustrato in precedenza.

4.3 La gestione di VPN con OpenVPN

Tratteremo in questa sezione una modalità alternativa di realizzare delle VPN che non si basa su un protocollo che opera direttamente a livello di rete come IPSEC, e che per questo deve essere implementato nel kernel, ma viene realizzata a livello di applicazione da un apposito programma in user space. Pur esistendo diversi programmi in grado di fare questo[27] abbiamo scelto di concentrarci su *OpenVPN* per l'utilizzo di una tecnologia standard ed ampiamente consolidata come SSL/TLS, da cui deriva una notevole robustezza ed affidabilità dell'implementazione.

4.3.1 Introduzione

Abbiamo accennato in sez. 4.2 come l'uso di FreeS/Wan presentasse delle notevoli complessità sia sul piano dell'installazione che su quello amministrativo. FreeS/Wan inoltre implementava direttamente la gestione del routing dei pacchetti uscenti dal tunnel IPSEC, appesantendo la configurazione del servizio, ed introducendo una distinzione innaturale fra connessioni dirette fra i due *security gateway* (che devono essere impostate a parte) e connessioni fra le reti che stanno dietro di essi. Anche se con le implementazioni più recenti tutto questo si è semplificato notevolmente resta il fatto che si deve gestire esplicitamente la eventuale presenza di NAT ed una complessità di configurazione non trascurabile.

Per questi motivi l'utilizzo di un programma in user space può risultare notevolmente semplificato, sia per quanto riguarda l'attraversamento di un firewall, sia per i dettagli implementativi, che non necessitano più di una implementazione all'interno del kernel, ma possono appoggiarsi direttamente a tutte le librerie e le funzionalità che sono disponibili in user space.

Il principio di *OpenVPN* è quello di utilizzare l'infrastruttura delle interfacce `tun/tap` del kernel, una funzionalità che consente di creare delle interfacce *virtuali* sulle quali i pacchetti ricevuti, invece che da un dispositivo fisico, arrivano da un programma, al quale poi vengono mandati tutti i pacchetti inviati sulla stessa interfaccia.

[27] esempi sono `vtun` o `cipe`, che presentano però presentano mancanze notevoli sul lato crittografico, come rilevato da Peter Gutmann in `http://www.cs.auckland.ac.nz/~pgut001/pubs/linux_vpn.txt`.

In questo modo si può realizzare una VPN utilizzando due applicazioni che dialogano fra loro con la normale interfaccia dei socket, ed il cui solo compito è cifrare opportunamente tutto il traffico che vedono arrivare da questa interfaccia per inviarlo sul socket, e rimandare indietro alla stessa interfaccia, dopo averlo decifrato, il traffico che invece ricevono dal socket. Si otterrà così che i pacchetti inviati in ingresso su una interfaccia su un capo della VPN compariranno in uscita sulla corrispondente interfaccia sull'altro capo, secondo lo schema di fig. 4.5.

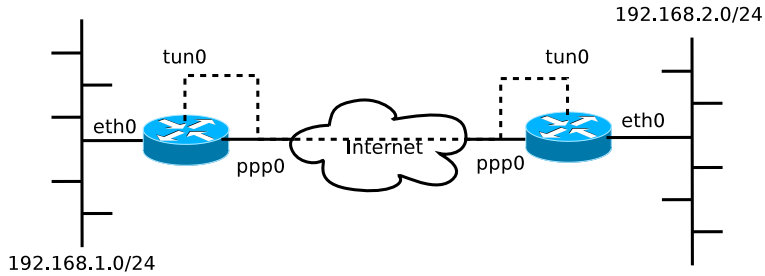


Figura 4.5: Schema di funzionamento di una VPN basata su OpenVPN.

Come si vede il concetto è tutto sommato molto semplice, l'unico problema di questo approccio è quello di una opportuna gestione del canale cifrato. OpenVPN ha risolto questo problema utilizzando dei socket UDP per l'invio dei dati attraverso internet e SSL/TLS come protocollo per la autenticazione degli estremi del tunnel e per lo scambio delle chiavi per la cifratura degli stessi.

La scelta di UDP è naturale in quanto i pacchetti inviati su una VPN implementano in sostanza una comunicazione a livello di rete facendo passare sul canale pacchetti IP generici, e UDP è il protocollo del livello di trasporto più vicino ad IP. L'uso di TCP per un tale lavoro infatti può portare a dei problemi di prestazione notevoli, in quanto le funzionalità che ne garantiscono le caratteristiche di affidabilità sono pensate per essere usate direttamente sopra IP, se queste vengono ripetute all'interno di un altro stream TCP si possono avere degli effetti di interferenza che possono portare ad una forte degradazione del traffico e anche alla caduta delle connessioni.[28]

La scelta di SSL/TLS è quello che rende superiore OpenVPN rispetto a molte altre implementazioni finora effettuate con lo stesso criterio. Programmi come `vtun` o `cipe` infatti hanno reimplementato un proprio modello di sicurezza che in analisi successive si è dimostrato essere molto debole; SSL/TLS invece è un protocollo di crittografia ed autenticazione la cui validità è stata verificata sul campo in anni di studio e di utilizzo, che pertanto rende OpenVPN pienamente affidabile sul piano della sicurezza.

Un'altra caratteristica notevole di OpenVPN è la sua disponibilità per una grande quantità di piattaforme diverse; in sostanza è stato portato su tutti gli Unix conosciuti e pure su Windows (a partire da Windows 2000), è pertanto molto semplice creare VPN anche con macchine che utilizzano sistemi operativi diversi; questo rende la soluzione estremamente flessibile e facilmente dispiegabile anche in ambienti eterogenei.

[28] una spiegazione molto chiara del perché è una cattiva idea fare una trasmissione di pacchetti TCP incapsulati su TCP si trova su `http://sites.inka.de/sites/bigred/devel/tcp-tcp.html`.

Data la maggiore semplicità dell'implementazione ed anche un sostanziale vantaggio sul piano della sicurezza non dovendo operare a livello del kernel, ci si può chiedere che senso abbia utilizzare ancora i derivati di FreeS/Wan ed un protocollo complesso come IPSEC. Un tempo uno degli svantaggi maggiori era la necessità di utilizzare una istanza diversa del programma per ciascun canale cifrato, ciascuna con un suo file di configurazione separato, cosa assai poco appetibile dal punto di vista dell'amministrazione (è in genere più gradito avere un singolo file di configurazione per ciascun servizio). Questo problema però non sussiste più con la versione 2.0 che consente di gestire con una sola istanza quanti canali si vogliono.

In realtà oggi la sola ragione a favore dei derivati di FreeS/Wan resta quella dell'interoperabilità. Benché OpenVPN sia ampiamente portabile e sia stato portato anche su molti router (ad esempio dal progetto OpenWRT), il vantaggio dell'uso di un protocollo standard come IPSEC consente di interagire con qualunque apparato che supporti lo standard, comprese implementazioni diverse dello stesso, come quelle presenti su router e firewall hardware su cui non si è in grado di installare OpenVPN. Qualora non si abbia questa necessità si può tranquillamente utilizzare OpenVPN risparmiandosi la complessità di gestione di IPSEC.

4.3.2 Installazione e configurazione di base

L'installazione di OpenVPN è estremamente semplice, anche perché viene fornito già pacchettizzato dalle principali distribuzioni;[29] inoltre il progetto fornisce pacchetti già pronti e l'installazione dai sorgenti è immediata segue la procedura standard dei classici `./configure; make; make install`.

Per quanto riguarda l'utilizzo OpenVPN può essere invocato direttamente a linea di comando, specificandone le relative opzioni, e, con l'eccezione delle opzioni utilizzate per la gestione delle chiavi, il programma verrà automaticamente eseguito in background, funzionando come un qualunque demone. Per ciascun canale cifrato si dovrà lanciare una istanza del programma, assegnandone opportunamente le porte da utilizzare.

Con la versione 2.x è diventato possibile utilizzare il programma in modalità server (vedi sez. 4.3.3), in cui più client si collegano ad una sola istanza dello stesso. Questo comporta una procedura di configurazione leggermente diversa rispetto alla configurazione classica ereditata della versione 1.x, che comunque resta utilizzabile. Inoltre nel Novembre 2004 OpenVPN 2.0 ha ottenuto dalla IANA l'assegnazione ufficiale della porta 1194 per il funzionamento in modalità server.

Tutte le opzioni di configurazione si possono specificare sia direttamente a linea di comando, come opzioni estese (cioè precedute da un `--`), che all'interno di un file di configurazione, in tal caso si potrà usare come direttiva il nome dell'opzione, omettendo il `--` iniziale. L'opzione che permette di specificare un file di configurazione è `--config` che richiede si fornisca come parametro il nome dello stesso, questa può essere ripetuta anche all'interno del file di configurazione stesso, consentendo una nidificazione (che comunque è limitata ad un numero ragionevole di livelli).

Dato che alla fine l'uso di un file di configurazione rende la gestione molto più semplice rispetto ad dover specificare un gran numero di opzioni a riga di comando, da qui in avanti faremo riferimento alle opzioni di configurazione nella loro forma di direttive.

[29] ad esempio Debian fornisce direttamente il pacchetto `openvpn`.

In genere con il pacchetto `openvpn` viene installato anche un opportuno script di avvio che permette di avviare automaticamente più canali, creati con altrettanti file di configurazione. Lo script legge il contenuto della directory `/etc/openvpn` e lancia una istanza del programma per ogni file con estensione `.conf` che trova in tale directory, passandogli lo stesso come file di configurazione.

L'installazione di default non configura nessun canale e lascia la directory `/etc/openvpn` vuota, questo significa che di default il servizio non verrà attivato. Per poter avviare il servizio occorrerà creare almeno un file di configurazione. Con il progetto vengono distribuiti una serie di file di esempio,[30] relativi a diverse configurazioni della rete, che possono essere utilizzati come scheletri, adattandoli alle proprie esigenze.

Le direttive del comando si possono dividere sommariamente in varie classi, la prima delle quali è quella delle direttive relative alla gestione del tunnel, che permettono di controllare le proprietà di quest'ultimo e le modalità della sua creazione. Alcune di queste sono fondamentali e devono essere sempre specificate per poter utilizzare il programma.

La direttiva `dev` indica quale tipo di interfaccia virtuale usare per creare il canale, ed i valori possibili sono `tap`, per l'incapsulamento su Ethernet e `tun` per l'incapsulamento su IPv4. Il tipo deve essere lo stesso su entrambi i capi della VPN, e se lo si indica senza specificare un numero di interfaccia viene utilizzata la prima disponibile, altrimenti l'assegnazione può essere fatta in maniera statica; ad esempio con "`dev tun0`" si richiede esplicitamente l'uso dell'interfaccia `tun0`.

La direttiva `ifconfig` permette di configurare le interfacce ai due capi del tunnel; i parametri da specificare cambiano a seconda che si sia usato una interfaccia di tipo `tun` o di tipo `tap`. Nel primo caso si devono specificare gli indirizzi IP ai due capi del tunnel, come si farebbe con le interfacce `ppp0` per i due capi di un collegamento punto-punto via modem.

Nel secondo caso invece si va ad operare direttamente a livello di protocollo Ethernet, e si deve quindi specificare un indirizzo IP e la relativa netmask. Questi verranno associati all'interfaccia `tap` esattamente come se si trattasse una interfaccia Ethernet ordinaria che si affaccia su una rete locale connessa tramite la VPN. Si tenga presente che questo deve essere fatto solo per un client che si collega dall'esterno in VPN, che a questo punto potrà inviare pacchetti ethernet sulla interfaccia `tap` come se questa fosse una scheda connessa, tramite OpenVPN, allo switch della LAN.

Perché questo funzioni dal lato del server che si affaccia su una LAN si dovrà in questo caso adottare una configurazione diversa, in quanto per poter reinviare i pacchetti che arrivano dall'interfaccia `tap` sulla rete locale questa dovrà essere unita alla interfaccia ethernet che su di essa si affaccia con un *bridge*, scruttando il relativo supporto e creando quest'ultimo con il comando `brctl`. Questo permette ad esempio di mantenere entrambi i lati della VPN sulla stessa rete, e di far passare sul canale cifrato quest'ultimo anche i protocolli di livello più basso (ad esempio si potrà usare un DHCP remoto attraverso la VPN).

Una terza direttiva fondamentale, che deve essere specificata per almeno uno dei due capi del tunnel, è `remote`, che serve a indicare l'indirizzo IP a cui è possibile contattare (passando attraverso internet) l'altro capo. Non specificarla significa che `openvpn` accetterà connessioni da qualunque indirizzo, previa autenticazione dell'altro capo con uno dei vari metodi supportati, permettendo così configurazioni *road warrior* in cui un capo della VPN (quello che non la usa) è quello fisso, e l'altro può porsi in contatto con lui attraverso appunto questa direttiva.

[30] su Debian sono installati in `/usr/share/doc/openvpn/example/sample-config-files/`.

Direttiva	Significato
`local`	indica l'indirizzo IP locale su cui far ascoltare il tunnel, se non specificato ascolta su tutti gli indirizzi disponibili su tutte le interfacce di rete.
`remote`	indica l'indirizzo IP dell'altro capo del tunnel.
`proto`	specifica il protocollo da utilizzare per la creazione del canale; il default è `udp`, ma se si vuole usare TCP allora occorrerà usare `tcp-client` sul capo del tunnel che inizia la connessione e `tcp-server` sull'altro capo che resterà in ascolto (è comunque sconsigliabile usare TCP).
`shaper`	impone una limitazione, specificata in byte al secondo, sul traffico che può passare attraverso il canale.
`ifconfig`	specifica due IP assegnati ai due capi del tunnel, il primo è l'IP locale, il secondo quello remoto.
`port`	specifica la porta da utilizzare per la comunicazione su internet, qualora si vogliano specificare due porte diverse per il capo locale e quello remoto si possono usare le opzioni `lport` e `rport`.
`nobind`	non effettua il binding ad una specifica porta usando la porta effimera data dal kernel.
`dev`	specifica l'interfaccia di rete da usare per creare il canale (assume i valori `tun` o `tap`).
`user`	imposta l'utente per conto del quale deve essere eseguito il demone una volta completata l'inizializzazione, in modo da rilasciare i privilegi non necessari.
`group`	analoga ad `user` per impostare il gruppo.
`log`	imposta un file sul quale eseguire il log dei messaggi del demone.
`verb`	imposta la prolissità dei messaggi di log.

Tabella 4.5: Le principali direttive di `openvpn` relative alla gestione del tunnel.

Altre direttive utili sono `log` che specifica un file su cui salvare i messaggi del demone e `verb` che indica un livello di *prolissità* degli stessi; in tab. 4.5 si sono comunque riportate le direttive per la gestione del tunnel maggiormente utilizzate; l'elenco completo, insieme ai dettagli dei parametri possibili e del loro significato si trova al solito nella pagina di manuale del comando, accessibile con `man openvpn`.

Una seconda classe di direttive è quella che riguarda le modalità per gestire l'autenticazione dei due capi della VPN e la trasmissione dei dati cifrati sul canale. OpenVPN supporta due modalità di funzionamento, la prima è quella più elementare e prevede dell'uso di una chiave statica condivisa fra i due capi della connessione, che viene usata sia per autenticare gli stessi che per cifrare i dati.

In questo caso si ha il vantaggio di una configurazione immediata, basta infatti indicare nella configurazione di ciascun estremo quale è il file contenente la chiave condivisa con l'uso della direttiva `secret`. Per gli ovvi motivi affrontati in sez. 1.2.2 ogni canale avrà bisogno di una chiave diversa. Data la criticità della chiave questa deve essere generata direttamente con il programma stesso invocandolo come:

```
monk:~# openvpn --genkey --secret chiave.key
```

che genererà un file `chiave.key` con un contenuto del tipo:

chiave.key

```
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
f6703eb17814284ef14df4fb1cf79f42
1b5718e1c86a0c16d06b8a5d4901a88d
4ed0f5fb9393cf858653daa4ed6ab65d
ec6ede77ec657ca11be448b7572ccb0b
...
...
6126425a3f4fff2f1f9743c7fd44d647
ce5058749cc4a01caaa9dd7de82fd8e7
-----END OpenVPN Static key V1-----
```

La configurazione in questo caso è immediata, in quanto basta avere una modalità sicura di copiare il file su entrambi gli estremi della VPN per avere un canale cifrato funzionante, questo però indebolisce la sicurezza complessiva del sistema, in quanto si deve comunque eseguire la copia di un file che è noto su entrambe le macchine, ed utilizzando una chiave di cifratura che non viene mai cambiata (se non manualmente) ci espone, in caso di furto della stessa, alla possibilità che tutto il traffico (compreso quello precedente al furto) possa essere decifrato.

Per questo si usa in genere questo metodo in fase di test, e poi si passa al secondo metodo di autenticazione è quello basato sull'uso di SSL/TLS. In tal caso tutta la problematica di negoziazione e scambio della chiave per la cifratura del canale viene gestita da OpenVPN attraverso una sessione TLS, che permette sia di autenticare i due capi della connessione che di creare un canale di controllo su cui scambiare le chiavi di sessione usate per cifrare il canale dei dati.

Per poter utilizzare questa funzionalità ogni capo della comunicazione deve poter disporre di un suo certificato e della relativa chiave, firmati dalla stessa *certification authority*. Ciascun capo della connessione verificherà che il certificato presentato dall'altro capo sia debitamente firmato, e se il controllo riesce a questo punto l'autenticazione sarà considerata ottenuta e si procederà con lo scambio delle chiavi di sessione.

Il primo passo per utilizzare la modalità SSL/TLS è quello di designare quale dei due capi assumerà il ruolo di server e quale quello di client (nella modalità tradizionale questi ruoli servono solo allo scopo della creazione del canale di controllo su TLS, la trasmissione dei dati è sempre punto-punto). Questo deve essere utilizzando rispettivamente le direttive `tls-server` e `tls-client` sui due estremi della VPN.

Il passo successivo è ottenere tutti i certificati necessari per i due capi, le relative chiavi ed il certificato della *Certification Authority* usata per stabilirne la validità (per le problematiche relative si consulti sez. 2.1.2). Una volta che questi siano disponibili basterà indicarli nella configurazione utilizzando le direttive `cert`, `key` e `ca`, ciascuna delle quali vuole come parametro il nome del rispettivo file in formato PEM.

Un'altra direttiva necessaria, ma solo sul lato server dell'autenticazione TLS, è `dh`, che specifica i parametri di *Diffie-Hellman*[31], un riassunto delle altre principali opzioni di configurazione di questa classe è stato riportato in tab. 4.6, al solito l'elenco completo è nella pagine di manuale.

[31] sono i parametri necessari ad ottenere la chiave di sessione con la procedura di *Diffie-Hellman* attivando il meccanismo della *Perfect Forward Security*, che assicura che anche se un attaccante venisse in possesso della chiave di uno dei certificati, non sarebbe in grado di decifrare il traffico precedente.

Direttiva	Significato
`secret`	specifica il file che contiene la chiave statica condivisa utilizzata per la cifratura del canale.
`cert`	specifica il file contenente il certificato locale firmato (in formato PEM).
`key`	specifica il file contenente la chiave del certificato locale (in formato PEM).
`ca`	specifica il file contenente il certificato della *Certification Authority* che firma i certificati locali (in formato PEM).
`dh`	specifica il file contenente i parametri necessari per la procedura di *Diffie-Hellman* sul lato `tls-server` (in formato PEM).
`pkcs12`	specifica il file che contiene il certificato locale, la relativa chiave ed il certificato della CA che li firma in formato PKCS12.
`tls-server`	abilita l'uso di SSL/TLS e facendo assumere al capo corrente il ruolo di server nella negoziazione.
`tls-client`	abilita l'uso di SSL/TLS e facendo assumere al capo corrente il ruolo di client nella negoziazione.
`tls-auth`	abilita una protezione da attacchi DoS richiedendo una autenticazione (a chiave condivisa) prima di iniziare la negoziazione SSL/TLS.
`askpass`	richiede la password della chiave del certificato locale prima di eseguire il demone (solo da riga di comando).
`reneg-sec`	specifica ogni quanti secondi rinegoziare le chiavi di sessione (analoghi sono `reneg-pkts` e `reneg-bytes`).
`comp-lzo`	abilita un algoritmo di compressione dei dati sul canale.
`status`	scrive lo stato delle operazioni sul file passato come argomento; prende un secondo argomento opzionale per indicare la frequenza di scrittura in secondi.

Tabella 4.6: Le principali opzioni di `openvpn` relative alla gestione dell'autenticazione e della cifratura.

Una terza classe di opzioni riguarda la capacità di eseguire delle operazioni ausiliarie da compiere nell'atto di attivare la VPN, come la possibilità di lanciare una serie di script ausiliari. In particolare sono definiti 5 momenti diversi, lungo la creazione di un canale, in cui il programma può eseguire degli script esterni in grado di compiere operazioni di appoggio. Questi sono riportati nell'ordine nelle prime cinque righe di tab. 4.7.

La configurazione più comune usando OpenVPN nella modalità tradizionale è, una volta stabiliti gli indirizzi dei due capi del tunnel con `ifconfig`, quella di utilizzare `up` per lanciare uno script non appena il canale si è attivato, che stabilisca le opportune rotte statiche all'avvio della VPN, in modo da usare i due capi del tunnel come gateway per le rispettive reti private. Questa, come le altre direttive di esecuzione, prendono come parametro il nome di un comando esterno che di norma è uno opportuno script di shell che verrà eseguito nel momento indicato in tab. 4.7.

Al momento dell'esecuzione al comando esterno saranno passati una serie di argomenti che specificano le proprietà del tunnel; questi dipendono dal tipo di dispositivo che si è indicato con `dev`, nel caso di `tun` saranno passati nell'ordine: il nome dell'interfaccia del tunnel, la MTU della stessa, la MTU del collegamento, l'indirizzo IP locale assegnato al tunnel e l'indirizzo IP remoto; nel caso invece si sia usata una interfaccia `tap` saranno passati il nome dell'interfaccia, la MTU

Direttiva	Significato
`up`	esegue il comando di shell passato come parametro una volta aperte con successo le interfacce del tunnel.
`tls-verify`	esegue il comando shell passato come parametro prima della verifica dell'autenticità dell'altro capo della connessione (eseguita con SSL/TLS).
`ipchange`	esegue il comando shell passato come parametro dopo l'autenticazione della connessione o quando un indirizzo remoto è cambiato.
`route-up`	esegue il comando shell passato come parametro dopo che sono state aggiunte le rotte alla tabella di routing.
`down`	esegue il comando di shell passato come parametro dopo chiusura delle interfacce del tunnel.
`route-delay`	attende il numero di secondi passato come parametro prima di inserire le rotte nella tabella di routing (un valore nullo, il default, indica l'inserimento immediato).
`route-gateway`	usa l'indirizzo passato come parametro come valore di default per il gateway di una rotta impostata con `route`.
`route`	aggiunge una voce alla tabella di routing una volta che è stata realizzata una connessione.
`mktun`	permette di creare un tunnel persistente (che viene mantenuto anche se OpenVPN non è attivo).
`rmtun`	rimuove un tunnel permanente.

Tabella 4.7: Altre opzioni di controllo di `openvpn`.

della stessa, la MTU del collegamento, l'indirizzo IP e la netmask del tratto di rete utilizzato. In entrambi i casi si potranno usare gli argomenti per eseguire (all'interno dello script) gli opportuni comandi per impostare l'instradamento dei pacchetti attraverso il tunnel.

Oltre all'uso di comandi esterni OpenVPN supporta anche una serie di direttive che fanno eseguire alcune operazioni direttamente al programma stesso. Fra queste una delle più importanti è `route` che permette di far impostare direttamente ad `openvpn` ulteriori rotte statiche una volta che la connessione viene stabilita, senza dover ricorrere all'uso di un comando esterno lanciato con `up`.

La direttiva prende come primo parametro obbligatorio un indirizzo IP, che può essere di una singola macchina o di una rete, nel qual caso occorrerà aggiungere come secondo parametro la maschera di rete ed eventualmente un gateway per la stessa. Se non si specifica un gateway il default è di usare l'indirizzo associato al tunnel (quello specificato dal secondo parametro di `ifconfig`) oppure quello impostato con la direttiva `route-gateway`. In questo modo diventa possibile, all'attivazione di un tunnel, inserire automaticamente la rotta per gli indirizzi della rete che si trova dietro l'altro capo dello stesso.

Collegata a questa direttiva è anche `route-delay` che stabilisce il numero di secondi (passato come parametro) da aspettare prima di aggiungere una rotta alla tabella di routing. Normalmente queste vengono inserite immediatamente dopo l'apertura del dispositivo di rete (TAP o TUN che sia) e l'esecuzione dello script indicato da `up`, ma prima che vengano ceduti i privilegi di amministratore secondo quanto specificato dalle direttive `user` e `group`. In certi casi però[32] è necessario attendere un certo lasso di tempo, che può essere indicato da questa direttiva.

[32] ad esempio quando si usa il DHCP per ottenere un indirizzo sul dispositivo TAP, o su Windows per l'inizializzazione dell'adattatore TAP-Win32.

Un esempio tipico di file di configurazione tradizionale di un tunnel per una VPN basata su chiavi condivise è il seguente, che fa riferimento al capo *statico* della connessione, cioè quello che è posto su un indirizzo fisso, anche se in realtà non è necessario che l'indirizzo IP sia statico, è sufficiente che esso sia raggiungibile in maniera certa dalla macchina sull'altro capo tramite la direttiva `remote` (cosa che si può ottenere anche usando un servizio di DNS dinamico):

openvpn.conf

```
# Use a dynamic tun device.
dev tun
# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.2 10.1.0.1
# Our up script will establish routes
# once the VPN is alive.
up ./simone.up
# Our pre-shared static key
secret simone.key
# OpenVPN 2.0 uses UDP port 1194 by default
; port 1194
# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
verb 3
```

si noti come con `dev` si sia scelto l'interfaccia di tipo TUN per usare l'incapsulamento su IP, poi si siano impostati gli indirizzi del tunnel con `ifconfig` ed usato `up` per invocare uno script di inizializzazione che si incarichi di impostare la rotta statica per raggiungere la rete privata dietro la nostra macchina. Con `secret` si è indicato il file su cui è salvata la chiave di accesso, e con `port` la porta da utilizzare (ogni tunnel dovrà usarne una diversa). Infine si è abilitato un adeguato livello di logging.

In corrispondenza alla precedente configurazione, quello che segue è l'estratto del file utilizzato sulla macchina all'altro capo del tunnel, che le consente di collegarsi (in questo caso da qualunque indirizzo) alla VPN:

openvpn.conf

```
# Use a dynamic tun device.
dev tun
# Our OpenVPN peer is the office gateway.
remote holland.truelite.it
# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.21 10.1.0.22
# Our up script will establish routes
# once the VPN is alive.
up ./simone.up
# OpenVPN 2.0 uses UDP port 1194 by default
; port 1194
# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
```

```
# 9 -- verbose, good for troubleshooting
verb 3
```

e come si può notare la sola differenza è che gli indirizzi di `ifconfig` sono invertiti di ruolo, e che in questo secondo caso, essendo su una macchina senza indirizzo prestabilito, si è specificato con `remote` l'indirizzo dell'altro estremo della connessione. Si noti come queste configurazioni non facciano uso della direttiva `route`, affidandosi a degli script esterni.

Quelle appena illustrate sono le configurazioni per la creazione di un tunnel, ma occorrerà prevedere anche una opportuna configurazione del firewall che consenta di far passare i pacchetti relativi al traffico eseguito sul canale cifrato. Per questo, assunto che OpenVPN sia attivo sul firewall stesso, oltre al traffico diretto del tunnel, si dovrà anche consentire il traffico dalla rete interna verso le interfacce del tunnel e viceversa. Un insieme di regole necessarie al funzionamento di OpenVPN in modalità punto-punto è il seguente:

firewall.sh

```
iptables -A INPUT -p udp --dport 1194 -m state --state NEW -j ACCEPT
iptables -A INPUT -i tun+ -m state --state NEW  -j ACCEPT
iptables -A FORWARD -i tun+ -m state --state NEW -j ACCEPT
```

dove è dato per scontato che si accettino i pacchetti in stato `ESTABLISHED` e `RELATED` e si è assunta una sola istanza che lavora sulla porta standard. La prima regola permette di accettare le connessioni per il funzionamento del tunnel e le altre due consentono le connessioni verso la macchina locale e verso la rete locale attraverso il tunnel. Ovviamente in presenza di più istanze si dovranno aprire le ulteriori porte da esse utilizzate.

Come accennato in precedenza quando si opera con OpenVPN in modalità punto-punto ci sono una serie di operazioni che devono essere compiute al di fuori dal programma tramite gli opportuni script. Tratteremo queste operazioni nel caso particolare di una VPN realizzata con l'interfaccia `tun` illustrato dai due estratti di file di configurazione; l'uso di `tap` infatti è normalmente più complesso e meno performante dovendo costruire un livello di incapsulamento in più.

Una volta che si è creato un tunnel IP tutto quello che si ha è una comunicazione cifrata fra i due estremi della VPN che si scambiano pacchetti attraverso una interfaccia come `tun0`; in modalità punto-punto OpenVPN si limita a questo, per ottenere una vera VPN (cioè una canale di comunicazione fra due tratti di reti private) sono necessari alcuni passi ulteriori, che come accennato sono realizzati tramite gli opportuni script specificati come parametro per la direttiva `up`.

Una volta che il canale è attivo quello che serve è aggiungere nella tabella di routing di ciascun estremo della VPN una rotta statica che consenta, dalle macchine nella rete privata dietro lo stesso, di raggiungere quelle della rete dietro l'altro estremo, facendo riferimento alla configurazione di esempio di fig. 4.5, sull'estremo di destra dovremo avere uno script del tipo di:

up.sh

```
#!/bin/bash
route add -net 192.168.1.0 netmask 255.255.255.0 gw $5
```

Si noti come si sia usato come gateway per il raggiungimento dell'altra rete il quinto argomento passato allo script; questo è l'indirizzo IP associato all'interfaccia `tun` dell'altro capo del

tunnel. In questo modo, fintanto che la nostra macchina riceve i pacchetti diretti verso la rete al di là della VPN, questi verranno instradati verso l'interfaccia del tunnel, e attraverso di questa saranno cifrati e poi trasmessi su internet usando il socket di collegamento di `openvpn`. Ovviamente perché il meccanismo funzioni occorre che la stessa operazione venga ripetuta sull'altro capo della VPN, altrimenti i pacchetti non avranno una strada per tornare indietro.

4.3.3 La configurazione in modalità server

Le opzioni di configurazione trattate finora sono le stesse sia per la versione 1.0 che per la versione 2.0 di OpenVPN, quest'ultima però, come accennato, ha introdotto una nuova modalità di funzionamento che consente di utilizzare una sola istanza del programma per fare da *server* nei confronti di un numero arbitrario di client, e che insieme a questo permette una gestione molto più sofisticata (e comoda) del canale cifrato.

La direttiva che stabilisce quale modalità di operazione utilizzare è `mode`, che prende come argomenti il valore `p2p` per indicare il meccanismo classico, con una istanza del programma per canale, o il valore `server` per indicare la nuova modalità di funzionamento con una unica istanza che fa da server, introdotta con la versione 2.0. Se non la si specifica viene assunto come default il comportamento classico, il che consente di riutilizzare i file di configurazione della versione 1.0 senza modifiche.

Il principale vantaggio della modalità server è che è sufficiente usare una sola porta (la 1194 UDP) ed un unico file di configurazione per gestire l'accesso alla propria rete privata da parte di un numero arbitrario di client. Essa inoltre supporta un meccanismo che consente al server di inviare ai client i dati necessari affinché questi possano configurare la rete in maniera corretta, senza dover ricorrere a degli script ad hoc come fatto in precedenza.

La scelta della modalità server però porta in maniera sostanzialmente obbligata anche all'uso di SSL per la creazione del tunnel, dato che altrimenti si sarebbe costretti all'uso di una singola chiave segreta identica per tutti i client, con gli ovvi problemi di sicurezza che questo comporta, configurazione che pertanto non prenderemo neanche in considerazione.

La direttiva principale per l'uso della modalità server è appunto `server`, che richiede due parametri, indirizzo e netmask della rete all'interno della quale saranno scelti gli indirizzi assegnati nella creazione dei singoli tunnel. In realtà, come illustrato nella pagina di manuale, questa non è altro che una direttiva riassuntiva fornita allo scopo di semplificare la configurazione sul server, che invoca automaticamente una serie di altre direttive.

In questa modalità infatti è compito dell'istanza che fa da server definire quali sono gli IP da assegnare ai capi di ciascun tunnel, che vengono scelti all'interno di un *pool* (la direttiva sottostante è in realtà `ifconfig-pool`) analogamente a quanto avviene per il DHCP. Specificando una rete con la direttiva `server` quello che accade è che al server verrà comunque associato il primo indirizzo della rete, e sarà creata una rotta statica per la suddetta rete in modo che tutti gli indirizzi dei tunnel siano raggiungibili. Per compatibilità con Windows l'allocazione degli indirizzi viene comunque effettuata all'interno di reti `/30`, per cui su una rete di classe C si avranno a disposizione un massimo di 64 tunnel.[33]

[33] in realtà a partire da OpenVPN 2.1 esiste la possibilità di assegnare la rete degli indirizzi interni dei client con diverse metodologie tramite la direttiva `topology`, ma il funzionamento richiede versioni aggiornate di tutti i software.

Come accennato il vantaggio della modalità server è che buona parte delle configurazioni dei client possono essere amministrate direttamente dal server. Questo avviene grazie alla direttiva `push` che consente di inviare ai client che si collegano al server una serie di direttive di configurazione. L'insieme delle configurazioni inviabili ad un client è limitato per motivi di sicurezza e fattibilità, le principali sono: `route`, `route-gateway`, `route-delay`; per un elenco più dettagliato si consulti la pagina di manuale. La direttiva `push` richiede un singolo parametro, per cui si deve aver cura di proteggere la direttiva che si intende inviare scrivendone il relativo testo fra virgolette.

Quando un client intende connettersi in modalità server a OpenVPN dovrà a sua volta utilizzare la direttiva `pull`, che abilita la ricezione delle direttive di configurazione dal server; queste ultime saranno poi applicate come se fossero presenti nel file di configurazione. In generale si usa al suo posto la direttiva generica `client` che oltre ad abilitare `pull` qualifica l'istanza di OpenVPN come client anche nella negoziazione della connessione SSL.

L'uso più comune di `push` è per inviare ai vari client delle direttive `route` che permettono di configurarne automaticamente la tabella di routing, inserendovi le rotte statiche relative alle varie reti private che sono raggiungibili attraverso il server. In questo modo se si aggiunge una nuova rete dietro al server non è più necessario modificare le configurazioni di tutti i client per aggiornare gli script di `up`.

Un secondo meccanismo molto utile fornito dalla modalità server è quello che consente di modificare dinamicamente la configurazione del server stesso in corrispondenza al collegamento di un client. Questo è governato dalla direttiva `client-config-dir`, che permette di indicare una directory (relativa a `/etc/openvpn`) in cui sono mantenute le opzioni di configurazione relative a ciascun client.

Per poter utilizzare questa funzionalità però è obbligatorio usare la gestione del canale con SSL, infatti per identificare un client il server utilizza il *Common Name* scritto nel certificato con cui esso si presenta in fase di connessione. Se all'interno della directory specificata da `client-config-dir` viene trovato un file con lo stesso nome[34] presente nel certificato una volta completata la connessione verranno eseguite le direttive di configurazione in esso contenute; anche in questo caso è disponibile solo un sottoinsieme limitato di direttive, le principali delle quali sono `iroute` e `push` (per l'elenco completo si faccia riferimento alla pagina di manuale). Se non viene trovato nessun file corrispondente ma è presente il file `DEFAULT`, verranno usate le direttive presenti in quest'ultimo.

L'uso più comune di questa direttiva è quello che consente di rendere visibili fra loro la rete dietro il server e una eventuale sottorete presente dietro al client. In questo caso però non basterà aggiungere (ovviamente usando la direttiva `route`) la rete posta dietro il client alla tabella di routing del server in modo che i relativi pacchetti siano inviati sull'interfaccia di tunnel;[35] si dovrà anche dire ad `openvpn` a quale fra i vari client eventualmente connessi esso deve instradare (internamente) detti pacchetti.[36] Questo viene fatto dalla direttiva `iroute` che prende come

[34] si faccia attenzione che alcuni nomi non vengono riconosciuti correttamente, ad esempio se si usa un indirizzo di posta (tipo `piccardi@truelite.it`) si possono avere dei problemi.

[35] si suppone che il server ed il client facciano da default gateway per le loro reti, in caso contrario si dovranno aggiungere le rotte statiche sulle singole macchine.

[36] questo problema non si pone con una configurazione punto-punto, in quanto ciascuna istanza usa una interfaccia diversa; però in tal caso occorre gestire l'inserimento di tutte le rotte su entrambi i capi della comunicazione con degli opportuni script da invocare esternamente.

argomenti l'indirizzo IP della rete e la relativa netmask;[37] ovviamente questa direttiva dovrà essere inserita nel file di configurazione specifico del client dietro il quale detta rete è posta.

La presenza della direttiva `iroute` è dovuta al fatto che quando opera in modalità server OpenVPN può ricevere il traffico proveniente da diversi client su una unica interfaccia, pertanto viene ad assumere, nei confronti di detto traffico, il ruolo di un router. Il comportamento di default del programma è quello di non instradare detti pacchetti, per cui ciascun client sarà in grado di "vedere" su tale interfaccia soltanto il server; si può però far agire OpenVPN come un vero router utilizzando la direttiva `client-to-client`. In tal caso infatti OpenVPN si incaricherà di instradare anche il traffico diretto ad altri client ad esso collegati, e questi potranno anche comunicare fra di loro attraverso il server.

Direttiva	**Significato**
`mode`	Specifica la modalità di funzionamento di OpenVPN, prende come parametro `p2p` o `server`.
`server`	Configura OpenVPN per operare in modalità server con tunnel di tipo TUN; prende due parametri (numero IP e netmask) che indicano la rete su cui vengono allocati i numeri IP usati per gli estremi dei vari tunnel.
`topology`	Specifica la modalità di assegnazione degli indirizzi dei client nella rete indicata da `server`; i valori possibili sono `net30` (il default) per l'uso di una sottorete `/30` per ciascun client, `p2p` per un singolo indirizzo per client, non supportata da client Windows, `subnet` sempre per un singolo indirizzo per client, funzionante anche con versioni recenti di Windows.
`push`	Invia ad un client che si connette la direttiva di configurazione passata come parametro.
`pull`	Indica ad un client di accettare le direttive di configurazione inviategli da un server con la direttiva `push`.
`ifconfig-pool`	Definisce un intervallo di indirizzi IP, i cui estremi sono indicati dai due parametri passati alla direttiva, all'interno dei quali sono scelti dinamicamente gli indirizzi da assegnare i capi di ciascun tunnel.
`ifconfig-pool-persist`	Definisce un file, passato come parametro, dove sono registrate le corrispondenze fra client ed IP assegnati.
`client-to-client`	Abilita l'instradamento del traffico fra diversi client così che questi possano vedersi fra loro; non richiede nessun parametro.
`client-config-dir`	specifica la directory contenente le configurazioni specifiche dei singoli client.
`iroute`	Imposta una rotta interna su uno specifico client, viene usata per inviare i pacchetti destinati ad una certa sottorete al tunnel relativo ad uno specifico client; si usa all'interno delle configurazioni specifiche dei client.
`client`	Indica una configurazione di tipo client, è equivalente all'uso delle due direttive `pull` e `tls-client`.
`duplicate-cn`	Consente il collegamento anche in presenza di più client che hanno un certificato con lo stesso common name.
`max-clients`	Consente il collegamento ad un numero massimo di client passato come parametro.

Tabella 4.8: Le opzioni di controllo di `openvpn` per la modalità server.

[37] qualora si intenda inserire una singola macchina invece di una rete la netmask può essere tralasciata.

Con l'uso di questa direttiva diventa anche possibile unire più reti private, presenti dietro vari client, in modo che esse possano comunicare fra loro. Questo può sempre essere fatto in modalità punto-punto, ma il costo è quello di inviare i pacchetti da una istanza all'altra di OpenVPN, e di dover predisporre per ciascuna istanza gli opportuni script per l'inserimento di tutte le relative rotte statiche. Lo svantaggio (a parte la minore efficienza) è che la riconfigurazione va fatta per tutti i client, per cui l'aggiunta di una nuova rete dietro un nuovo client comporta la modifica delle configurazioni di tutti gli altri.

La caratteristica interessante della modalità server è che invece, una volta attivata la direttiva `client-to-client`, detta configurazione può essere realizzata operando solo sul server. In tal caso infatti oltre ai passi precedentemente illustrati per "pubblicare" la rete presente dietro il nuovo client, basterà usare la direttiva `push` per inviare a tutti i client la rotta della nuova rete.[38]

In tab. 4.8 si sono riportate le principali opzioni relative alla configurazione di OpenVPN in modalità server (sia per il server che per il client). Al solito l'elenco completo delle varie opzioni/direttive è disponibile sulla pagina di manuale del comando `openvpn`.

Vediamo allora un esempio tipico di configurazione per OpenVPN in modalità server. Partiamo dal file di configurazione sul lato server, per una macchina che fa da gateway per la rete `192.168.1.0` (ufficio) cui si collega un client che fa da gateway per la rete `192.168.0.0` (casa), una volta eliminati commenti e righe vuote si avrà:

server.conf

```
port 1194
proto udp
dev tun
ca /etc/ssl/certs/Truelite-cacert.pem
cert vpncert.pem
key vpnkey.pem   # This file should be kept secret
dh dh4096.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "route 192.168.1.0 255.255.255.0"
client-config-dir ccd
route 192.168.0.0 255.255.255.0
keepalive 10 120
comp-lzo
user nobody
group nogroup
persist-key
persist-tun
status openvpn-status.log
verb 3
```

In questo caso si è usata la porta standard, per il resto le prime 6 direttive sono analoghe a quelle che sarebbero presenti in modalità punto-punto. Rispetto agli esempi illustrati in sez. 4.3.2 questa volta si è predisposto un tunnel creato con SSL per cui si sono usate le direttive `ca`, `cert`, `dh` e `key` al posto di `secret` per dichiarare i file contenenti i dati necessari.

La prima differenza con una configurazione punto-punto è la presenza della direttiva `server` che dichiara la rete su cui saranno allocati gli indirizzi IP dei tunnel. L'impostazione della rete

[38] fra questi ovviamente non ci sarà il client dietro cui tale rete si trova; questo viene curato automaticamente da OpenVPN, che se una rete è citata in una direttiva `iroute` nella configurazione di un client, modifica il comportamento di `push` in modo che essa non gli invii una rotta statica relativa a detta rete.

presente dietro il server è invece effettuata dalla direttiva `push`, che invierà la relativa direttiva di configurazione a tutti i client. Per poter accedere alla rete `192.168.0.0` invece si usa direttamente la direttiva `route` sul server, preceduta dall'impostazione della directory per le configurazioni dei singoli client con `client-config-dir`.

Per potersi collegare al server della configurazione precedente, si potrà invece usare una configurazione client come quella del seguente esempio, in cui di nuovo si sono rimosse righe vuote e commenti:

client.conf

```
client
dev tun
proto udp
remote holland.truelite.it 1194
resolv-retry infinite
port 1194
user nobody
group nogroup
persist-key
persist-tun
ca  Truelite-cacert.pem
cert havnor-cert.pem
key  havnor-key.pem
comp-lzo
verb 3
```

Si noti come le direttive per SSL siano (ad eccezione di `dh`) sostanzialmente analoghe, e come una volta usata la direttiva `client` sia stato sufficiente indicare con `remote` l'indirizzo pubblico del server, senza avere più la necessità di predisporre script da lanciare con `up`.

Capitolo 5

Sistemi di *Intrusion Detection*

5.1 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.

5.1.1 La sicurezza e gli IDS

Come abbiamo illustrato in sez. 1.1 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. 1.1.2). 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.

5.1.2 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. 5.4.2), 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. 5.3.2) 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. 5.4.2) 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. 5.3.

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.

5.2 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.

5.2.1 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. 5.1, 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. 5.3).

Tabella 5.1: 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. 5.2. 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 5.2: 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. 5.1, 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. 5.1 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. 5.3. 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 5.3: 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. 5.3, 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 5.4: 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. 5.4, 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. 5.5. 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 5.5: 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.

5.2.2 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. 5.2.4) 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. 5.6. 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 5.6: 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.

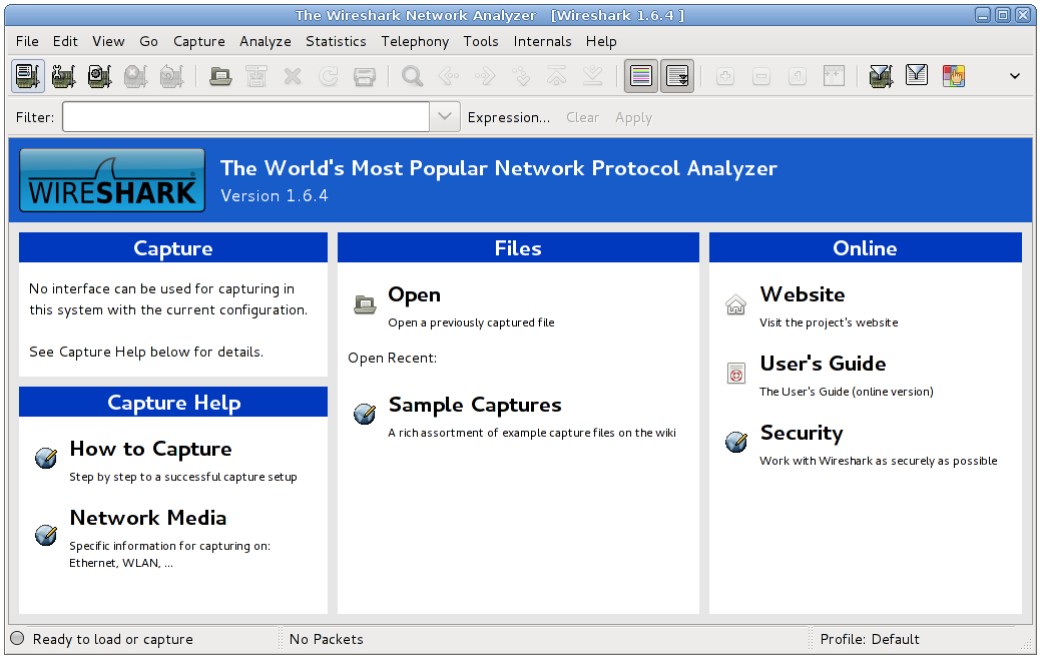


Figura 5.1: Finestra di avvio di `wireshark`.

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. 5.1.

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. 5.1 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.

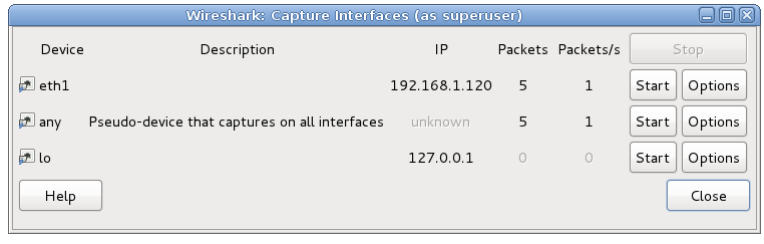


Figura 5.2: Finestra di selezione delle interfacce di `wireshark`.

[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. 5.2, 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. 5.3. 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.

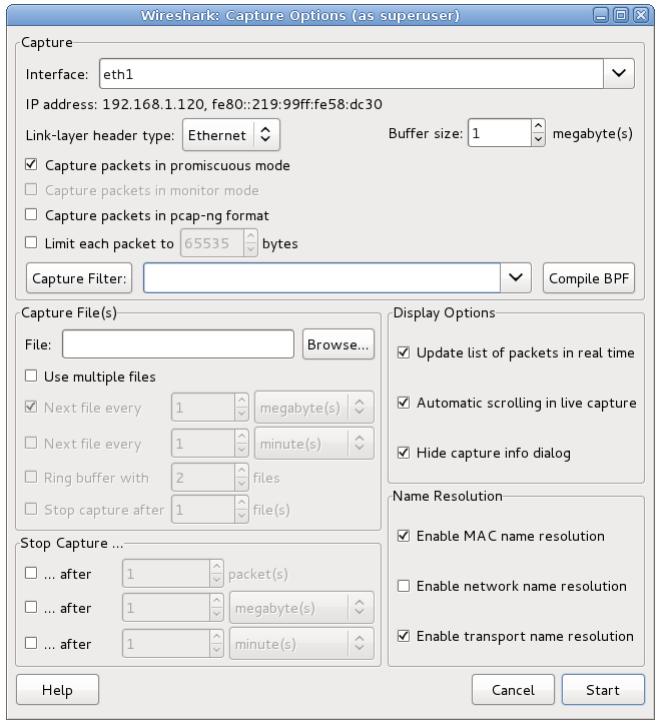


Figura 5.3: Finestra di impostazione delle opzioni di cattura di `wireshark`.

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.

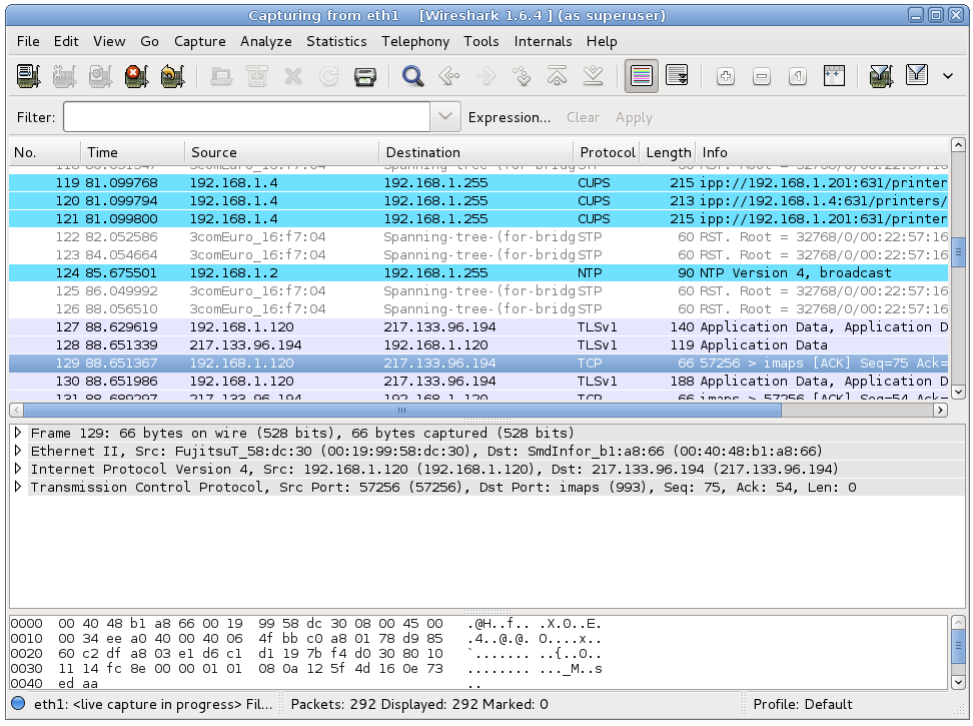


Figura 5.4: Finestra principale di `wireshark`.

Una volta avviata la cattura il programma inizierà a mostrare i pacchetti ricevuti nella sua finestra principale, riportata in fig. 5.4, 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. 5.3), 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. 5.5 dove si è selezionata la parte relativa al protocollo TCP.

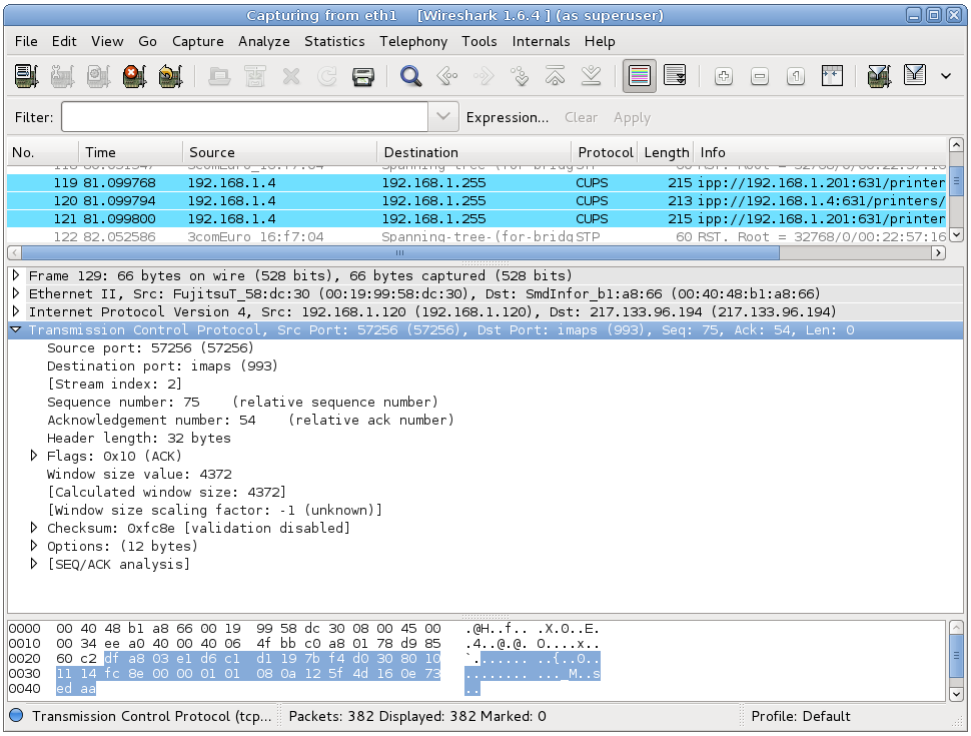


Figura 5.5: Finestra principale di `wireshark`, nella selezione di una parte del pacchetto.

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 5.7: 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. 5.7.

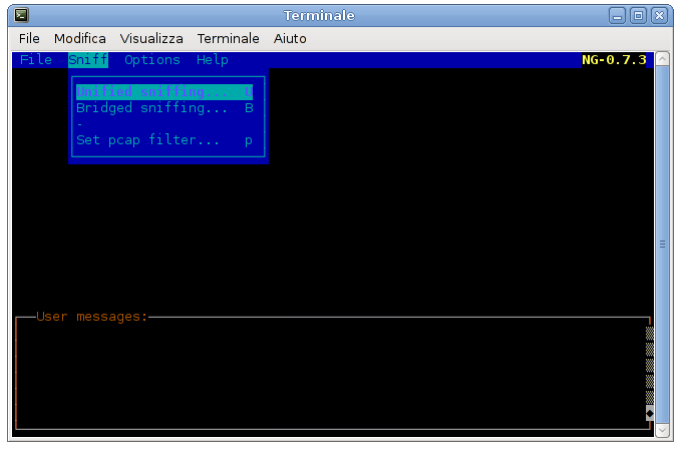


Figura 5.6: La schermata iniziale `ettercap` nella versione semigrafica.

In fig. 5.6 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. 5.7. Dal menù File si possono anche leggere dati già acquisiti, ma non tratteremo questo caso.

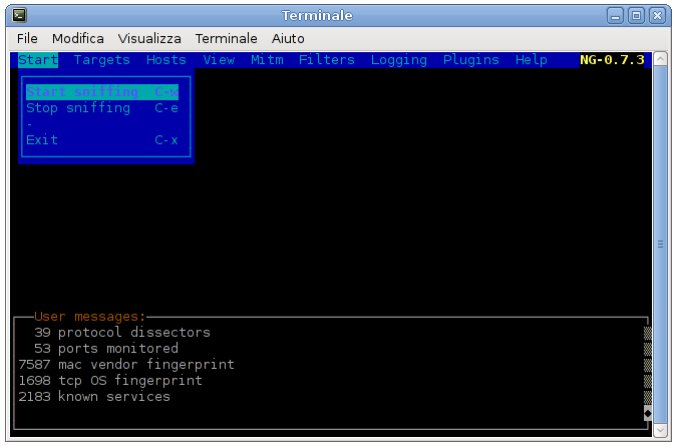


Figura 5.7: La schermata di `ettercap` per la lista delle macchine sulla rete.

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. 5.7. 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.

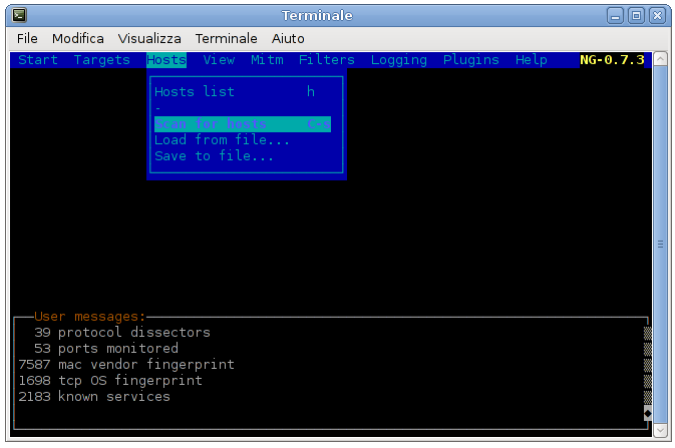


Figura 5.8: La schermata di `ettercap` con il menù per la ricerca delle macchine sulla rete.

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. 5.8.

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.

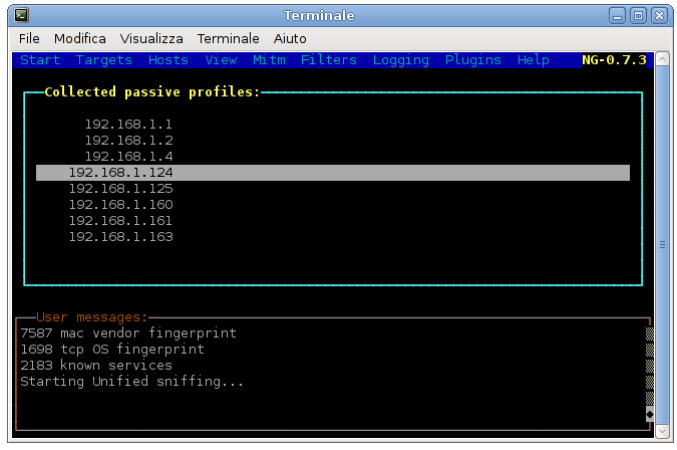


Figura 5.9: La schermata di `ettercap` che mostra l'elenco dei profili raccolti in maniera passiva.

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. 5.9.

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.

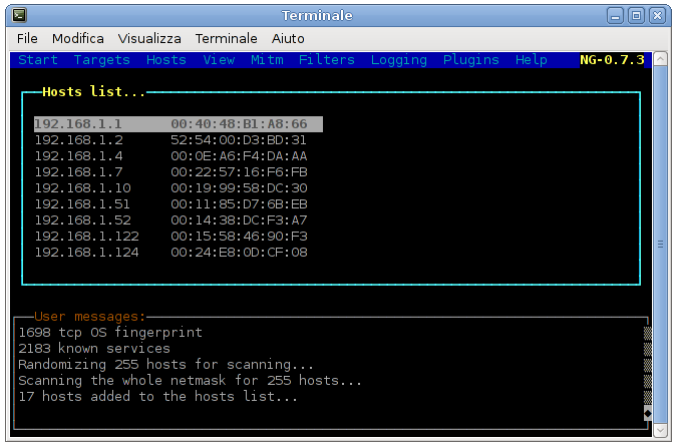


Figura 5.10: La schermata di `ettercap` la lista delle macchine sulla rete.

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. 5.10. 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.

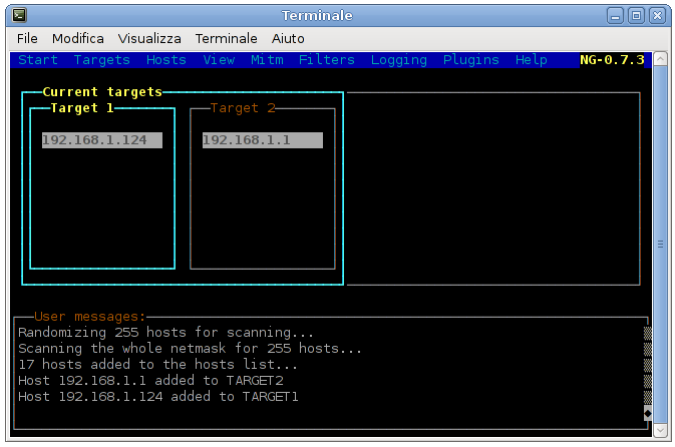


Figura 5.11: La schermata di `ettercap` con la selezione dei bersagli.

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. 5.11. 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.

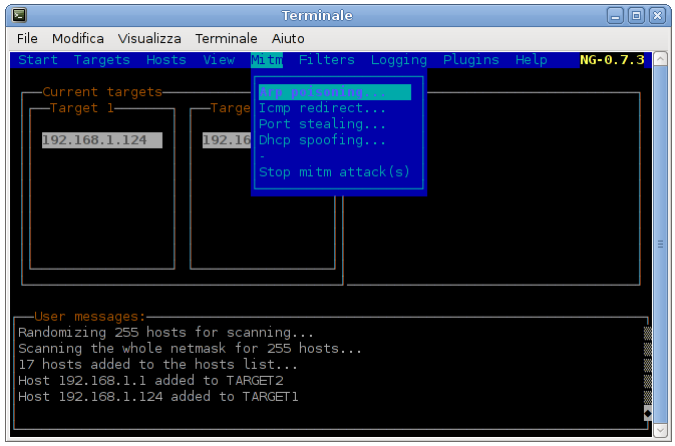


Figura 5.12: La schermata di `ettercap` con la selezione degli attacchi.

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. 5.12, 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. 5.13).

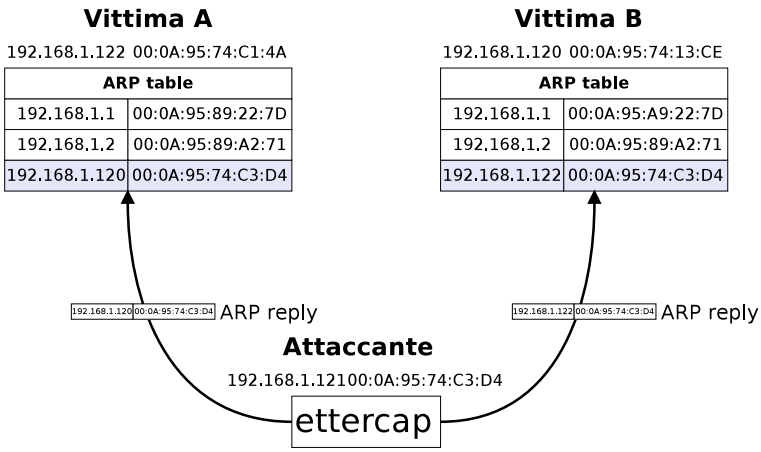


Figura 5.13: Schema di funzionamento dell'*ARP poisoning*.

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. 5.14 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

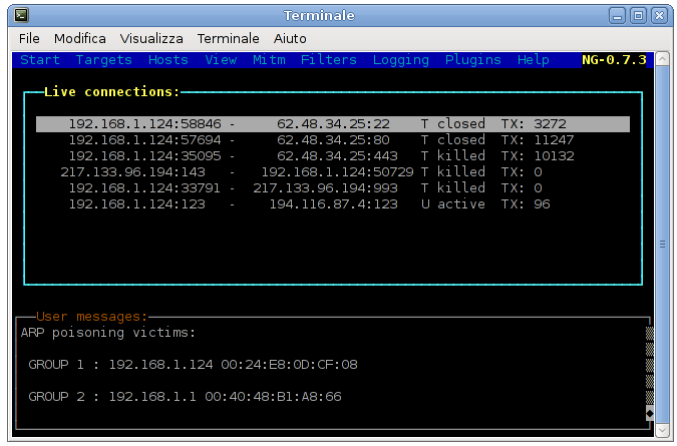


Figura 5.14: La schermata di `ettercap` con la lista delle connessioni rilevate nell'intercettazione del traffico fra due macchine.

indicare le due direzioni dello stesso (se disponibili), come mostrato in fig. 5.15. 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.

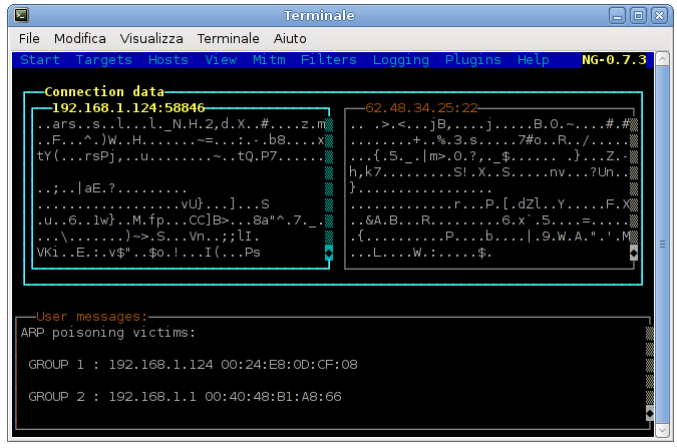


Figura 5.15: La schermata di `ettercap` con il traffico intercettato in una connessione.

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).

5.2.3 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.

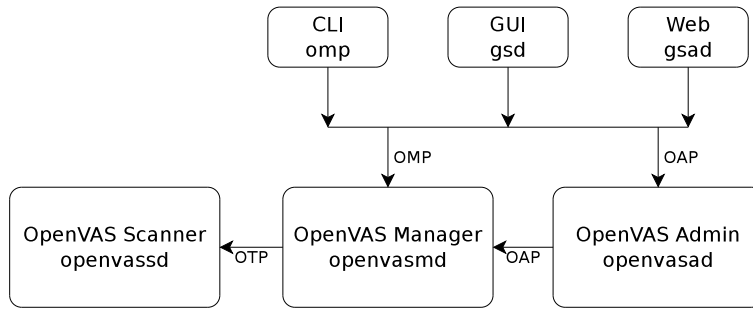


Figura 5.16: Architettura delle varie componenti di OpenVAS.

L'architettura di OpenVAS nelle versioni più recenti è schematicamente illustrata in fig. 5.16; 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 attraverso l'uso di un protocollo di comunicazione chiamato OTP (*OpenVAS Transfer Protocol*).

I client però non interagiscono direttamente con lo scanner, ma si appoggiano ad un demone di gestione, `openvasmd`, l'*OpenVAS Manager*, che cura l'interazione fra gli stessi, cui offre un protocollo di comunicazione denominato OMT (*OpenVAS Management Protocol*) e lo *scanner* per gestire richieste e risposte tramite il protocollo OTP. Inoltre il manager si appoggia al demone `openvasad`, l'*OpenVAS Administrator*, che gestisce gli i criteri di accesso e gli utenti di OpenVAS, utilizzando il protocollo OAP (*OpenVAS Administration Protocol*) fornito da quest'ultimo.

A queste tre componenti server si aggiungono i vari client che consentono agli utenti di richiedere le scansioni ed ottenere i relativi risultati (che in genere interagiscono sia con `openvasmd` via OMP che con `openvasad` via OAP. Il programma ne prevede ben tre, un programma a riga di comando, un programma 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. 2.1. 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. 2.1, 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. 5.16 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 raggiungibile in maniera diretta solo localmente, se lo si vuole in ascolto su un indirizzo generico occorrerà utilizzare l'opzione `-a`. Su Debian sia l'indirizzo che la porta su cui `openvassd` si pone in ascolto vengono impostati in `/etc/default/openvas-scanner` e sono utilizzati dallo script di avvio per lanciare opportunamente il demone.

Le altre opzioni principali di `openvassd` sono riportate in tab. 5.8, 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,

[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.

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, il default è `127.0.0.1`.
`-p port`	specifica la porta su cui si pone in ascolto il server (il default è 9391).
`-S ip1,...`	specifica la lista degli indirizzi che verranno usati come IP sorgente nelle scansioni (devono essere fra quelli assegnati alla macchina).
`-f`	mantiene il server attivo sul terminale.
`-s`	stampa le opzioni di configurazione.

Tabella 5.8: Principali opzioni di `openvassd`.

`/etc/openvas/openvassd.conf`, che consentono di impostare le posizione dei vari file e directory 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. La configurazione corrente può essere stampata con l'opzione `-s`.

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*; nel caso di Debian queste impostazioni sono mantenute nel file `/etc/default/openvas-manager` ed utilizzate direttamente dallo script di avvio per lanciare il demone.

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. 5.9, per l'elenco completo si rimanda al solito alla pagina di manuale accessibile con `man openvasmd`.

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 5.9: 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 --rebuild
```

che consentono di ricostruire (o creare quando eseguiti la prima volta) il database delle informazioni, senza il quale il programma non parte, a questo si aggiunge la compilazione dei dati informativi SCAP (*Security Content Automation Protocol*) e CERT (*Computer Emergency Readiness Team*) relativi alle vulnerabilità che si ottengono con i comandi:

```
# openvas-scapdata-sync
# openvas-certdata-sync
```

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

Opzione	Descrizione
`-a` *addr*	specifica l'indirizzo IP su cui il servizio è in ascolto.
`-p` *port*	indica la porta su cui il servizio è in ascolto.
`-c` *cmd*	esegue un comando OAP (valori possibili `add_user`, `remove_user`, `list_users`).
`-r` *role*	indica il ruolo da assegnare quando si crea un utente, può assumere i valori `admin` o `user`.
`-n` *nome*	indica il nome utente.
`-w` *pass*	indica la password.
`-f`	mantiene il server attivo sul terminale.

Tabella 5.10: Principali opzioni di `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. 5.17.

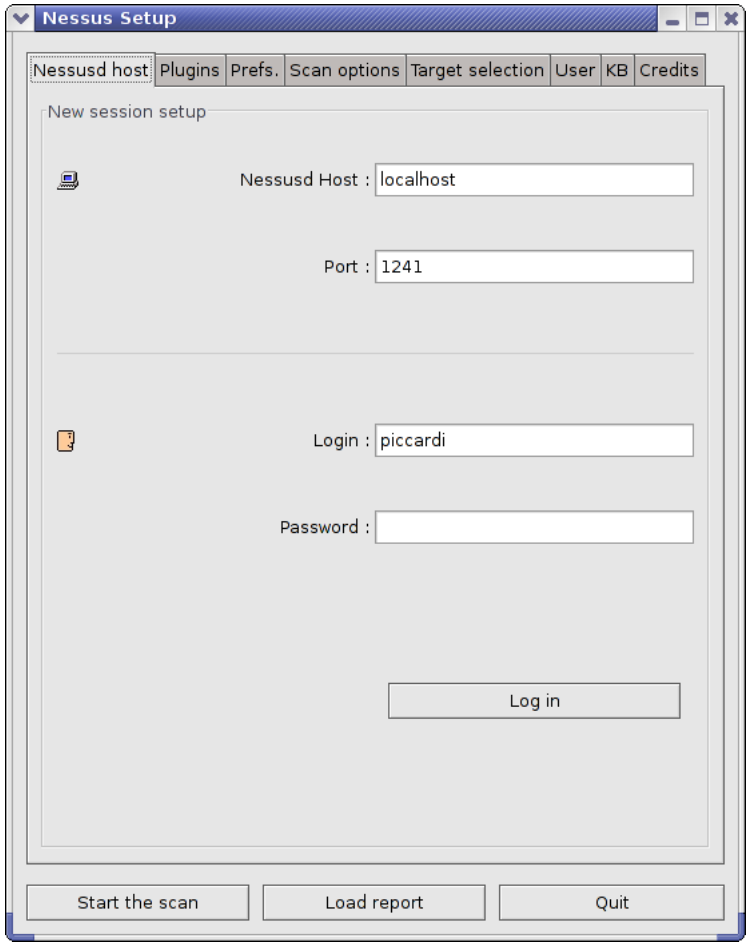


Figura 5.17: Finestra di login di `openvas`.

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. 5.18, avvisando al contempo, con una ulteriore finestra in pop-up, che i plug-in potenzialmente pericolosi

sono stati disabilitati.

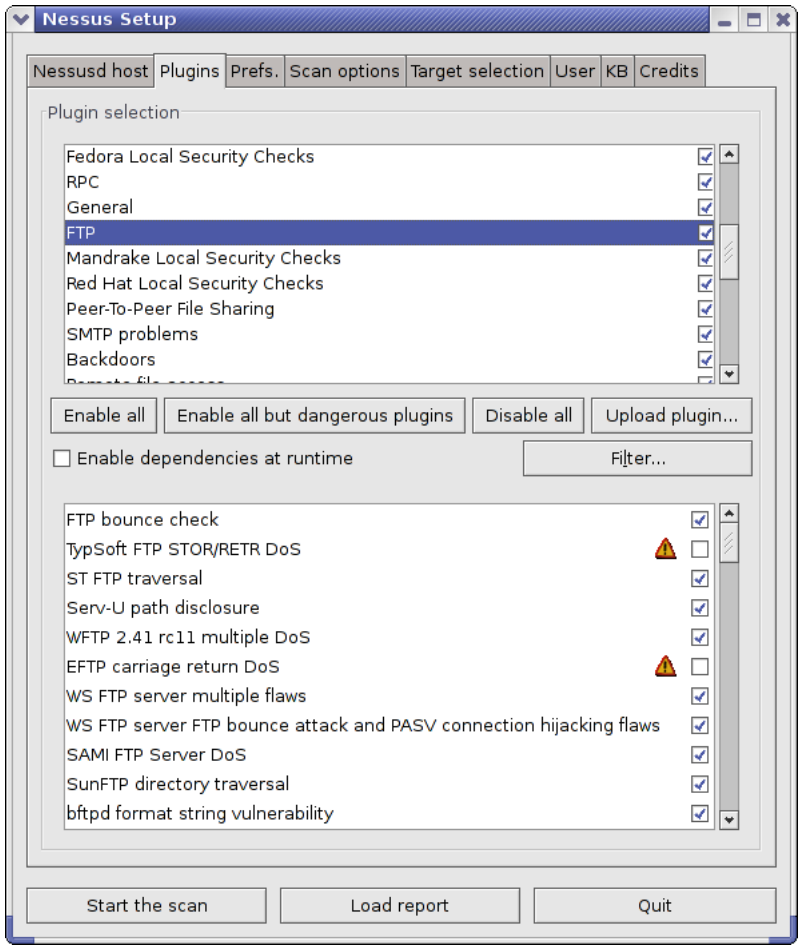


Figura 5.18: Finestra di selezione dei plugin di `openvas`.

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 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. 5.19. 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 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.

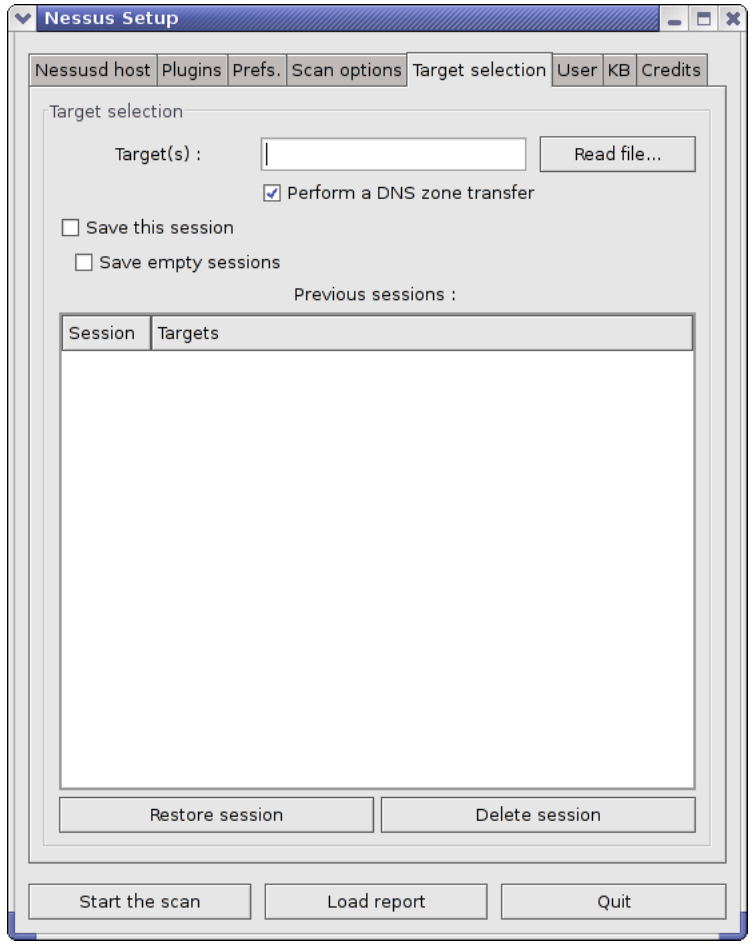


Figura 5.19: Finestra dei bersagli della scansione di `openvas`.

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.[24]

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 una finestra di progresso del tipo di quella riportata in fig. 5.20. 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.

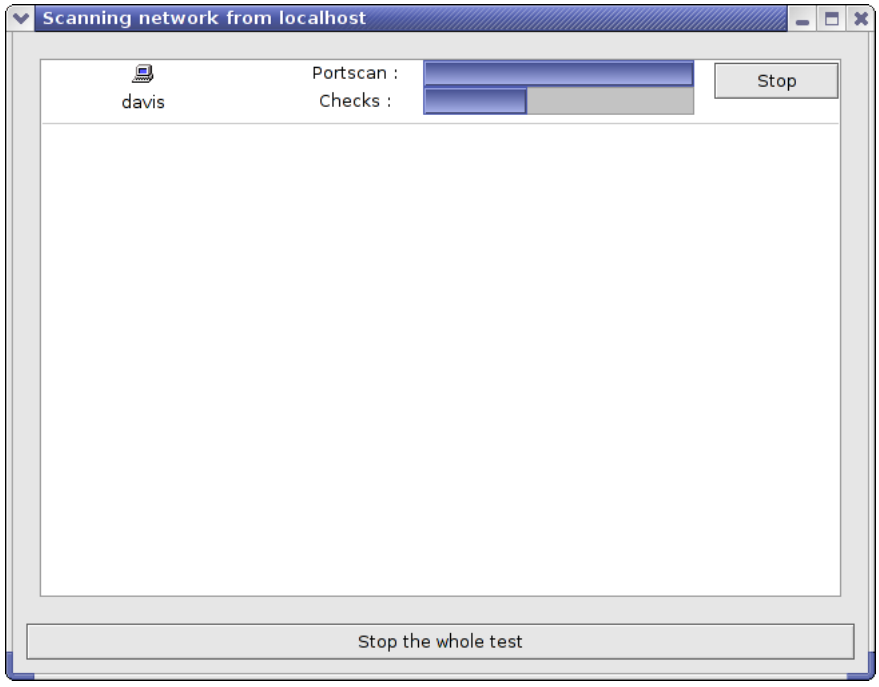


Figura 5.20: Finestra del progresso delle operazioni di openvas.

Una volta completata la scansione openvas genererà un rapporto dei risultati ottenuti in una finestra apposita, come quella riportata in fig. 5.21. 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

[24] 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.

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. 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.

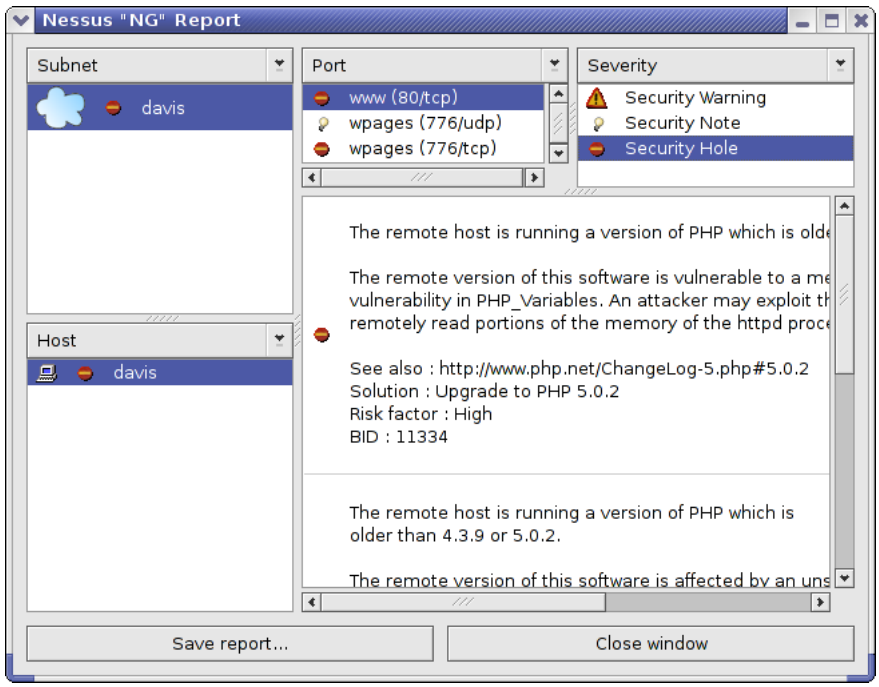


Figura 5.21: Finestra dei rapporti dei risultati di `openvas`.

5.2.4 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. 5.22.

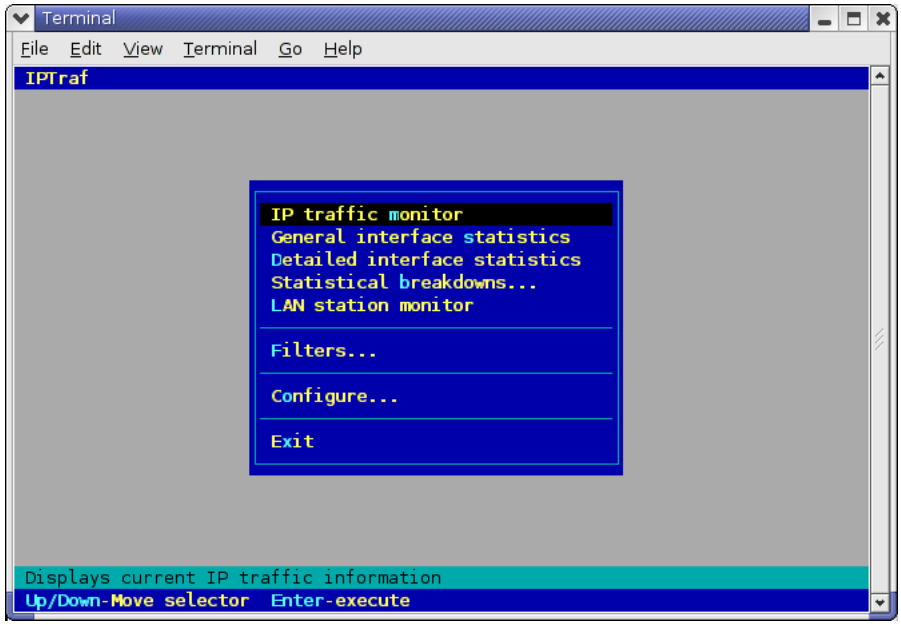


Figura 5.22: Menù principale di `iptraf`.

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 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. 5.22, 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. 5.23) 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. 5.24, in cui vengono mostrati i risultati della cattura dei pacchetti in tempo reale.

Come si può notare in fig. 5.24 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

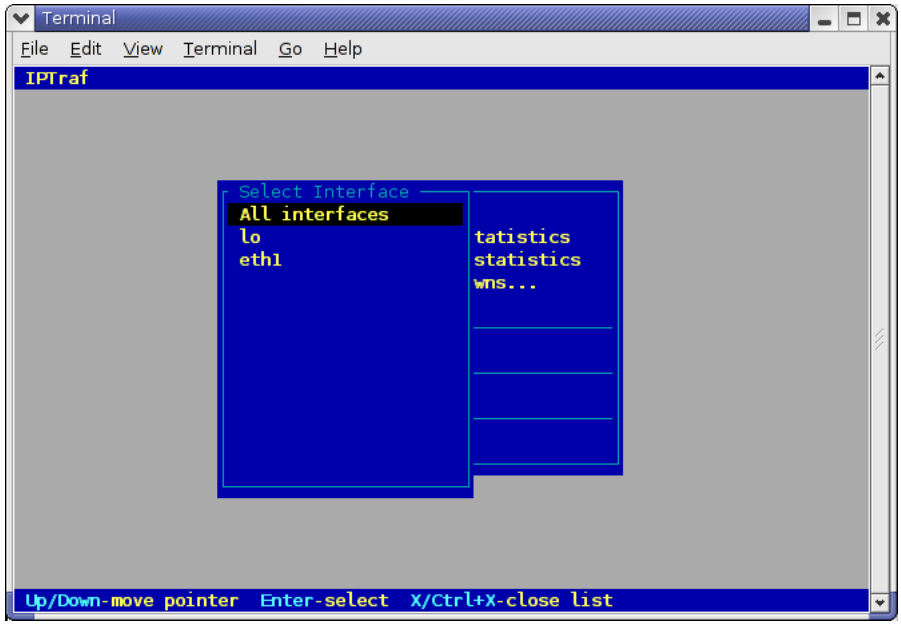


Figura 5.23: Menù di selezione delle interfacce di `iptraf`.

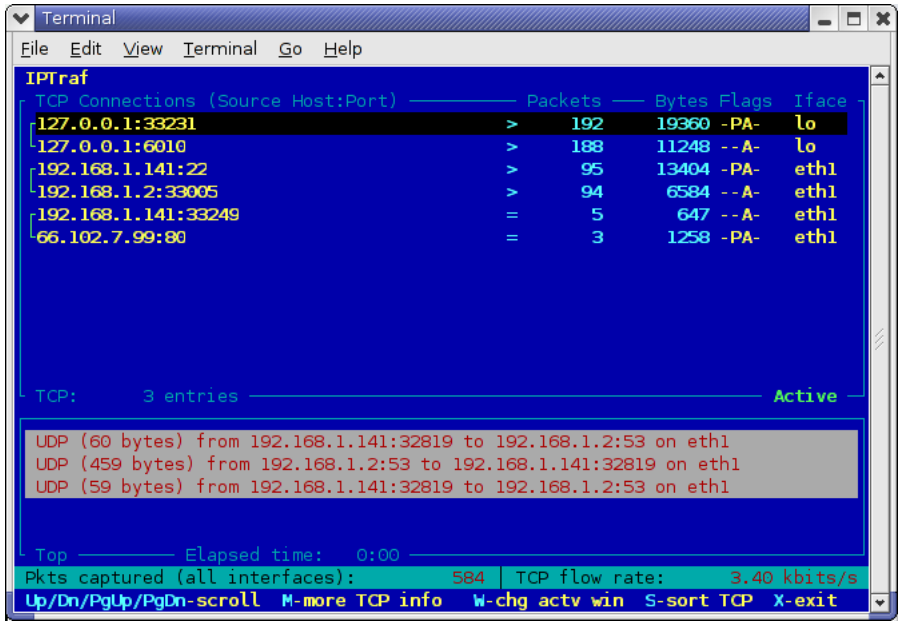


Figura 5.24: Finestra del monitor del traffico di `iptraf`.

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. 5.24 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. 5.25), che raccoglie i dati relativi a tutte le interfacce presenti sul sistema, e che riporta un sommario delle informazioni essenziali relative a ciascuna interfaccia.

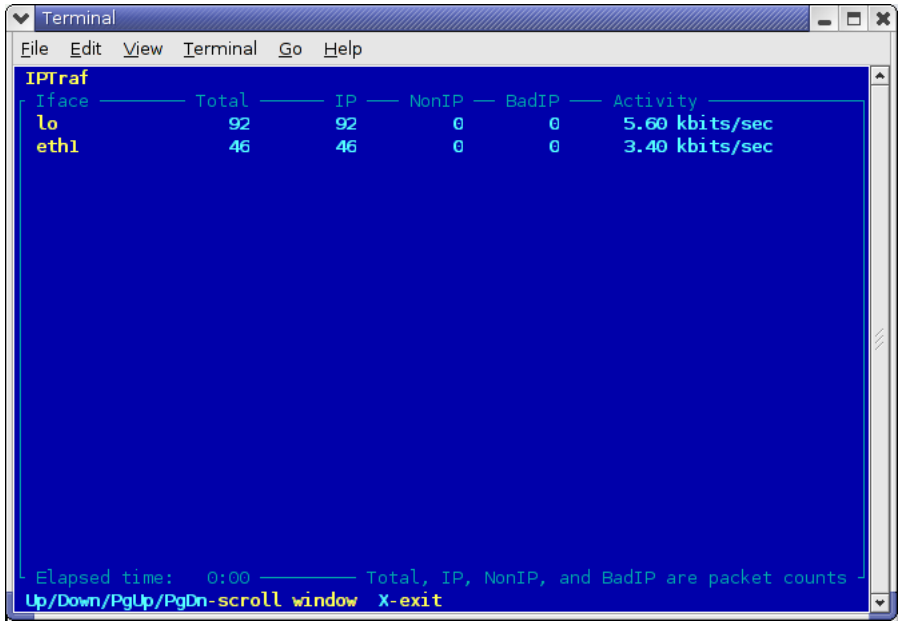


Figura 5.25: Finestra delle statistiche generali di `iptraf`.

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

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. 5.26 si dovrà nuovamente scegliere l'interfaccia da un menù, analogo a quello di fig. 5.23, 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.

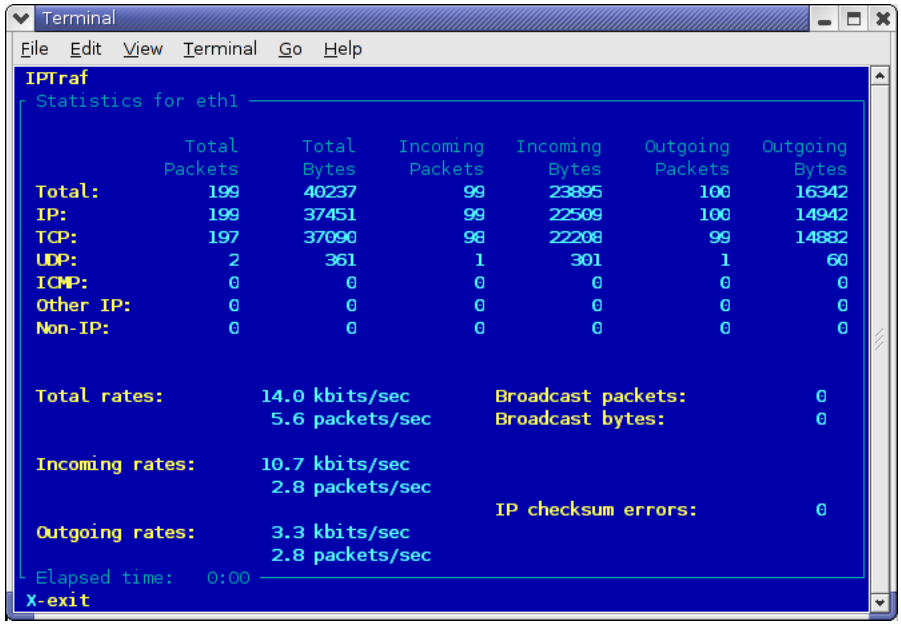


Figura 5.26: Finestra delle statistiche dettagliate per interfaccia di `iptraf`.

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.

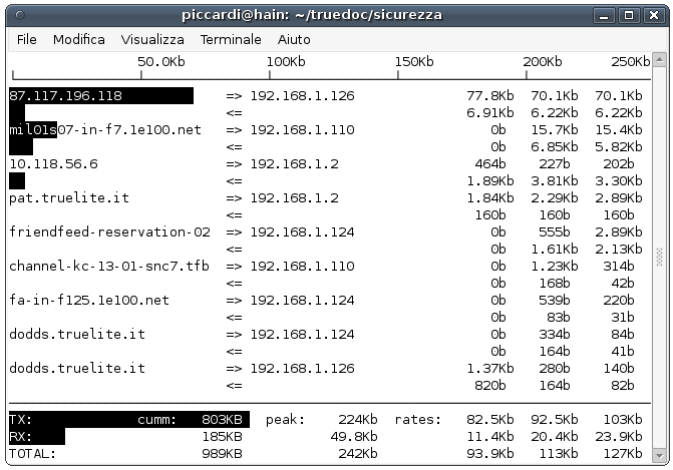


Figura 5.27: Finestra di `iftop`.

Il comando non richiede nessun argomento o opzione obbligatoria, e se lanciato senza specificare 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. 5.27.

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. 5.11, 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 5.11: 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. 5.12.

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:

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 5.12: I messaggi di allarme di `arpwatch`.

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`.

5.3 I *sistemi antintrusione locali*

Come accennato in sez. 5.1 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.

5.3.1 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[25] 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,[26] 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 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
```

[25] 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.

[26] 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.

```
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.

5.3.2 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.[27]

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.

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

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. 5.28, 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`.

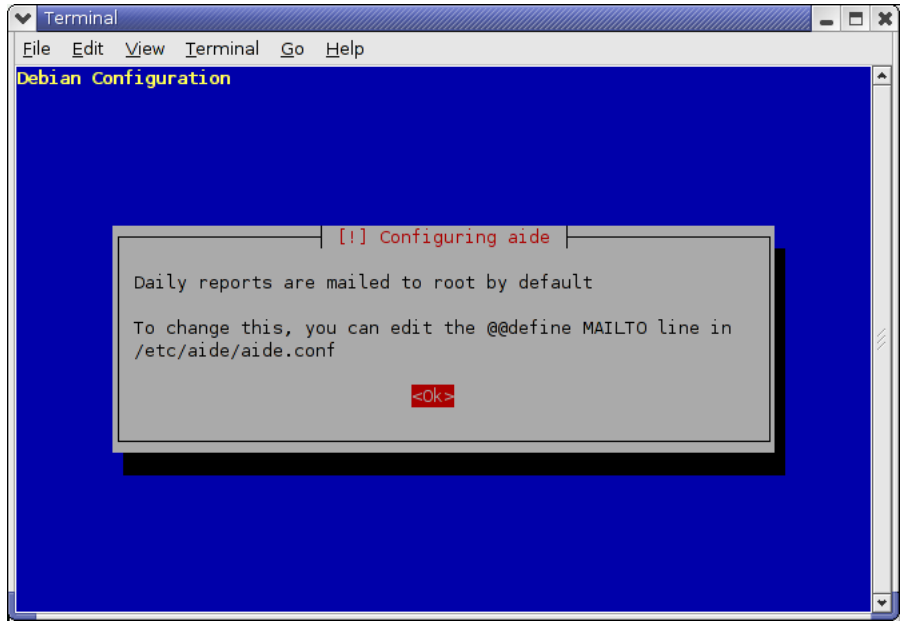


Figura 5.28: Prima finestra di configurazione dell'installazione di `aide`.

Una volta letta la schermata e premuto invio, la configurazione ne propone una seconda, riportata in fig. 5.29, 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

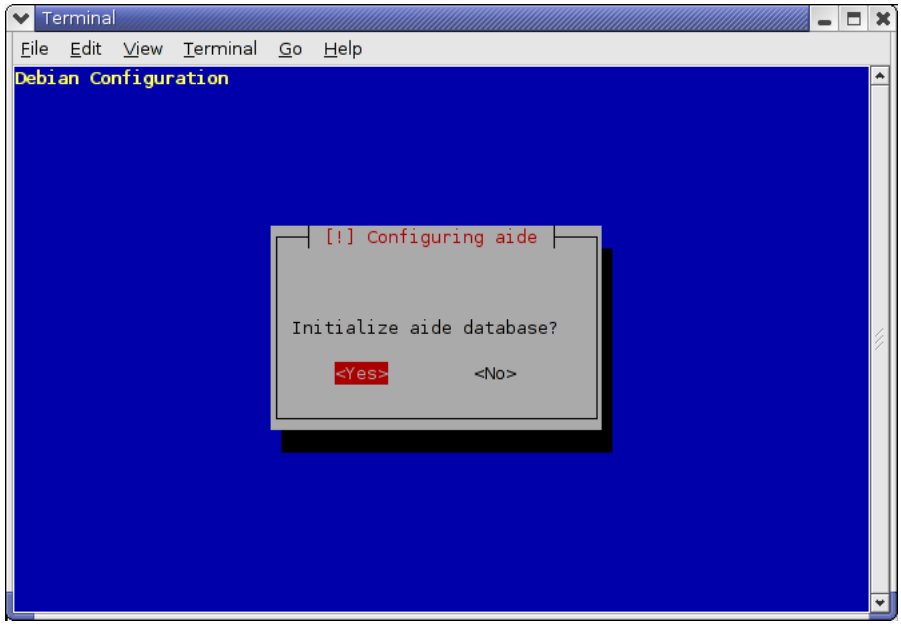


Figura 5.29: Seconda finestra di configurazione nell'installazione di `aide`.

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*[28] 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:

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

```
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`.

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 5.13: Gruppi predefiniti per le proprietà dei file in `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. 5.13, 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.

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.

5.4 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.

5.4.1 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. 5.2.2 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*.[29]

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

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

5.4.2 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. 5.2.2. 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`[30] 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`.

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. 5.14. In questo

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

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

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. 5.15, per l'elenco completo si faccia al solito riferimento alla pagina di manuale.

5.4.3 La configurazione di `snort` come NIDS

Come accennato in sez. 5.4.2 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 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. 5.16 l'elenco completo è riportato nello *Snort User Manual* disponibile su `http://www.snort.com/`.

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 5.15: Principali opzioni 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

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. 5.4.4), è equivalente all'opzione -o.

Tabella 5.16: Principali opzioni della direttiva config nel file di configurazione di snort.

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*,[31] 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

[31] chiama così la tecnica, usata anche da `nmap` come accennato in sez. 5.2.1, 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. 5.2.4.

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
```

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.[32]

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

[32] 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.

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. 5.4.4, 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
...
```

5.4.4 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, 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. 5.17, 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 5.17: 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. 5.4.3. 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.[33] Questi sono tre e sono usati principalmente per la 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

[33] la struttura dell'intestazione di IP è riportata in fig. 3.7.

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. 5.18. 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+;
```

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 5.18: Identificatori dei flag TCP nell'opzione `flag`.

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, 5.2.1, che è contraddistinto da un valore nullo di tale campo. L'opzione richiede un valore numerico; un esempio è il seguente:

```
ack: 0;
```

itype imposta un criterio di corrispondenza sul valore del campo *type* di un pacchetto ICMP,[34] 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:

[34] una spiegazione sul significato di questi pacchetti è stata affrontata in sez.3.4.4, ed in tab.3.10 e tab.3.11 sono riportati i valori numerici dei vari campi del protocollo.

```
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. 5.19, separate da virgole; un possibile esempio è il seguente:

```
resp: rst_all
```

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 5.19: Significato delle azioni dell'opzione `resp`.

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:

```
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 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful 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, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using

public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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:

A. 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.

B. 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 fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. 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.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve 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.

J. 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.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. 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.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket 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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

A.11 Relicensing

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

A.12 Addendum: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Indice analitico

Bibliografia

[AIPAsec] Autorità per l'Informatica nella Pubblica Amministrazione. La sicurezza dei servizi in rete requisiti, modelli, metodi e strumenti. Technical report, AIPA, 2001.

[ApplCrypt] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[CompSec] Matt Bishop. *Computer Security*. Addison Wesley, 2003.

[GaPiL] Simone Piccardi. *Guida alla Programmazione in Linux*. http://gapil.gnulinux.it, 2012.

[IS-LDAP] Simone Piccardi. *Integrazione sistemistica con LDAP*. http://labs.truelite.it, 2011.

[ShannonCrypt] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 1949.

[StS] Aleph1. Smashing the stack for fun and profit. *Phrack*, 1996.

[WebServ] Simone Piccardi. *I servizi web*. http://labs.truelite.it, 2003.