UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Studio e analisi degli attacchi "ARP poisoning" e delle possibili contromisure

Relatore: Prof.ssa Emilia Rosti
Correlatore: Prof. Danilo Bruschi

Tesi di Laurea di
Alberto Marco Ornaghi
Matricola 562686

Anno Accademico 2001-2002

A *mio nonno*,
per avermi aiutato a scrivere
quel manuale che si chiama "vita".

Ringraziamenti

Ringrazio la Professoressa Emilia Rosti e il Professore Danilo Bruschi per avermi dato la possibilità di svolgere questa tesi e per avermi aiutato a completarla.

Un grazie di cuore ai miei genitori per avermi dato così tanto e non aver mai chiesto nulla in cambio. Non dimenticherò mai quella mail...

È doveroso ricordare: Lorenzo (gigi sullivan) Cavallaro, per avermi consigliato alcune "dritte" nei punti giusti; gli amici di antifork e #phrack.it con cui scambio giornalmente idee su come realizzare le cose più strane; Marco (naga) Valleri con cui ho condiviso il progetto "ettercap" senza il quale il mio interesse nei confronti dell' "ARP poisoning" non sarebbe stato così elevato.

Infine, ma non per importanza, un pensiero speciale alla persona che più di tutte mi è sempre vicino, la mia Anna.

Indice

Capitolo 1

Introduzione

I protocolli fondamentali che vengono utilizzati attualmente nella comunicazione tra host sono stati progettati attorno agli anni 1970-1980. In quel periodo, Internet come la conosciamo ora stava ancora nascendo e la rete che collegava le principali università americane era ARPANET. Il numero di host connessi e la larghezza di banda erano molto inferiori rispetto all'attuale Internet. Data la ridotta capacità di calcolo e larghezza di banda, il principale scopo dei protocolli progettati per la comunicazione erano l'efficienza e la sopravvivenza della connettività in caso di guasti. La sicurezza venne involontariamente trascurata perché si assumeva che gli host facenti parte della rete fossero tutti fidati.
Il dipartimento della difesa americano commissionò ad una università della California un protocollo che fosse in grado di mettere in comunicazione due host anche dopo malfunzionamenti di parti della rete che li connette. Questo comportamento era dettato dalla necessità che due basi militari fossero in grado di comunicare anche se una base intermedia fosse stata distrutta da un attacco nemico. Ciò non era possibile mediante le normali reti telefoniche in quanto una volta stabilita la connessione i dati sono inviati sempre sullo stesso cavo. Se il cavo viene interrotto, la comunicazione cade con esso. Fu così sviluppato il protocollo IP in grado di instradare datagrammi su percorsi differenti per raggiungere la medesima destinazione. Il presupposto fondamentale era che gli host appartenenti a questa rete si potessero fidare l'uno dell'altro. Problemi quali lo "spoofing" dei messaggi non vennero considerati come di primaria importanza.
Internet come la conosciamo oggi è notevolmente cambiata. Gli host con-

nessi tra loro sono milioni e non appartengono più tutti allo stesso ente (negli anni settanta il dipartimento della difesa). La rete è continuamente sotto la minaccia di attacchi informatici sferrati da host maligni. Secondo i dati riportati dal CERT-CC [1], il numero di questi attacchi è in costante aumento. Negli ultimi anni in particolare si sta assistendo anche ad una crescita, del numero di attacchi portati a termine dall'interno della rete (anche se spesso questi attacchi non sono riportati al CERT per politiche aziendali). Gli attacchi interni sono spesso i più insidiosi. Infatti, molte aziende si preoccupano di difendere solo il perimetro della propria rete, trascurando completamente il problema dei dipendenti che possono perpetrare azioni offensive dall'interno. Per questo motivo è necessario porre l'attenzione anche su attacchi che possono essere portati a termine su rete locale o Local Area Network (LAN). Questi attacchi possono essere molto pericolosi e non devono essere sottovalutati. Infatti sfruttando tecniche di tipo "man in the middle" un attaccante può arrivare a compromettere comunicazioni cifrate o modificare i dati di una connessione in corso.

Un protocollo comunemente usato nelle reti locali che comunicano usando TCP/IP [2] su Ethernet [3] è ARP [4]. Ethernet è utilizzato nella quasi totalità delle reti locali e di conseguenza anche il protocollo ARP viene utilizzato da host che voglia comunicare con un altro host sulla LAN. Il protocollo di risoluzione indirizzi (Address Resolution Protocol, ARP) è responsabile di risolvere gli indirizzi IP in indirizzi Ethernet in modo che le schede di rete possano interpretare correttamente i frame inviati sul mezzo fisico. Questo protocollo presenta notevole insicurezza per quanto riguarda la procedura di inserimento dei dati all'interno del kernel di sistema. Il protocollo ARP non mantiene uno stato interno e questo permette di ricevere e accettare risposte senza mai aver effettuato le domande. Inoltre è possibile che un particolare di messaggio di risposta, chiamato "gratuitous ARP reply", sia inviato da un host per annunciare un cambiamento di indirizzo Ethernet. Questo messaggio viene elaborato da tutti gli host della rete che ne memorizzano il contenuto, associando un nuovo indirizzo Ethernet all'IP di chi ha spedito il messaggio. Poiché ARP non garantisce l'identità del mittente, un attaccante può mandare falsi messaggi spacciandosi per un host vittima. Questo comporta un aggiornamento dell'indirizzo Ethernet associato all'IP fasullo contenuto nella risposta. Un attacco così strutturato è chiamato "ARP poisoning" poiché consiste nell'avvelenare la cache ARP degli host

della rete. Questa tecnica consiste nel modificare, senza che la vittima se ne accorga, le tabelle che contengono le associazioni <IP, MAC address>. In questo modo i pacchetti spediti ad un determinato IP "avvelenato" saranno in realtà incapsulati in frame con un MAC address di destinazione che non corrisponde a quello reale. L'attaccante può quindi deviare il flusso dei frame Ethernet verso di sè e inoltrare i pacchetti al reale destinatario. Potendo intercettare e inoltrare il traffico al corretto destinatario, l'attaccante è a tutti gli effetti nel mezzo della comunicazione. Questa azione è chiamata attacco "man in the middle". Da questa favorevole posizione l'attaccante è in grado di modificare il contenuto della comunicazione, inserire nuovi pacchetti, fornire certificati falsi per inserirsi nelle connessioni cifrate. Questi attacchi possono portare conseguenze molto serie. Basti pensare ad una connessione cifrata verso un server che richiede l'utilizzo di carta di credito. Un host nella stessa rete locale potrebbe essere nel mezzo e immagazzinare tutti questi dati sensibili.

In questa tesi si proporrà un protocollo ARP sicuro che impedisce ad un attaccante di effettuare ARP poisoning.

La prima parte della tesi sarà incentrata sull'analisi delle vulnerabilità del protocollo ARP e delle cause che portano all'attuazione di attacchi man in the middle sfruttando la tecnica "ARP poisoning". Verranno analizzati gli attacchi che possono essere lanciati quando l'attaccante si inserisce nel mezzo dei una connessione, quali "sniffing", "hijaking", "command injection", "malicious code injection" e "content filtering on the fly". Particolare attenzione verrà data anche alle conseguenze che questi attacchi possono avere. La compromissione di programmi per connessioni cifrate come Secure Shell (SSH) o Secure Socket Layer (SLL) e la possibilità di sniffing su reti connesse tramite switch o reti wireless, sono solo alcune delle gravi rischi che questo tipo di attacchi comporta.

Sarà inoltre analizzata l'implementazione del protocollo ARP nel kernel di linux 2.4.x e le soluzioni proposte per il problema dell'ARP poisoning, in particolare la soluzione proposta nel Secure Link Layer si basa sulla cifratura di tutto il traffico a livello link e opera in modalità kernel. Verrà effettuato un confronto tra questa soluzione e il protocollo che sviluppato nel corso della tesi (Secure ARP protocol).

Poiché per prevenire l'ARP poisoning è sufficiente l'autenticazione dei messaggi ARP, è stato implementato un nuovo protocollo, Secure ARP, che si limita ad aggiunge autenticazione forte al protocollo ARP esistente.

Il protocollo Secure ARP è strutturato come un'estensione del protocollo ARP esistente mediante aggiunta dell'autenticazione del mittente delle risposte ARP per poterne garantire la provenienza. L'autenticazione dei soggetti (e quindi dei mittenti dei messaggi ARP) viene realizzata mediante l'utilizzo di firma digitale. Secondo questo schema, ogni host della rete è provvisto di una coppia di chiavi pubblica/privata. Le chiavi pubbliche degli host della rete verranno inserite, tramite un canale sicuro, nel database di un server per la gestione (distribuzione e certificazione) delle chiavi. Quest'ultimo è un host della rete che svolge anche funzione di distribuzione delle chiavi pubbliche per chi ne fa richiesta e di certificatore di autenticità delle stesse.

I messaggi Secure ARP sono firmati mediante l'algoritmo DSA. Quando un host riceve una SARP reply ne verifica l'autenticità. Effettuerà quindi una richiesta al gestore delle chiavi il quale, se l'IP corrispondente è presente nel suo database, risponderà con la chiave pubblica relativa. Una volta in possesso della chiave pubblica la verifica può essere effettuata. Se la firma corrisponde ai dati contenuti nel messaggio, questi vengono inseriti nelle tabelle di sistema, altrimenti il messaggio viene rifiutato.
Le chiavi pubbliche vengono memorizzate dagli host in una apposita cache per le chiavi in modo da non dover sovraccaricare il gestore delle chiavi di richieste che sono già state effettuate in precedenza. Tuttavia può accadere che un host generi una nuova coppia di chiavi asimmetriche. In questo caso la cache degli host e il database dell'AKD si troverebbero in uno stato inconsistente. Per risolvere questo problema, ogni qualvolta una chiave presa dalla cache non verifichi la firma riportata in un messaggio, questa viene richiesta nuovamente all'AKD. Se la chiave è la stessa, il messaggio viene scartato, altrimenti la chiave nella cache viene aggiornata e si procede alla normale verifica. Opportuni controlli sono stati aggiunti per evitare attacchi di tipo "replay".

Il protocollo Secure ARP è stato progettato per funzionare sia in ambienti dove gli indirizzi IP sono assegnati staticamente, sia laddove è presente un server DHCP per l'assegnazione dinamica degli indirizzi. In quest'ultimo

caso, si renderà necessaria una modifica al codice del DHCP client/server per supportare il protocollo SARP.

Capitolo 2

ARP: Address Resolution Protocol

Il protocollo di risoluzione degli indirizzi (Address Resolution Protocol, ARP) opera come anello di congiunzione tra il livello 3 (network) e il livello 2 (data-link) dello stack ISO/OSI. Il suo scopo è quello di tradurre un indirizzo IP nel suo corrispondente indirizzo di livello data-link. Nel seguito si farà riferimento solo al mezzo fisico più usato: Ethernet.

In questo capitolo verrà introdotto brevemente il funzionamento dello stack TCP/IP su Ethernet, illustrando tramite un esempio come avviene la comunicazione tra due host della stessa rete locale (Local Area Network). In seguito sarà descritto come i dati attraversino i livelli del TCP per poi giungere sul mezzo fisico incapsulati nei frame Ethernet. Particolare attenzione sarà data al passaggio dal livello 3 al 2 dove opera appunto il protocollo ARP. Sarà mostrato il funzionamento di questo protocollo soffermandosi sui dettagli implementativi e introducendo il concetto dei "Gratuitous ARP". Questa funzionalità verrà poi analizzata come il punto debole del protocollo e sarà trattata in modo più approfondito nel capitolo 3.

2.1 Introduzione

Per ridurre la complessità di progettazione, la maggior parte delle reti è organizzata a livelli, ognuno dei quali viene costruito sopra a quello inferiore. Il numero di livelli, il nome di ciascun livello e le funzionalità offerte

cambiano a seconda della tipologia di rete [5]. Tuttavia esiste uno standard di riferimento OSI (Open System Interconnection [6]) sul quale è possibile mappare le diverse implementazioni esistenti. Questo modello consiste di 7 livelli (come mostrato in figura 2.1).

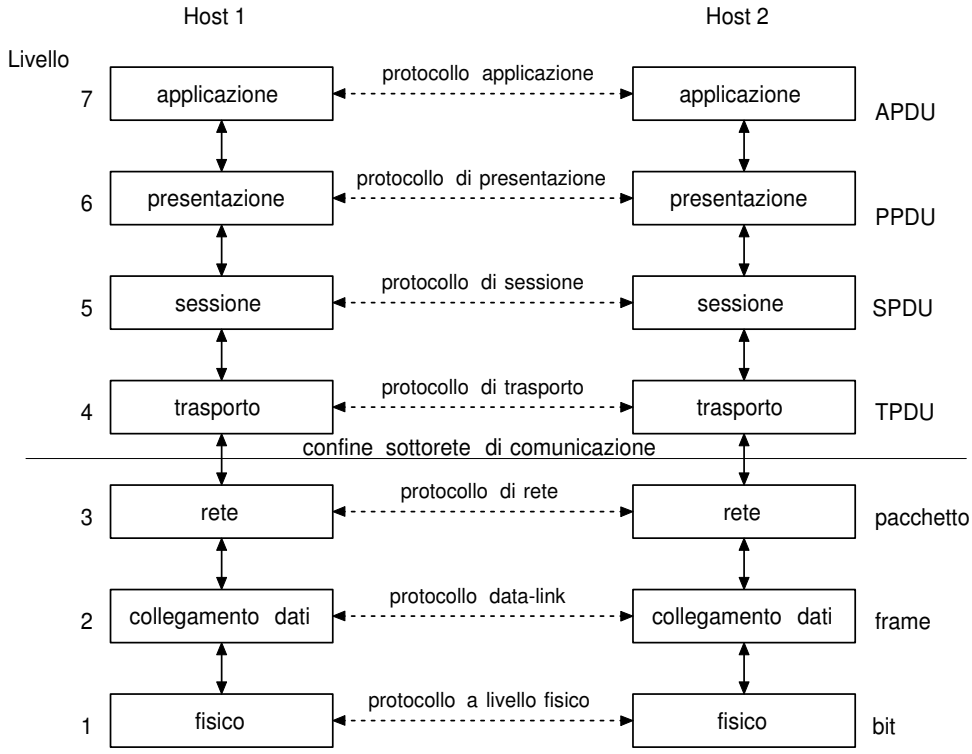

Figura 2.1: Il modello di riferimento OSI.

- **Il livello fisico** fa riferimento alla trasmissione dei bit sul canale di comunicazione. Si preoccupa di codificare correttamente i bit per permettere una trasmissione affidabile da un capo all'altro del mezzo fisico.

- **Il livello data-link** ha come scopo la trasformazione di un canale grezzo in una linea di livello superiore esente da errori di trasmissione.

- **Il livello network** si fa carico di determinare come i pacchetti devono essere instradati dalla sorgente verso la destinazione. Per gestire i percorsi si utilizzano protocolli di routing statici o dinamici.

- **Il livello di trasporto** ha il compito di accettare dati dal livello superiore, spezzarli in pacchetti da passare al livello rete sottostante e assicurarsi che tutti i frammenti arrivino a destinazione.

- **Il livello di sessione** permette ad utenti su macchine differenti di stabilire sessioni, le quali permettono il trasporto ordinato di dati.
- **Il livello di presentazione** fa riferimento alla sintassi e alla semantica delle informazioni trasmesse. Un esempio classico è la codifica dei dati secondo uno standard riconosciuto.
- **Il livello di applicazione** contiene i protocolli applicativi che sono impiegati tra due o più processi per comunicare.

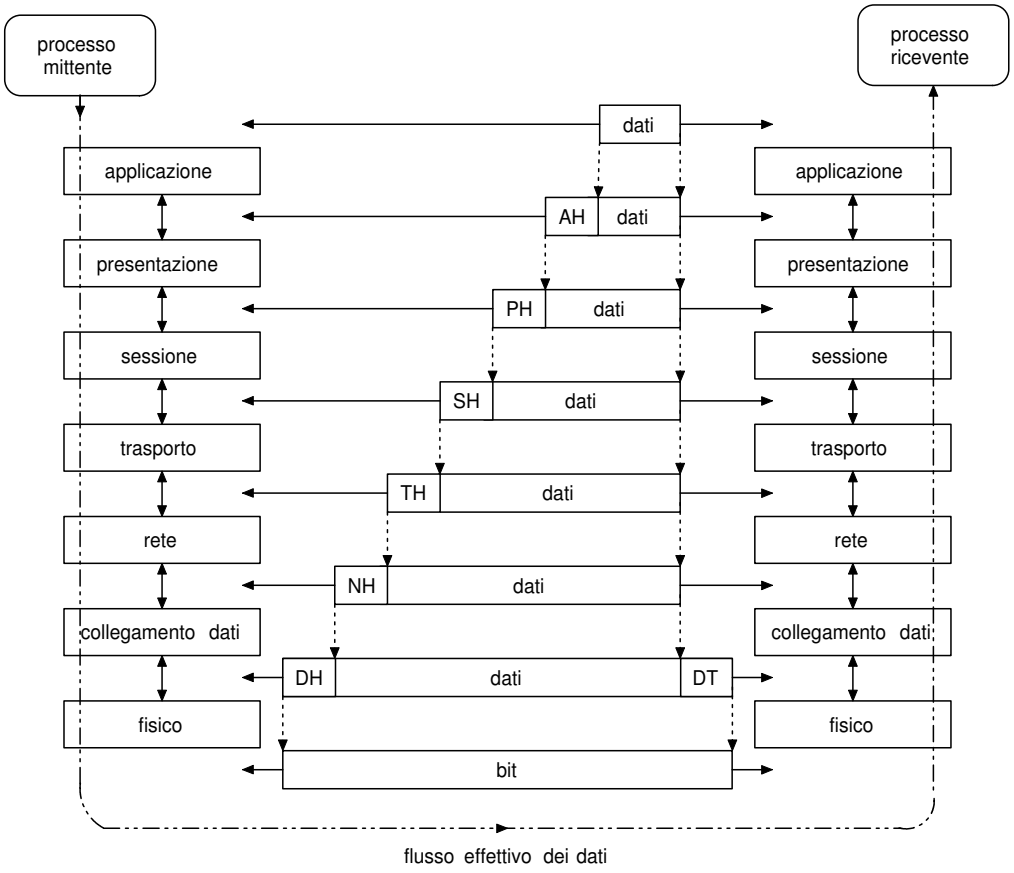

Figura 2.2: Funzionamento del modello OSI.

La figura 2.2 mostra come ogni livello aggiunga una propria intestazione (header) ai dati veri e propri (payload) per quel livello che provengono dal livello immediatamente superiore. Secondo questo schema i dati, partendo dal livello applicazione, discendono tutto lo stack OSI e vengono trasmessi sul canale fisico. Durante questa discesa, vengono opportunamente incapsulati con l'aggiunta di un nuovo header per ogni livello attraversato. Una

volta che i dati sono stati ricevuti dall'altro capo della comunicazione, vengono fatti risalire lungo lo stack, vagliando ad uno ad uno gli header dei rispettivi livelli (o peer). A questo punto i dati hanno raggiunto il livello applicazione remoto corrispondente.

2.1.1 I livelli TCP/IP

Lo scopo principale del dipartimento della difesa americano con lo sviluppo del TCP/IP, era la creazione di un protocollo robusto in grado di funzionare anche in presenza di malfunzionamento di parti della rete, vale a dire un protocollo in grado di trasmettere dati finché la sorgente e la destinazione fossero rimaste attive, anche se alcune delle linee di interconnessione fossero state messe fuori uso. La premessa che tutti gli host appartengano alla stessa rete di computer "fidati" è fondamentale per capire le cause che portano ad alcuni grossolani errori di sicurezza in fase di progettazione di questo protocollo. Vedremo in seguito (capitolo 3) come queste disattenzioni siano diventate oggi delle vere e proprie minacce per la comunicazione in rete.

Lo stack del protocollo TCP/IP (mostrato in figura 2.3) non implementa tutti i 7 livelli OSI ma ne solo 4, in particolare:

- **Il livello host-rete:** nella specifica del modello TCP/IP questo livello, che riassume i livelli 1 e 2 del modello OSI, non viene considerato. L'unica informazione a riguardo è la necessità di essere in grado di spedire pacchetti IP (del livello superiore) lungo il canale.
- **Il livello Internet:** corrispondente al livello 3 del modello OSI, definisce un formato di pacchetto (datagamma) ed un protocollo associato chiamato IP (Internet Protocol). Questo protocollo permette ai pacchetti di essere inviati indipendentemente uno dall'altro verso la destinazione, seguendo anche percorsi e reti differenti. Il protocollo assicura che tutti i datagrammi arrivino a destinazione.
- **Il livello trasporto:** corrispondente al livello 4 del modello OSI, è diviso in due servizi principali, uno orientato alla connessione (Transmission Control Protocol) e l'altro privo di connessione (User Datagram Protocol).

- **Il livello applicazione:** corrispondente al livello 7 del modello OSI, si appoggia direttamente sul livello trasporto (i livelli presentazione e sessione risultasno inutili nella maggior parte delle applicazioni).

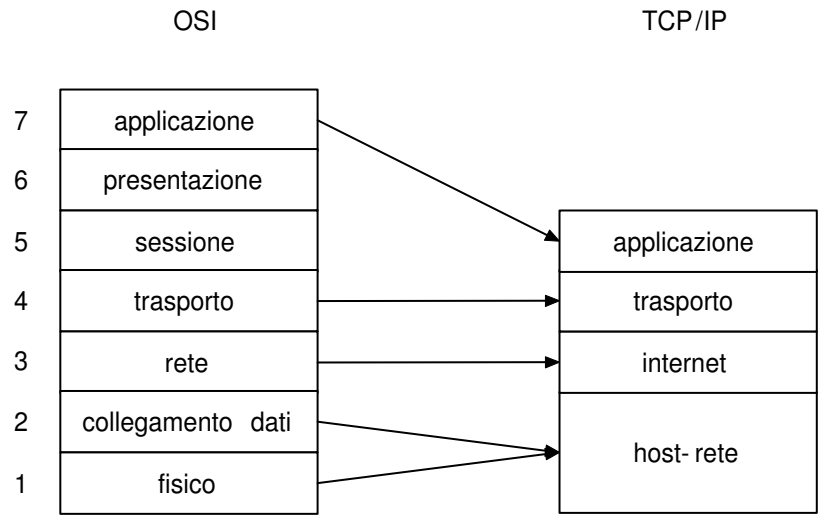

Figura 2.3: Modello di riferimento TCP.

2.1.2 Livello Data-Link: Ethernet

Questo livello è situato sotto al livello Internet e incapsula i pacchetti IP all'interno di frame con formato Ethernet per essere trasmessi sul doppino di rame. Ciascun elemento della rete è individuato da un indirizzo a livello Ethernet. Gli indirizzi Ethernet vengono comunemente chiamati MAC (Medium Access Control) Address e sono rappresentati nella forma aa:bb:cc:dd:ee:ff (12 valori esadecimali separati a due a due dai due punti). I frame Ethernet hanno la struttura presentata in figura 2.4.

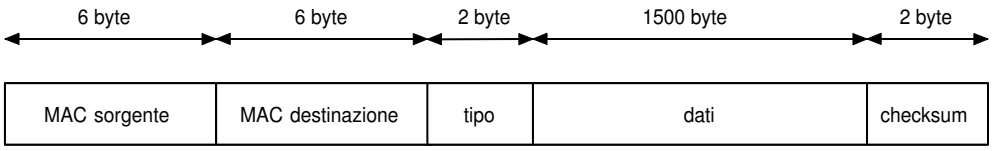

Figura 2.4: Formato di un frame Ethernet.

Sono presenti due campi da 48 bit ciascuno per gli indirizzi Ethernet sorgente e destinazione. Un campo da 16 bit è riservato per determinare il tipo

di payload contenuto nel frame e altri 16 bit in coda sono usati come Frame Check Sequence.

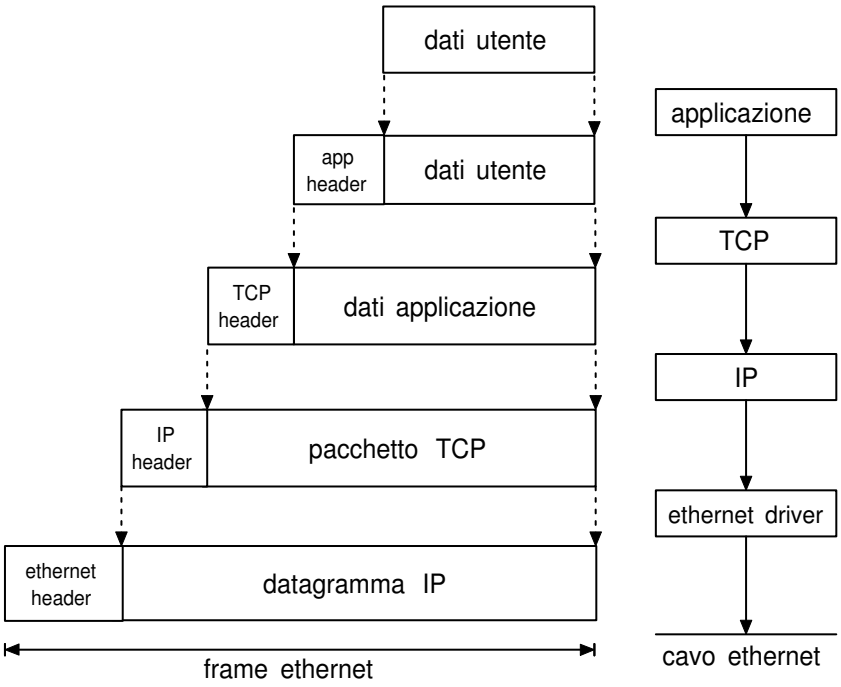

Figura 2.5: Incapsulazione dei dati attraverso lo stack TCP/IP over Ethernet.

La figura 2.5 mostra come una unità dati percorre in discesa lo stack TCP/IP fino ad arrivare al livello fisico. Durante la discesa ogni livello aggiunge la propria intestazione. Nell'ultimo passaggio, il pacchetto viene incapsulato all'interno di un frame Ethernet appena prima la spedizione sul mezzo fisico.

Il problema più grosso che si presenta durante questa discesa e' riuscire ad associare all'indirizzo IP (formato da 32 bit nella versione 4) su un indirizzo Ethernet da 48 bit. In altre parole il kernel del sistema sorgente deve conoscere l'indirizzo Ethernet corrispondente all'IP della destinazione che gli viene passato dal livello superiore. Questo è necessario per poter trasmettere sul cavo un frame con il corretto indirizzo di destinazione. Questa operazione di mappatura è affidata al protocollo ARP.

2.2 Cos'è ARP

L'Address Resolution Protocol è il protocollo che si preoccupa di associare ad ogni indirizzo IP della rete locale un indirizzo nel formato del mezzo di comunicazione. Esistono vari tipi di indirizzi di livello 2 (Ethernet, ARCnet, PROnet, AX.25 NET/ROM). In seguito si faremo riferimento solo alla mappatura su Ethernet.

L'unico scopo di ARP è dunque quello di associare ad un pacchetto IP in uscita il corrispettivo indirizzo Ethernet di 48 bit. Tutto questo deve avvenire automaticamente, nel senso che deve essere trasparente all'utente e alle applicazioni.

Illustriamo il funzionamento di ARP mediante un esempio. Eseguendo un semplice comando quale:

```
$ ftp meltemi
```

(collegamento all'host remoto meltemi per trasferire file) avvengono i seguenti passi:

- L'applicazione (il client ftp) esegue una chiamata alla system call (chiamata di sistema) gethostbyname() per tradurre il nome canonico "meltemi" in un indirizzo IP.
- FTP chiede allo stack TCP di instaurare una connessione verso quell'indirizzo IP.
- Lo stack TCP prepara un datagramma IP con quell'indirizzo come destinazione.
- Se l'IP di destinazione è locale, si può spedire il datagramma direttamente a quell'host. Se invece l'IP è remoto, le policy di routing determineranno l'host verso il quale spedire i datagrammi. In entrambi i casi il datagramma è spedito ad un host della stessa rete locale.
- L'host sorgente deve quindi convertire l'indirizzo IP a 32 bit in un indirizzo Ethernet a 48 bit. E' necessaria una traduzione da un indirizzo logico ad un indirizzo fisico comprensibile dall'hardware. Questa è la funzione del protocollo ARP.

- ARP manda un pacchetto Ethernet chiamato *ARP request* indirizzato a tutti gli host della rete (con indirizzo di destinazione broadcast). L'ARP request contiene l'indirizzo IP dell'host "meltemi" alla quale solo l'host con quell'IP risponderà inviando il proprio indirizzo Ethernet.

- L'host associato all'indirizzo IP riceve il pacchetto al livello ARP e riconosce che il mittente chiede di lui. Quindi prepara una "*ARP reply*" contenente la coppia <IP, MAC address> e la invia al richiedente in modalità unicast.

- L'ARP reply è ricevuta dal mittente originario il quale viene a conoscenza dell'indirizzo fisico desiderato.

- Il datagramma IP può ora essere incapsulato nel corretto frame Ethernet e spedito.

2.3 Formato dei pacchetti ARP

La figura 2.6 mostra il formato di un generico pacchetto ARP usato su link Ethernet.

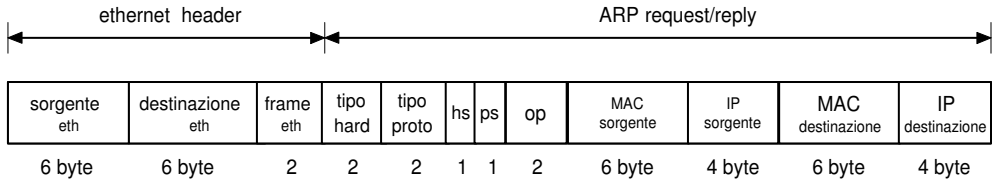

Figura 2.6: Formato di un pacchetto ARP su Ethernet

I primi due campi da 6 byte sono gli indirizzi sorgente e destinazione Ethernet. Segue l'identificatore del payload (frame eth) che specifica che si tratta di un pacchetto ARP. Per indicare che nel payload Ethernet è contenuto un pacchetto ARP, questo campo assume il valore di `0x0806` in "network order", cioe' con il byte più significativo nella posizione più significativa. Segue il pacchetto ARP vero e proprio. Poichè ARP è progettato per risolvere indirizzi anche diversi da IP e da Ethernet, sono presenti 4 campi che specificano il tipo di indirizzi in gioco. I primi due sono i tipi degli indirizzi utilizzati (tipo hard e tipo proto), gli altri rappresentano la dimensione di

tali indirizzi (hs e ps). Nel caso di IP su Ethernet queste dimensioni saranno rispettivamente 6 e 4. Seguono 2 byte per identificare il tipo di pacchetto ARP. I valori possibili sono: request (valore 1) o reply (valore 2) (tralasciamo i valori per le richieste RARP che esulano da questa trattazione). Gli ultimi 4 campi sono l'indirizzo Ethernet e IP sorgente e l'indirizzo Ethernet e IP destinazione.

Per una ARP request tutti i campi sono riempiti tranne l'indirizzo Ethernet destinazione. Questo indirizzo è impostato a broadcast Ethernet (FF:FF:FF:FF:FF:FF) nell'header Ethernet e a zero (00:00:00:00:00:00) nell'header ARP. Quando un host riceve una request, riempie il campo rimasto a zero, scambia gli indirizzi sorgente con quelli di destinazione, cambia il campo "op" a 2 (ARP reply) e spedisce indietro il pacchetto.

2.4 ARP Cache

Per rendere il protocollo più efficiente è necessario mantenere le coppie <IP, MAC address> in una apposita cache ARP su ogni host. Questa cache contiene i risultati delle richieste più recenti per fare in modo che non sia necessario richiedere nuovamente la risoluzione di un indirizzo chiesto precedentemente. La vita media di tali valori è di 20 minuti circa, scaduti i quali la riga contenuta nella tabella viene rimossa. Dopo tale periodo sarà necessario effettuare una nuova richiesta per aggiornare la ARP cache con la coppia <IP, MAC> desiderata. L'RFC [4] specifica che questi valori dovrebbero essere rimossi anche se l'entry è in uso, ma nelle implementazioni odierne la scadenza del timeout viene rimandata nel tempo ad ogni utilizzo.

Per esaminare il contenuto della ARP cache possiamo utilizzare il comando `arp` [1] con l'opzione `-a` .

```
$ arp -a
```

al quale il sistema risponde con la stringa sottostante:

```
meltemi (192.168.0.1) at 00:A0:24:36:00:C3 [ether] on eth0
```

dove i 48 bit di indirizzo Ethernet sono mostrati come valori esadecimali separati dai due punti. Sono inoltre visualizzate informazioni relative al tipo ([ether]) e al nome dell'interfaccia associata (eth0).

[1] Il comando `arp` sarà descritto in modo più approfondito nella sezione 2.5.1

2.5 Esempi di utilizzo

In questa sezione verranno presentati due scenari in cui arp è utilizzato. I log sono stati realizzati con il comando `tcpdump` [7].

2.5.1 Normale utilizzo

Supponiamo che la ARP cache del sistema sia vuota e di volerci collegare ad un sistema remoto. Lo stato della cache ARP e' facilmente controllabile tramite il comando:

```
$ arp -a
```

Il comando `arp` è utile per manipolare o visualizzare il contenuto della ARP cache del proprio sistema. Esso possiede numerose opzioni che permettono di visualizzare tutto il contenuto della cache (opzione -a), cancellare selettivamente una entry (opzione -d), inserire una entry in modo statico (opzione -s). Le entry inserite manualmente sono statiche, nel senso che non subiscono timeout e non possono nemmeno essere aggiornate automaticamente. Come vedremo nel capitolo 3 questa opzione può essere utile a prevenire un comportamento indesiderato del protocollo. Per un elenco completo delle opzioni del comando `arp` si veda la pagine di manuale arp.

A questo punto vogliamo connetterci ad un altro host

```
$ telnet meltemi

Trying 192.168.0.1...
Connected to meltemi.
Escape character is '^]'.
^]
telnet> quit
Connection closed.
```

usando il comando

```
$ tcpdump -v -n -e -i eth0 -f arp
```

possiamo osservare il traffico arp ottenendo il seguente risultato

```
15:07:55.346417 0:8:2:3:5a:f9 ff:ff:ff:ff:ff:ff 0806 42:
                arp who-has 192.168.0.1 tell 192.168.0.7
15:07:55.346795 0:a0:24:36:0:c3 0:8:2:3:5a:f9 0806 60:
                arp reply 192.168.0.1 is-at 0:a0:24:36:0:c3
```

Possiamo vedere come l'host (0:8:2:3:5a:f9) dal quale stiamo effettuando la richiesta di connessione telnet a 192.168.0.1 faccia una richiesta ARP con destinazione broadcast. Come ci si può aspettare, l'host che possiede quell'IP risponde in breve tempo (< 0.4 msec) con il proprio indirizzo Ethernet (0:a0:24:36:0:c3). È da notare che l'ARP reply è mandata solo all'host che ne ha fatto richiesta (0:8:2:3:5a:f9).
A questo punto la connessione ha inizio bilateralmente. L'host destinazione non ha bisogno di fare a sua volta una richiesta ARP verso la sorgente. La sua ARP cache è stata aggiornata all'arrivo della prima richiesta e l'indirizzo sorgente inserito nella coppia <IP, MAC address>.

Nel caso in cui l'host cercato non fosse appartenente alla rete locale, le politiche di routing forniranno l'indirizzo dell'host al quale mandare il datagramma. In questo caso l'host successivo è il gateway e quindi l'ARP request sarà inoltrata chiedendo dell'indirizzo IP del gateway che, essendo un host locale, potrà rispondere con l'adeguata ARP reply.

2.5.2 Richiesta verso un host inesistente

Lo scenario è leggermente differente se la connessione è instaurata verso un host inesistente. Provando a connettersi ad un host inesistente (nel nostro caso 192.168.0.2)

```
$ time telnet 192.168.0.2

Trying 192.168.0.2...
telnet: connect to address 192.168.0.2: No route to host
telnet 192.168.0.2  0.0s user 0.0s system  cpu 2.995 total
```

possiamo notare come lo stack TCP/IP si arrenda dopo circa 3 secondi [2]. Monitorando la rete con `tcpdump` possiamo verificare:

```
$ tcpdump -vv -n -i eth0 -f arp
```

[2] Test effettuato su kernel linux 2.4.18

```
18:06:18.435117 arp who-has 192.168.0.2 tell 192.168.0.7
18:06:19.427804 arp who-has 192.168.0.2 tell 192.168.0.7
18:06:20.427816 arp who-has 192.168.0.2 tell 192.168.0.7
```

le ARP request inviate sono 3 e con un distacco di circa un secondo l'una dall'altra. Questo dipende dal fatto che si suppone che se un host in rete locale non risponde entro 3 secondi, quell'host è sconnesso dalla rete. Infatti 3 secondi sono un tempo enorme se confrontato con il tempo di risposta di una normale ARP reply (nell'esempio del paragrafo 2.5.1 il tempo di risposta era minore di 400 microsecondi).

2.6 Gratuitous ARP

Un'altra peculiarità del protocollo ARP è la presenza di particolari messaggi di richiesta chiamati *gratuitous ARP*. Questi pacchetti sono mandati da un host che chiede di risolvere il proprio indirizzo. Di solito sono usati per configurare la scheda di rete al boot. In termini di formato del pacchetto, questi messaggi hanno gli indirizzi sorgente e destinazione identici.
Questo tipo di pacchetti svolge due principali funzioni:

- Permette ad un host di controllare la presenza di altre macchine configurate con il suo stesso IP. Infatti se alla richiesta segue una risposta, si viene a conoscenza della presenza di un'altra macchina cui e' stato assegnato l'IP per cui e' stato inviato il messaggio di richiesta. Nel caso venga rilevato un conflitto, un messaggio è inviato a video alla console dell'amministratore di rete per segnalare che la rete è malconfigurata.

- Permette ad un host di annunciare in broadcast che il proprio indirizzo Ethernet è cambiato. Infatti questo tipo di pacchetto permette di aggiornare le entry delle ARP cache degli host nei quali era presente la coppia <IP, vecchio MAC address>. L'RFC [4] specifica che se un host riceve una ARP request (mandata in broadcast) da un indirizzo che è già presente nella sua ARP cache, la relativa entry deve essere aggiornata. Questo accade anche se il tipo di messaggio ARP è una reply (come vedremo più in dettaglio nella sezione 3.2).

Capitolo 3

Attacchi "Man In The Middle"

In questo capitolo vengono illustrati gli attacchi di tipo "man in the middle" basati su ARP poisoning, differenziando tra le varie tipologie: full-duplex, half-duplex, transparent o proxy attack. In particolare la tecnica ARP poisoning e' documentata in ogni sua parte. Se ne descrive il funzionamento teorico, fornendo anche esempi pratici e tool di attacco. Vengono forniti numerosi schemi che illustrano come questo attacco può assumere diverse topologie a seconda che si tratti di attacco da locale a locale, da locale a remoto oppure half duplex remoto.
Il capitolo si conclude con l'analisi delle conseguenze che un attacco man in the middle può avere. Gli attacchi che possono derivare illustrati nel capitolo sono lo sniffing su reti connesse tramite switch, il dirottamento (hijaking) di connessioni attive, la modifica (filtering) del contenuto e l'inserimento (injecting) di dati e/o codice all'interno di uno flusso di dati. In fine è spiegato come il man in the middle permetta di compromettere connessioni protette crittograficamente come quella del protocollo SSH versione 1.

3.1 Cos'è il man in the middle

L'attacco che va sotto il nome di *man-in-the-middle* consiste nel dirottare il traffico generato durante la comunicazione tra due due host verso un terzo host attaccante così da far credere a ciascuno gli end-point della comunicazione di parlare con il legittimo interlocutore, quando invece parlano con l'attaccante che impersona i due host a turno. Come mostrato in figura 3.1,

l'attaccante riceve tutto il traffico generato dagli host 1 e 2 e si preoccupa di inoltrare correttamente il traffico verso l'effettiva destinazione dei pacchetti ricevuti.

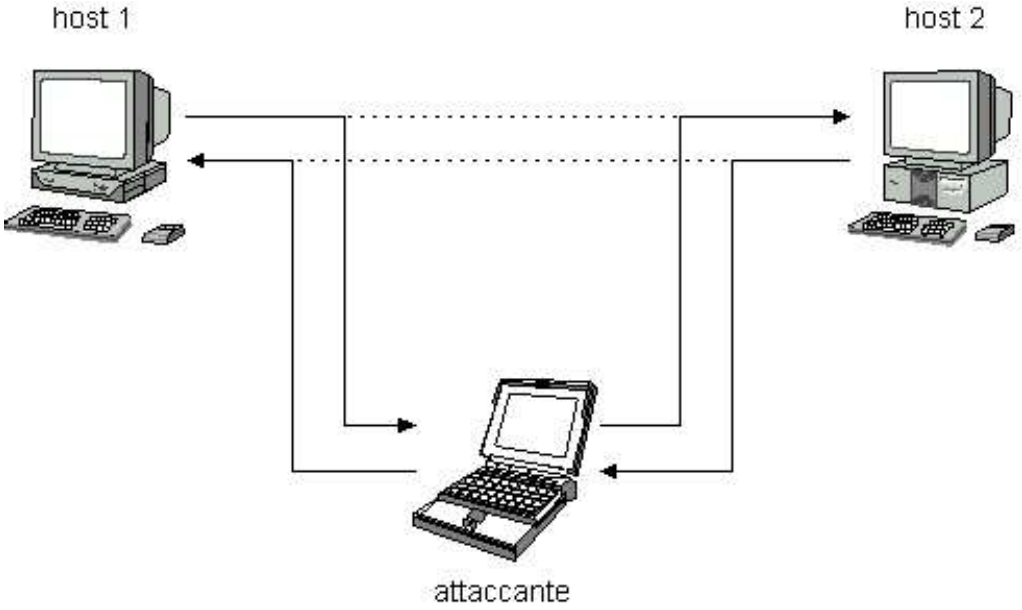

Figura 3.1: Connessione tra due host intercettata secondo lo schema "man in the middle"

A seconda della capacità di riuscire a dirottare solo uno o entrambi i flussi della connessione l'attacco verrà chiamato "mitm half duplex" o "mitm full duplex". Questa distinzione è importante perché alcuni degli attacchi presentati nella sezione 3.3 possono essere portati a termine solo avendo il controllo di entrambi i flussi della comunicazione.

Come mostra la figura 3.2 l'attacco mitm Full Duplex riesce a dirottare completamente entrambi i flussi della comunicazione verso la macchina attaccante. In figura sono mostrate le direzioni della comunicazione logica (quella che si aspettano le due vittime) con una linea tratteggiata e la comunicazione fisica (quella come in realtà avviene) con tratto continuo.

Come vedremo in seguito (sezione 3.2.2) questo attacco può essere portato a termine solo se entrambe le vittime sono raggiungibili dall'attaccante, cioè se si trovano nello stesso dominio di broadcast (nel caso di un attacco tramite ARP spoofing) e sono entrambe vulnerabili.

La figura 3.3 descrive lo scenario nel quale l'attaccante sia riuscito a dirottare solo un flusso della comunicazione. In questo caso il traffico dall'host 1

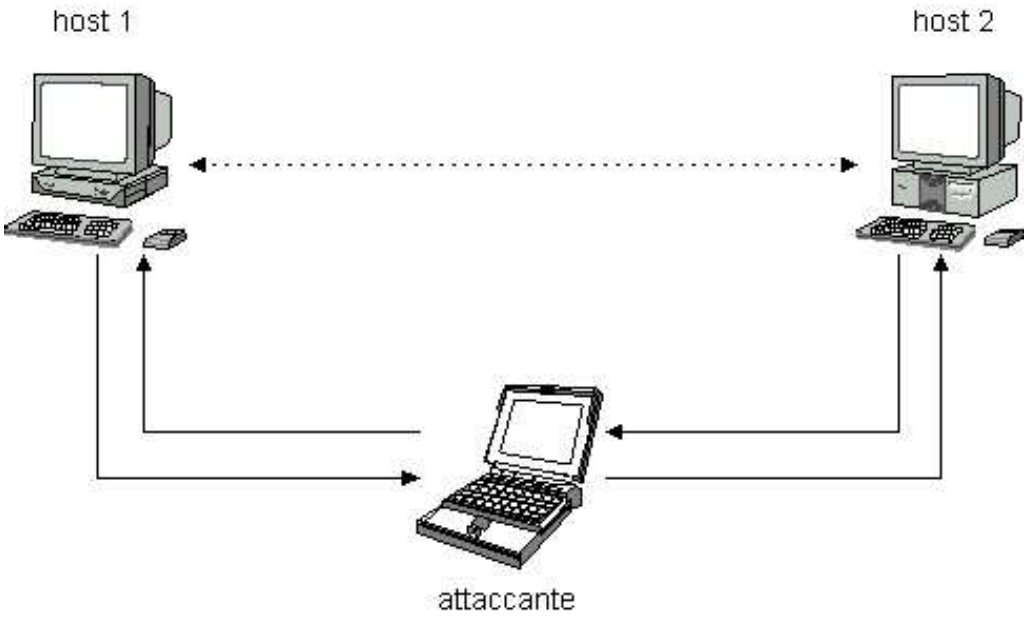

Figura 3.2: Attacco man in the middle Full Duplex

al 2 viene intercettato dall'attaccante, ma il traffico opposto giunge intatto a destinazione. È subito chiaro come questo attacco sia meno pericoloso dei quello Full Duplex, ma è necessario prestare attenzione poiché anche in questa situazione le azioni che possono esser compiute dall'attaccante sono tante.

In seguito (sezione 3.3) vedremo quali tipologie di attacchi possono comunque essere portate a termine anche in presenza di mitm half duplex. Infatti è comunque possibile controllare entrambi i flussi della comunicazione anche se inizialmente si è in regime di Half Duplex. L'unica condizione necessaria è quella di essere in grado di intercettare il primo pacchetto scambiato durante l'handshake della connessione. Se infatti, l'attaccante riesce ad intercettare il lato da cui parte il SYN del protocollo TCP, è possibile portare a termine un attacco di tipo Proxy (sezione 3.1.2) e passare ad un attacco Full Duplex.

3.1.1 Transparent attack

I pacchetti diretti verso la seconda vittima (che abbiano cioè come indirizzo IP di destinazione quello reale dell'altro host) intercettati possono essere inoltrati esattamente come sono stati ricevuti. Il campo Time To Live

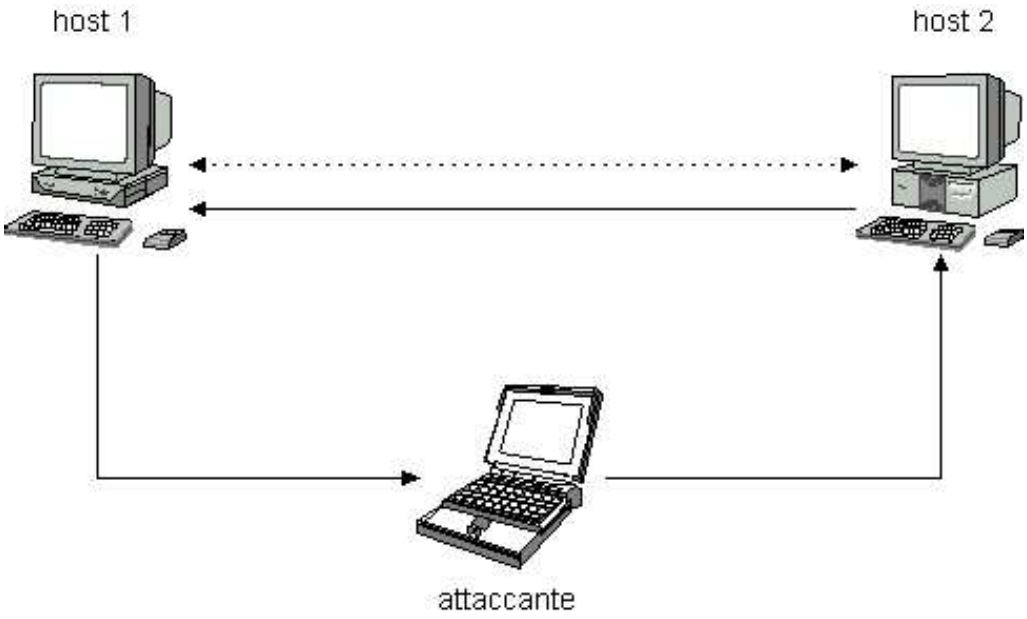

Figura 3.3: Attacco man in the middle Half Duplex

dell'header IP non sarà modificato e questo non consentirà di verificare a posteriori che il pacchetto ha attraversato un nodo in più. La possibilità di lasciare inalterato il TTL dipende dal tool di attacco impiegato.

Se si utilizzano programmi quali `dsniff` [8] o `arpspoof` è necessario abilitare l'opzione di IP forwarding nel kernel. Questo fa in modo che sia il kernel a preoccuparsi di inoltrare i pacchetti destinati ad altri host. Così facendo però il kernel diminuisce di un'unità il campo TTL dell'header IP, segnalando la presenza di un nodo intermedio tra i due host vittima. Altri software di attacco quali ad esempio `ettercap` , [9] disabilitano l'IP forwarding ed effettuano manualmente il forwarding del pacchetto che resta così intatto non consentendo di individuare il nodo intermedio attraversato. È possibile rilevare i nodi intermedi attraverso il comando `tracepath` .

3.1.2 Proxy attack

Se si tratta di connessioni TCP è possibile che l'attaccante accetti la connessione sulla propria macchina e ne crei un'altra con il reale destinatario. Se l'indirizzo destinazione è quello dell'attaccante, esso agirà da proxy vero e proprio per la connessione. Saranno cioè instaurate due connessioni: una dalla prima vittima al mitm e una dal mitm alla seconda vittima.

Come accennato in precedenza esiste un escamotage che permette ad un attaccante di trasformare un mitm half duplex in un proxy attack full duplex. Infatti è possibile intercettare il SYN dell'header TCP e instaurare con l'indirizzo IP destinazione una connessione separata. Rispondendo adeguatamente al SYN con un SYN+ACK la vittima non si accorgerà della differenza. Nel frattempo il lato della comunicazione che sarebbe rimasto scoperto è stato soppiantato da una connessione vera e propria dalla macchina attaccante.

3.1.3 Man in the middle a vari livelli

Una attacco man in the middle deve essere portato a termine sfruttando il livello ISO/OSI inferiore a quello verso il quale l'attacco è diretto. Poiché l'architettura a livelli è pensata appositamente per essere trasparente ai peer remoti, ogni livello sarà convinto di comunicare con il proprio corrispondente remoto senza intermediari. È proprio questa caratteristica che viene sfruttata per realizzare il man in the middle.
Se la comunicazione è dirottata ad un livello sottostante, il peer di livello superiore non si accorgerà della differenza. Ogni livello utilizza un proprio metodo per indirizzare i dati. A livello applicazione l'indirizzamento avviene tramite nomi simbolici (es: www.kernel.org), a livello trasporto tramite porte (es TCP porta 80), a livello network tramite indirizzi IP (es 204.152.189.116), a livello data-link tramite MAC address (es 00:A0:24:36:00:C3) e a livello fisico è il cavo che decide dove i pacchetti arriveranno.
Se volessimo attaccare il livello applicazione, basterebbe intercettare la risoluzione del nome simbolico in indirizzo IP. Intercettando la richiesta al Domain Name System (DNS) e' possibile rispondere con un indirizzo IP differente e dirottare su di esso il traffico.
Per attaccare il livello IP, intercettiamo le risoluzioni ARP (che vedremo tra breve) e dirottiamo verso un MAC address differente.
Per attaccare il livello data-link, l'unica soluzione è quella di dirottare fisicamente i cavi di connessione e a questo non c'è rimedio, se un attaccante ha accesso fisico alle macchine e non è possibile fermarlo.

3.2 Attacco MITM tramite ARP poisoning

Abbiamo visto cos'è un attacco di tipo *man in the middle*, ora vedremo come è possibile sfruttare le debolezze di progettazione del protocollo ARP. Infatti questo protocollo fu disegnato per essere il più possibile efficiente (negli anni 70 i link avevano una banda ridottissima e ogni byte in meno scambiato in rete era un guadagno notevole), ma non si tenne in considerazione il lato sicurezza delle scelte effettuate.
Vedremo come sia possibile spedire messaggi ARP fasulli e modificare il contenuto delle cache ARP di altri host appartenenti alla stessa rete locale.

3.2.1 ARP reply non sollecitate

Grazie all'utilizzo dei gratuitous ARP (sezione 2.6) abbiamo visto come sia possibile aggiornare la ARP cache di host appartenenti alla rete locale. Questa funzionalità fu progettata principalmente per ragioni di efficienza. Se un host avesse cambiato la scheda di rete, avrebbe potuto comunicarlo a tutti i suoi vicini senza attendere che a turno ciascuno chiedesse un aggiornamento.
Purtroppo non si è tenuto conto delle problematiche di sicurezza che un simile meccanismo comporta. Infatti se un host malintezionato crea un pacchetto ARP contenente informazioni fasulle, le ARP cache dei sui vicini saranno aggiornate senza alcun tipo di controllo. Questo può essere molto pericoloso poiché mette un qualsiasi host della rete locale in condizione di diventare il man in the middle di una qualsiasi comunicazione.
L'attaccante quindi non dovrà fare altro che creare finte ARP reply non sollecitate nelle quali annuncia che l'host vittima ha cambiato MAC address. Se il nuovo MAC pubblicizzato corrisponde a quello dell'attaccante, esso produrrà l'effetto di dirottare la connessione in modo desiderato. Infatti l'host la cui ARP cache e' stata così "avvelenata" (dall'inglese *ARP poisoning*) da quel momento in poi manderà in rete frame con indirizzo di destinazione dell'attaccante pur essendo convinto di comunicare con un altro host (quello corrispondente all'IP usato).

Poiché la cache è periodicamente cancellata dal kernel, per poter portare a termine un attacco ARP poisoning a lungo termine, l'attaccante dovrà continuamente rinfrescare il valore delle entry mandando le relative ARP

reply ad intervalli regolari minori del timeout. In presenza di Intrusion Detection System (IDS), un simile comportamento da parte di un host sarebbe segnalato come anomalo.

Sempre per motivi efficienza si è pensato di memorizzare nella ARP cache anche l'indirizzo sorgente di una richiesta ARP. Infatti dai campi dell'header ARP è possibile risalire alla coppia <IP, MAC address> del richiedente. Questo meccanismo è stato implementato con l'idea che un host che effettua una richiesta ad un altro host, a breve l'host contattato avrà bisogno di conoscere il suo MAC address per poter rispondere ai pacchetti inviati. Per evitare di dover fare una nuova ARP request, i valori sono già immessi in cache.
Questo porta ad un altro buco di sicurezza. Se un host effettua una ARP request con indirizzo sorgente fasullo, l'host destinazione immetterà nella sua cache la coppia fasulla. Il risultato è il medesimo: l'ARP cache è avvelenata.

Alcuni sistemi operativi non inseriscono nella propria cache i valori di ARP reply se l'IP della coppia <IP, MAC address> non è già presente in cache. In altre parole si può solo aggiornare una entry, ma non crearne una nuova. Questo potrebbe far pensare ad una soluzione al problema, ma non è così. Se infatti l'attaccante manda un ICMP echo request modificato, la vittima è costretta a risolvere l'IP sorgente (spoofato dall'attaccante) e inserire una entry nella propria ARP cache. A questo punto è possibile aggiornare quelle entry con una ARP reply forgiata ad arte.

3.2.2 Esempi di attacco

Esaminiamo uno scenario con tre host: due vittime e un attaccante. Questi sono i rispettivi indirizzi IP e MAC:

HOST	**IP**	**MAC Address**
HOST 1	192.168.0.1	01:01:01:01:01:01
HOST 2	192.168.0.2	02:02:02:02:02:02
ATTACCANTE	192.168.0.3	03:03:03:03:03:03

Supponendo che un host sia interessato a dirottare il traffico tra host 1 e host2, esso manderà due false ARP reply dicendo ai due host vittime che

il MAC address del proprio interlocutore è cambiato in 03:03:03:03:03:03 (indirizzo di rete dell'attaccante). Vediamo in dettaglio:

Inizialmente l'ARP cache dell'host 1 sarà simile a questa:

[host1] `$ arp -a`

```
host2 (192.168.0.2) at 02:02:02:02:02:02 [ether] on eth0
```

mentre quella dell'HOST 2 :

[host2] `$ arp -a`

```
host1 (192.168.0.1) at 01:01:01:01:01:01 [ether] on eth0
```

L'attaccante manderà delle ARP reply a:

HOST 1 dicendo che 192.168.0.2 ha come MAC 03:03:03:03:03:03
HOST 2 dicendo che 192.168.0.1 ha come MAC 03:03:03:03:03:03

Le arp cache si presentano ora così:

[host1] `$ arp -a`

```
host2 (192.168.0.2) at 03:03:03:03:03:03 [ether] on eth0
```

[host2] `$ arp -a`

```
host1 (192.168.0.1) at 03:03:03:03:03:03 [ether] on eth0
```

A questo punto i pacchetti da e verso host 1 e host 2 saranno mandati in un frame Ethernet con indirizzo di destinazione 03:03:03:03:03:03 (che è quello dell'attaccante).
Per portare a termine l'attacco è necessario che i pacchetti siano inoltrati correttamente verso l'effettiva destinazione. Quindi l'attaccante non deve far altro che inoltrare:

i pacchetti indirizzati a 192.168.0.1 a HOST 1 (01:01:01:01:01:01)
i pacchetti indirizzati a 192.168.0.2 a HOST 2 (02:02:02:02:02:02)

per fare ciò gli basterà modificare il MAC address destinazione e inoltrare di nuovo il pacchetto.

3.2.2.1 Attacco locale-locale

Un attacco ARP poisoning totalmente in locale consiste nel dirottare la comunicazione tra due host che appartengono alla stessa rete dell'attaccante. La figura 3.4 mostra come avviene l'attacco. L'attaccante avvelena periodicamente le ARP cache delle due vittime con finte ARP reply (linee tratteggiate). I due host avvelenati cominceranno a spedire i pacchetti indirizzati verso l'attaccante (linee tratto-punto). A questo punto l'attaccante si preoccupa di inoltrare correttamente i pacchetti da esso ricevuti e la comunicazione può avere luogo.

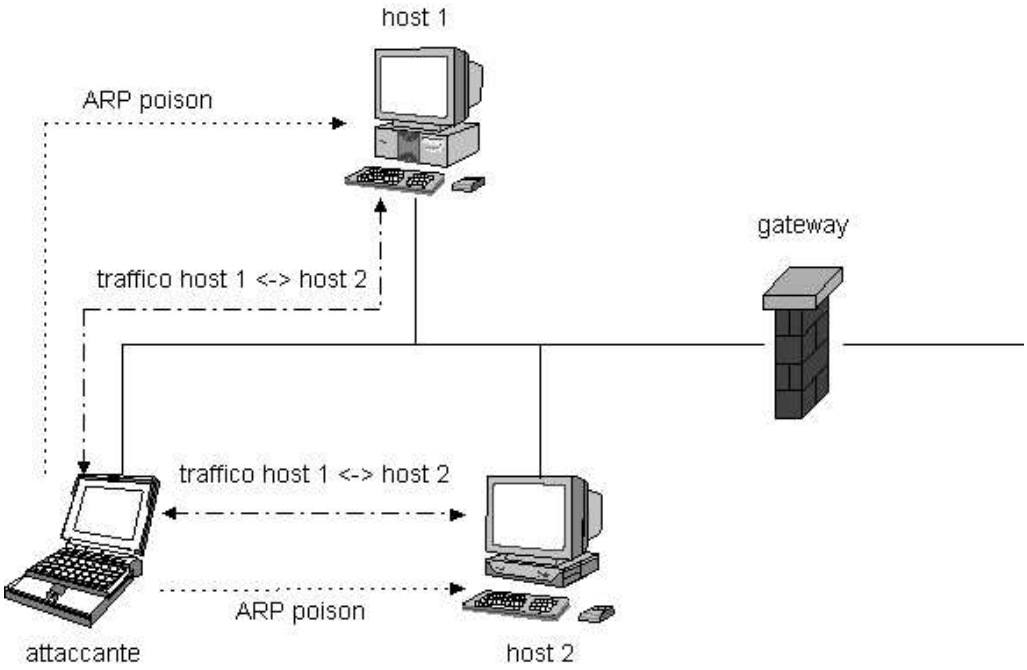

Figura 3.4: Attacco man in the middle da locale a locale

3.2.2.2 Attacco locale-remoto

L'attacco che intercetta una connessione da locale a remoto potrebbe sembrare molto più difficile da attuare. Infatti, come noto, le ARP reply non vengono inoltrate all'esterno del dominio di broadcast. Questo impedisce di avvelenare l'host remoto.
Analizzando come avviene l'effettiva comunicazione si capisce come questo attacco possa essere portato a termine. Un host che vuole instaurare una connessione verso l'esterno riceve, in base alle sue politiche di routing, l'in-

formazione del prossimo host da contattare. In questo caso sarà il gateway ad essere il suo next hop. Essendo il gateway un host locale, esso può subire l'avvelenamento esattamente come una qualsiasi altra macchina della rete.

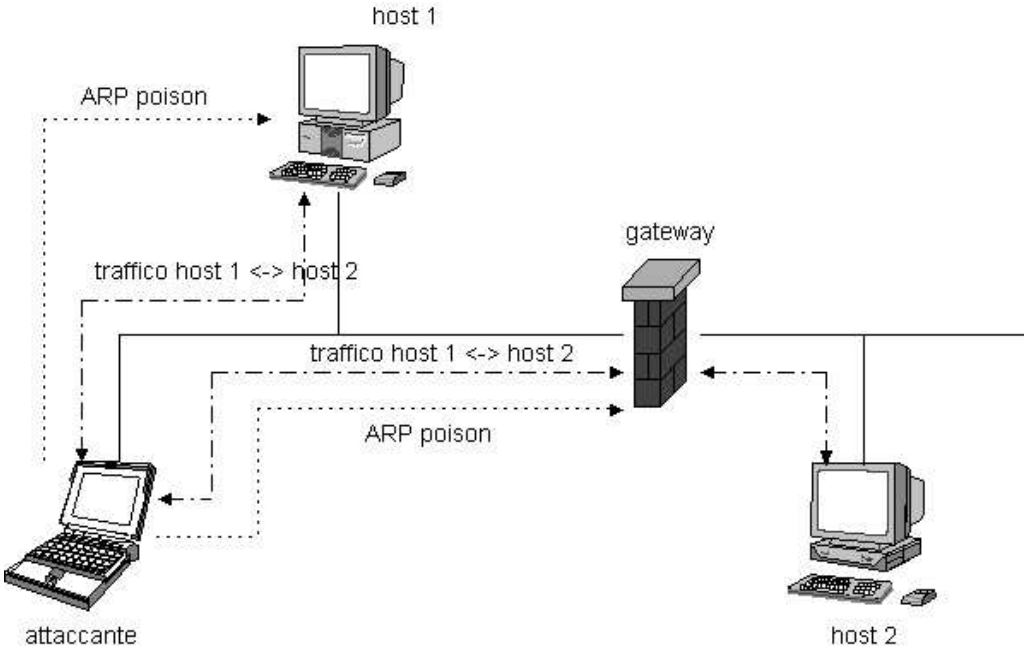

Figura 3.5: Attacco man in the middle da locale a remoto

L'attaccante quindi avvelena l'host vittima modificando il MAC address del gateway con il proprio. E avvelena il gateway modificando quello della vittima. A questo punto entrambi inoltreranno i loro pacchetti all'host attaccante (vedi figura 3.5).

3.2.2.3 Attacco half duplex

È possibile però che una delle due vittime risulti immune all'attacco di ARP poison. Le cause potrebbero essere ad esempio la presenza di ARP entry statiche (e quindi non aggiornabili) nella ARP cache. In questo caso è possibile intercettare solo metà del traffico.

Come mostrato in figura 3.6 spesso sono i client che mantengono una entry statica per il gateway e non il contrario. Altrimenti il gateway dovrebbe conoscere preventivamente tutti gli indirizzi MAC degli host della rete, che potrebbero essere centinaia o essere associati a IP differenti grazie a meccanismi di DHCP.

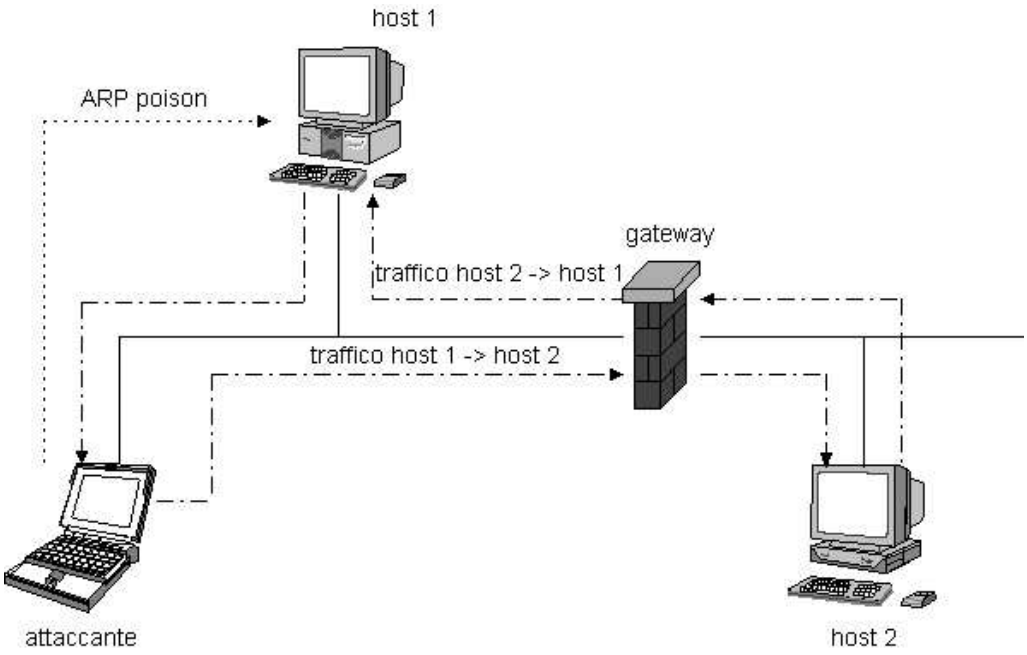

Figura 3.6: Attacco man in the middle remoto half duplex

Dalla figura risulta come il traffico sia deviato al rientro nella rete. Il gateway è convinto di spedire i pacchetti di ritorno da internet all'host che ne aveva fatto richiesta. Invece è l'attaccante a riceverli e a inoltrarli al corretto richiedente.

3.3 Conseguenze di un attacco

Il risultato più importante ottenibile con la tecnica del man in the middle è la capacità da parte di un attaccante di modificare il flusso di dati della connessione. Vediamo meglio in dettaglio che tipologie di attacchi è possibile portare a termine dopo essere riusciti ad occupare la posizione di man in the middle.

3.3.1 Sniffing su reti connesse mediante HUB

Lo sniffing (dall'inglese to sniff, *annusare*) è la capacità da parte di un host di catturare i pacchetti non destinati ad esso.
Questo attacco è facilmente attuabile su rete locale Ethernet connessa tramite HUB. Un HUB (o concentratore) lavora a livello 1 del modello di

riferimento OSI, quello fisico. A questo livello l'HUB si limita a copiare su tutte le porte di uscita quello che riceve in entrata da una sua porta. In questo modo tutti gli host della rete ricevono tutto il traffico dell'intera rete (come mostrato in figura 3.7).

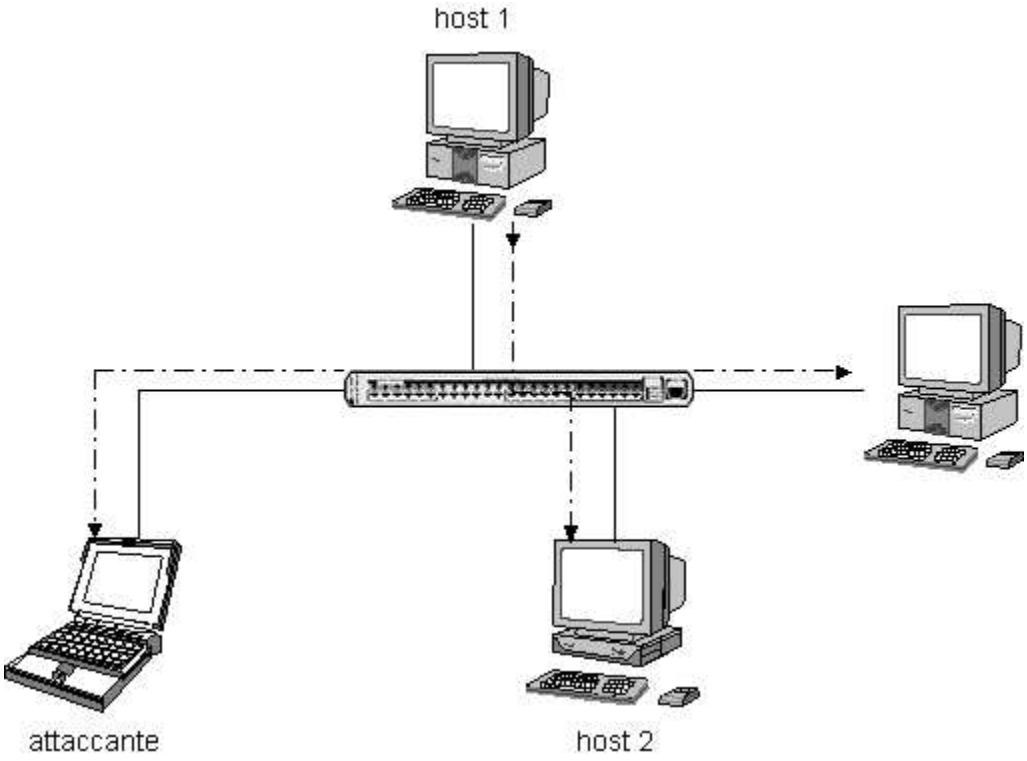

Figura 3.7: Rete Ethernet connessa tramite HUB (layer 1)

Sono i singoli host che discriminano se il traffico è indirizzato a loro a seconda dell'indirizzo MAC destinazione. E' sufficiente però configurare la scheda di rete in modalità `promisc` per fare in modo che il kernel riceva anche i pacchetti con MAC address differente da quello della scheda di rete (Network Interface Card).

3.3.2 Sniffing su reti connesse tramite switch

Poiché le reti connesse tramite HUB soffrono di un eccessivo numero di collisioni a livello fisico, sono stati creati gli switch. Essi operano a livello 2 dello stack ISO/OSI e hanno il compito di suddividere i domini di collisione del mezzo fisico.
Su una rete connessa tramite switch ogni frame attraversa solo i cavi sorgente e destinazione (figura 3.8). Per poter conoscere quale sia la porta su cui

inoltrare i frame, lo switch deve conservare una tabella (hash) nella quale sono memorizzate tutte le coppie <numero di porta, MAC address>. Questa tabella viene creata in modo adattivo. In principio, quando la tabella è vuota, si utilizza l'algoritmo di flooding su tutte le porte, ma con il tempo la tabella si popola secondo l'algoritmo di *backward learning*. Osservando l'indirizzo di provenienza dei frame gli switch possono stabilire a quale porta si legato un determinato MAC address. In questo modo le tabelle vengono riempite in breve tempo.

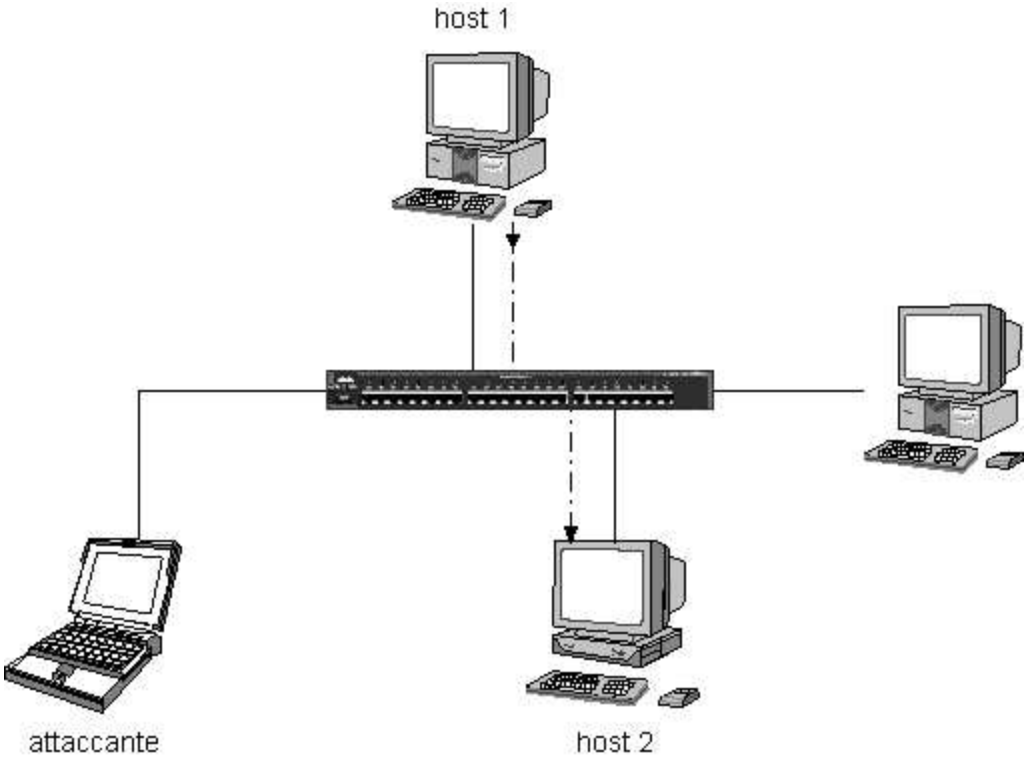

Figura 3.8: Rete Ethernet connessa tramite switch (layer 2)

In questo scenario l'attaccante non può sniffare il traffico dalla vittima al server. È qui che entra in gioco il man in the middle attraverso l'uso di ARP poison.

Come descritto nella sezione 3.2 è possibile modificare, tramite l'invio di ARP reply fasulle, la ARP cache di uno o più host della rete. Da quel momento in poi i frame spediti sul mezzo fisico avranno come MAC address di destinazione quello dell'attaccante. In questo caso switch non può fare nulla poiché il suo compito è quello di controllare l'indirizzo di destinazione del frame e di spedirlo sulla corretta porta. Ciò non può prevenire la de-

viazione dei pacchetti, infatti i frame spediti dalle vittime contengono come reale indirizzo di destinazione quello dell'attaccante.
Si noti che l'ARP poisoning non ha alcun effetto sulle tabelle interne dello switch, essendo mirato a modificare le ARP cache degli host. Quindi a meno di switch che operino a livello 3 che ispezionano anche il contenuto dell'header ARP, un semplice switch non è sufficiente a fermare l'attacco ARP poison.

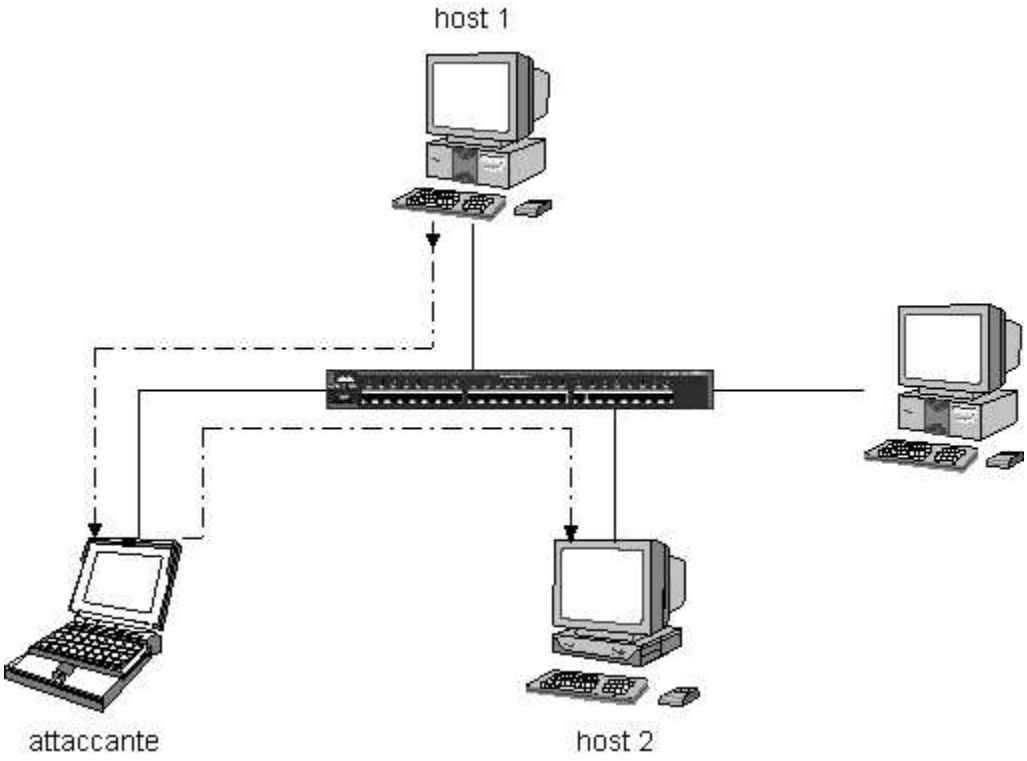

Figura 3.9: Attacco ARP poisoning su rete connessa tramite switch

Alcuni switch implementano una funzionalità chiamata *Port Security*. Essa consiste nel legare definitivamente un MAC address ad una porta e rifiutare qualsiasi altro MAC address proveniente da essa. Questo permette di impedire la modifica del reale MAC address dell'attaccante, ma non gli impedisce di utilizzare un indirizzo lecito per portare a termine l'attacco. E' quindi inefficace per proteggersi dall'ARP poisoning.

E' evidente quindi come lo sniffing sia immediato se l'attaccante è riuscito a guadagnare il "middle". Infatti i pacchetti della connessione dirottata transitano attraverso l'host attaccante; sarà sufficiente una socket di tipo PF_PACKET per poterli ricevere a livello applicazione.

3.3.2.1 Sniffing su reti wireless

Le reti wireless forniscono accesso alla rete ai terminali mobili. L'accesso ad una determinata rete è discriminato usando alcuni algoritmi crittografici: dal più vecchio e meno sicuro WEP (Wired Equivalent Privacy [10]) al più nuovo ma fuori standard EAP (Extensible Authentication Protocol [11]).

Una volta connessi ad un Access Point (conoscendo le giuste chiavi di autenticazione), una rete wireless si presenta come una normale LAN connessa tramite switch. Ovvero un host riceve solo il traffico ad esso destinato.
Esistono alcune schede wireless (come le Cisco, Prism2 o Orinoco) che permettono di configurare la scheda in *monitor mode* e di intercettare anche pacchetti spediti da altri host. Questa modalità però permette di leggere i pacchetti in formato raw (non decifrato) e non è utile per lo sniffing vero e proprio. La modalità monitor è utilizzata di solito per identificare IP, NETMASK e chiavi WEP deboli intercettando un numero sufficiente di pacchetti. Essendo l'equivalente di una rete connessa tramite switch, valgono le stesse ipotesi fatte nella sezione precedente (3.3.2).

3.3.3 Hijacking

L'hijacking (dall'inglese hijack, *dirottare*) è un attacco che permette di intercettare una connessione già stabilita tra due host e di prenderne possesso. Questa tecnica non è affatto facile da portare a termine poiché per "impadronirsi" di una connessione TCP bisogna conoscere i giusti numeri di sequenza e di acknowledgement dell'header TCP. Questo è molto difficile se la connessione è remota e di deve ricorrere a tecniche di tipo blind (alla cieca). Ma se riflettiamo sul fatto che i pacchetti della connessione transitano attraverso l'host dell'attaccante, ci rendiamo subito conto di quanto gli sia facile leggerli e conoscere i giusti numeri di sequenza.

In questo scenario è estremamente facile chiudere la connessione da un lato e impossessarci dell'altro end point. Basterà utilizzare i giusti numeri di sequenza per mandare un RST da un lato e sincronizzare la propria connessione con l'altro end-point.

3.3.4 Injecting

Il man-in-the-middle aumenta la potenza rispetto ad un Hijack normale. Infatti è possibile aggiungere alla connessione alcuni pacchetti, pur mantenendo la connessione perfettamente sincronizzata ed attiva tra le vittime.
Per fare ciò l'attaccante dovrà modificare i numeri di sequenza dei pacchetti TCP ricevuti prima di inoltrarli verso l'altro host. Memorizzando quanti byte sono stati inviati da un lato e dall'altro, con semplici addizioni e sottrazioni si mantiene la connessione sincronizzata.
Questo tipo di attacco va sotto il nome di Injecting (dall'inglese Inject, *iniettare*). Può essere portato a termine solo in presenza di mitm Full Duplex poiché è necessario sincronizzare entrambi i lati della connessione.

Nel caso l'attacco sia di tipo proxy (vedi sezione 3.1.2) questi problemi non sono rilevanti poiché vengono instaurate due connessioni separate: client-attaccante e attaccante-server, quindi i numeri di sequenza sono gestiti direttamente dal kernel.

3.3.4.1 Command injection

Questo tipo di attacco di solito è sferrato contro quei sistemi di autenticazione non riproducibili nel tempo. Sistemi cioè che cambiano la chiave ad ogni accesso (One time password, token RSA , ecc). Solitamente questo tipo di autenticazione si applica a protocolli insicuri quali rlogin e telnet.
Questi protocolli viaggiano in chiaro (non criptati) sul cavo e sono quindi passibili di password sniffing. Si utilizzano quindi autenticazioni one time per impedire all'attaccante di poter riutilizzare la password catturata.

Se però l'attaccante ha "guadagnato il middle", può impadronirsi della sessione fin tanto che è attiva e sfruttare la TCP injection per eseguire comandi sul lato server. Ovviamente l'output di tali comandi sarà poi filtrato verso il client che non si accorgerà di nulla.

3.3.4.2 Malicious code injection

Come visto nel paragrafo precedente l'attaccante può inserire comandi in una sessione aperta di telnet. Se invece di una sessione telnet viene intercettata una connessione HTTP e vengono aggiunte parti di codice (java,

javascript, VBscript ecc) ad una pagina web che sta per essere interpretata dal browser, si parla di "malicious code injection".
Questa tecnica non si limita alle sole pagine web ma può diventare ancora più pericolosa se la connessione intercettata è un FTP e si aggiunge parte di codice ad un eseguibile o ad un sorgente scaricato in quel momento. In questo modo l'attaccante è in grado di aggiungere codice maligno (e.g. virus) all'interno di programmi che si stanno scaricando da siti che sono ritenuti fidati.

3.3.5 Content Filtering

Abbiamo visto come sia possibile modificare i numeri di sequenza di una connessione TCP (sezione 3.3.4). Allo stesso modo è possibile modificarne solo il payload. Ovviamente si dovrà fare attenzione a ricalcolare il corretto checksum dell'header TCP. L'attaccante è quindi in grado di creare filtri che a run-time modificano il contenuto dei dati scaricati.

Per come è strutturato il protocollo un attaccante che ha il controllo su un solo flusso della connessione sarà in grado di modificare il contenuto dei pacchetti senza però cambiare la lunghezza degli stessi (pena la desincronizzazione dei numeri di sequenza, dato che non sarà in grado di modificare gli ACK nel senso opposto). Se invece si è in ambito di mitm Full Duplex valgono le considerazioni fatte per l'injecting.

3.3.6 Sniffing del protocollo SSH v1

SSH e' un protocollo di comunicazione tra host che utilizza la cifratura per proteggere i dati scambiati. È stato pensato per rimpiazzare telnet. Quando una connessione SSH v1 comincia, il server invia la sua chiave pubblica al client.
Poiché l'attaccante è in grado di cambiare il contenuto dei pacchetti on-the-fly, può sostituire la chiave pubblica del server con una generata in precedenza. In questo modo il client riceve una chiave pubblica di cui l'attaccante ha la rispettiva chiave privata.
Quando il client manda il pacchetto contenente la chiave di sessione cifrata con la chiave pubblica (fasulla) del server, l'attaccante sarà in grado di decifrarla e di salvare la chiave 3DES di sessione. A questo punto il server

riceve la chiave di sessione cifrata con la chiave pubblica corretta del server (precedentemente salvata).
La connessione è instaurata normalmente, ma l'attaccante conosce la chiave di sessione. A questo punto è possibile decifrare tutto il traffico semplicemente usando la chiave 3DES precedentemente catturata. La connessione rimane attiva anche quando il mitm si interrompe poiché non è un regime di proxy per la connessione, ma solo una decifratura dei dati con la chiave di sessione.
Sono comunque possibili soluzioni di tipo proxy ottenibili con tutte le altre tecniche di m-i-t-m. In questo caso si avranno due scambi di chiavi separati, uno fra client e attaccante, l'altro fra attaccante e server.

La teoria alla base di questo attacco è utilizzabile anche contro altri canali cifrati come ad esempio SSL, dove le chiavi sono scambiate all'inizio della comunicazione.

E' possibile sostituire i parametri che il server e il client si scambiano per modificare il comportamento di entrambi.
Ad esempio è possibile far sì che una connessione che richiede SSH2 sia stabilita in SSH1. Se il client non ha nella propria configurazione il parametro "Strict", vedendo la risposta del server, decide quale versione di SSH utilizzare.
Risposte del server possibili:

- SSH-1.99 – il server supporta ssh1 e ssh2
- SSH-1.51 – il server supporta solo ssh1

quindi per forzare un client ad usare SSH versione 1, è possibile creare un filtro che sostituisce il banner SSH-1.99 con SSH-1.51. In questo modo la connessione sarà instaurata usando il protocollo 1 facilmente vulnerabile al m-i-t-m (sezione 3.3.6).

Questo può portare anche ad aggirare il problema della coppia <server, chiave pubblica> salvata nel file known_host dal lato client. Se il server supporta entrambe le versioni del protocollo, il client sceglierà molto probabilmente la SSH v2 ad ogni connessione, salvando la coppia <server, chiave> relativa a questa versione del protocollo. Supponiamo a questo punto che un client si connetta ad un server a cui si era già connesso in precedenza

e sia vittima di un attacco di questo tipo. Il client non troverà la coppia relativa a quel server per la versione 1 del protocollo, dato che non era mai stata scelta in precedenza per quel server, non potrà quindi accorgersi che si tratta di una chiave fasulla, e non stamperà nessun tipo di banner di warning.

Capitolo 4

Implementazione di ARP

Questo capitolo descrive l'implementazione del protocollo ARP nel kernel di Linux. Saranno descritte le principali strutture dati che riguardano il networking e la loro inizializzazione, quindi si illustrerà come l'ARP cache è implementata. Si presenteranno due interfacce di uso comune a livello utente per manipolare le tabelle ARP: ioctl() e socket netlink.
Nella seconda parte del capitolo verrà esposta una possibile soluzione al problema dell'ARP poisoning con i mezzi che si hanno a disposizione nell'implementazione attuale. Verrà analizzato Secure Link Layer, una soluzione proposta da Frank Hunleth per risolvere questo e altri problemi. I punti deboli di questa soluzione e le ragioni per cui non è stata presa come punto di partenza in questa tesi concludono il capitolo.

4.1 Implementazione nel kernel di Linux

Esattamente come il modello di rete ISO/OSI è composto da vari livelli (sezione 2.1), così l'implementazione dei protocolli di rete nel kernel Linux è realizzata a livelli, come illustrato in figura 4.1.

Al livello applicazione vediamo le socket INET che sono la parte di networking di una famiglia di socket molto più ampia: le BSD socket. Queste ultime sono una interfaccia comune per comunicare con le applicazioni e permettono di implementare la comunicazione tra processi, siano essi locali, tramite AF_UNIX, siano remoti, tramite AF_INET. Al di sotto dell'interfaccia delle socket troviamo il livello IP.

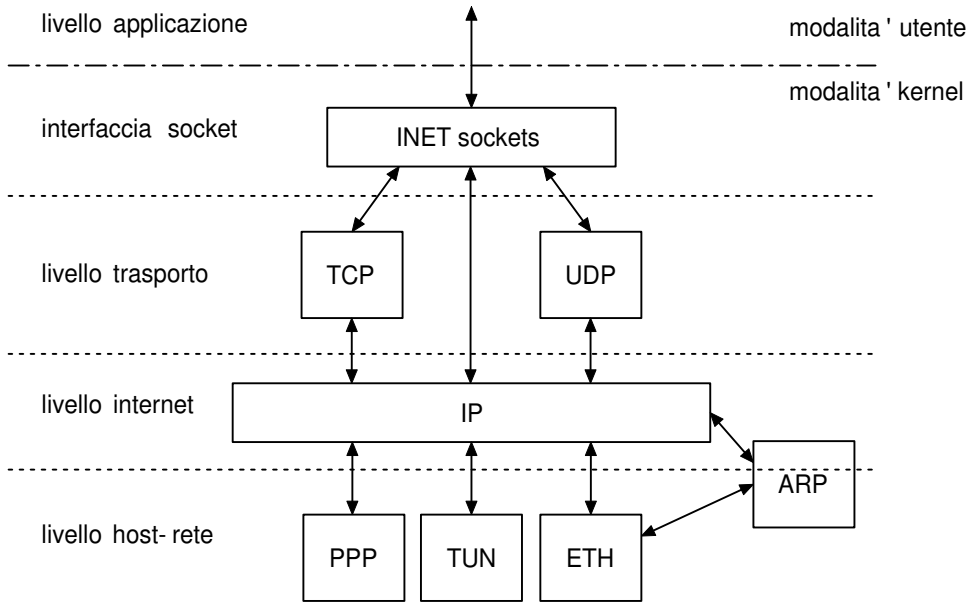

Figura 4.1: I layer del networking di linux

4.1.1 Il livello IP

Uno dei problemi di avere tutti questi livelli nello stack dei protocolli è la forte dipendenza che si crea tra essi. Ogni livello ha bisogno di aggiungere header ai dati quando devono essere trasmessi e toglierli quando il pacchetto è ricevuto. Ciò rende il passaggio dei dati tra livelli laborioso poiché ogni livello deve indovinare dove inizia e finisce il proprio header. Una soluzione potrebbe essere copiare i buffer del proprio payload al livello successivo, ma questo è molto inefficiente. Linux utilizza invece una struttura chiamata `sk_buff` per passare i dati attraverso i livelli e i device driver della scheda di rete. `sk_buff` contiene dei puntatori alla lunghezza, all'inizio e alla fine dei dati che ogni livello deve analizzare attraverso le funzioni implementate nel kernel.

4.1.1.1 La struttura sk_buff

La figura 4.2 mostra la struttura `sk_buff` definita in /usr/src/linux/include/linux/skbuff.h.

Ogni struttura ha un blocco `data` associato ad essa e attraverso 4 puntatori, head, data, tail, end, viene manipolato il suo contenuto:

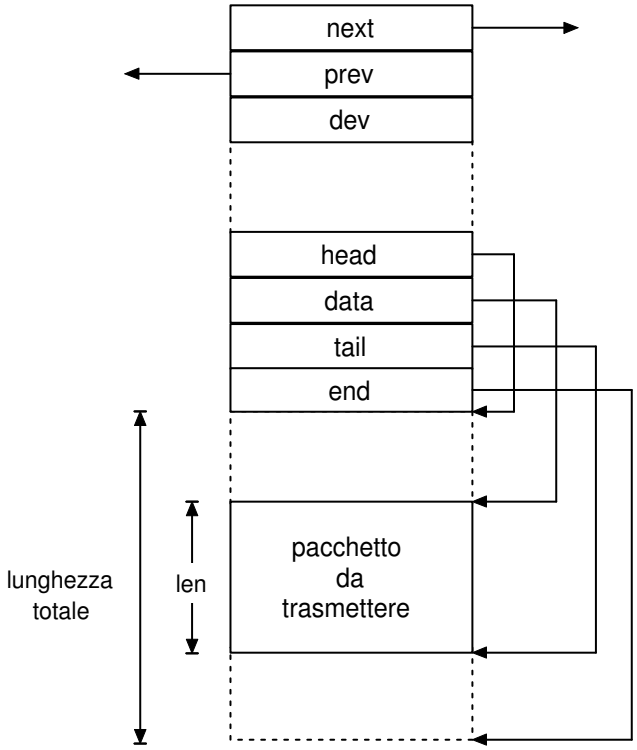

Figura 4.2: La struttura sk_buff

head: punta all'inizio dell'area `data`. Questo valore è fissato al momento dell'allocazione della struttura.

data: punta all'inizio dei dati del protocollo corrente. Questo puntatore varia a seconda del livello in cui la struttura è usata.

tail: punta alla fine dei dati del protocollo corrente.

end: punta alla fine del blocco dati che inizia dalla zona di memoria puntata da head.

Oltre a questi puntatori, ve ne sono molti altri che servono per associare una sk_buff ad una specifica socket e per inserire tutte queste strutture in una lista circolare doppiamente linkata. Il kernel fornisce API per la manipolazione di queste strutture.

4.1.1.2 Ricezione di pacchetti

Il sistema di device driver di Linux consiste in una serie di strutture dati associate a ciascun device. Queste strutture sono collegate tra loro in una

lista che ha per base dev_base. Ogni struttura descrive il suo device e fornisce una serie di puntatori a funzione che vengono richiamate dal livello networking.
Quando un device riceve un pacchetto dalla rete il driver deve salvarlo in una struttura `sk_buff`. Questa struttura è poi aggiunta alla coda di ricezione. In caso la coda sia piena, la struttura è cancellata.

Quando il "bottom half" [1] del livello networking viene eseguito dallo scheduler del kernel, controlla questa coda e determina a quale livello deve essere passata la struttura `sk_buff`.

All'avvio di Linux, tutti livelli sono inizializzati e ognuno di essi registra una struttura `packet_type` nella hash table `ptype_base`. La definizione della struttura `packet_type` è riportata di seguito.

```
struct packet_type
{
   unsigned short type;       /* This is really htons(ether_type) */
   struct net_device *dev;    /* NULL is wildcarded here    */
   int (*func) (struct sk_buff *, struct net_device *,
                struct packet_type *);
   void *data;                /* Private to the packet type */
   struct packet_type *next;  /* next element */
};
```

In essa è contenuto il tipo (`type`) di pacchetto da registrare, il device (`dev`) al quale associarlo, il puntatore alla funzione (`func`) che deve essere utilizzata per processare il pacchetto e un puntatore (`data`) utilizzato internamente dal kernel per gestire gli skb condivisi.

La funzione che registra questa struttura è la:

```
void dev_add_pack(struct packet_type *pt);
```

definita in /usr/src/linux/net/core/dev.c

La hash table `ptype_base` viene usata per decidere quale protocollo debba ricevere il pacchetto appena ricevuto. Il "bottom half" controlla che il tipo del protocollo corrisponda a quello della struttura `sk_buff` e lo destina al livello appropriato.

[1] Insieme di funzioni che gestiscono le code di interrupt inizializzate dal "top half" [12]

4.1.2 Ricezione ARP reply

Allo startup del kernel, tutte le funzioni dichiarate come `__init` vengono eseguite per la corretta inizializzazione di tutti i moduli del kernel.

Nel file /usr/src/linux/net/ipv4/arp.c troviamo la funzione `void __init arp_init (void)` che tramite una chiamata a `dev_add_pack()` registra la struttura

```
static struct packet_type arp_packet_type = {
   type: __constant_htons(ETH_P_ARP),
   func: arp_rcv,
   data: (void*) 1, /* understand shared skbs */
};
```

nella hash table `ptype_base`. Da questo momento in avanti, alla ricezione di un frame con tipo ETH_P_ARP (0x0806) il "network bottom half" passerà la struttura `sk_buff` alla funzione:

```
int arp_rcv(struct sk_buff *skb, struct net_device *dev,
            struct packet_type *pt)
```

la quale si preoccuperà di gestire i pacchetti ARP per conto del kernel.
La gestione delle tabelle di ARP verrà spiegata nella sezione 4.1.4.

4.1.3 Spedizione ARP request

Quando il kernel necessita della risoluzione di un indirizzo di livello rete, ma questa non è presente nella ARP cache, il livello ARP deve mandare una ARP request sul cavo. Per fare questo, crea una nuova entry nelle tabelle di neighbor (sezione 4.1.4) e una struttura `sk_buff` con all'interno del campo data un pacchetto ARP request. Mette la struttura `sk_buff` in coda di trasmissione tramite `dev_queue_xmit()` e imposta il timer (circa un secondo) per la richiesta ARP. Se non arriva risposta in tempo, la funzione cercherà di fare altre request per un certo numero di volte (solitamente 3) e se entro questo limite non giunge ancora risposta, l'entry nella cache verrà rimossa. Tutte le strutture `sk_buff` in coda che dipendono da quella entry riceveranno una notifica di fallimento, e tocca al livello IP gestire la situazione

imprevista.
Se, al contrario, giunge una ARP reply, la entry nella ARP cache viene marcata come "completa" e tutte le `sk_buff` in coda possono essere trasmesse.

4.1.4 Il sistema di Neighbor

Il compito del sistema di Neighbor di Linux è quello di tradurre gli indirizzi IP (versione 4 o 6) in indirizzi di livello 2 (tipicamente Ethernet). Il livello IP ha bisogno di questa traduzione per poter passare le strutture `sk_buff` al device driver della scheda di rete.

Il Neighbor controlla se il device driver ha bisogno dell'header di livello 2 e, nel caso se questo debba essere ricostruito. Infatti Linux mantiene una cache di header link layer per evitare di doverlo ricreare ad ogni frame spedito. Questa operazione è controllata dalle strutture `neigh_ops` e dai puntatori a funzione in essa contenuta.

Il livello ARP di Linux è costruito intorno al sistema di Neighbor. Questo è inizializzato all'avvio del kernel. I livelli di risoluzione ARP, per IPv4 e NDISC, per IPv6, registrano all'interno del Neighbor delle strutture `neigh_table` tramite la chiamata alla funzione `neigh_table_init()`.
Una volta inizializzato, il Neighbor contiene nella cache tutte le coppie <IP, MAC address>. Queste entry sono allocate al primo utilizzo e deallocate dopo che il timeout è scaduto.
Ognuna di queste entry è formata da :

- un timestamp che indica l'ultimo utilizzo della entry
- un timestamp che indica l'ultima volta che la entry è stata aggiornata
- un flag per indicare lo stato della entry
- l'IP a cui si riferiscono
- l'indirizzo di livello 2 corrispondente
- l'header di livello 2 precostruito
- un timer per la ritrasmissione della richiesta

- un counter per il numero di ritrasmissioni
- una lista di `sk_buff` che sono dipendenti da questa entry

Gli stati di una entry possono essere:

none: il neighbor è vuoto

incomplete: il neighbor è in stato di risoluzione

reachable: il neighbor è valido e raggiungibile

stale: il neighbor è valido, ma probabilmente non raggiungibile, il kernel effetturà un controllo al primo utilizzo

delay: un pacchetto è stato mandato al neighbor stale, il kernel è in attesa di conferma

probe: il delay timer è scaduto, ma non ci sono state risposte, il kernel ha cominciato a fare probe con messaggi ARP/NDISC

failed: la risoluzione è fallita

noarp: il neighbour è valido, non sarà effettuato nessun controllo tramite pacchetti ARP

permanent: è come una noarp, ma solo l'amministratore può rimuovere la entry dalla cache

La topologia della rete può subire cambiamenti nel tempo e alcuni IP possono essere assegnati ad altri indirizzi di rete (es: sostituzione scheda di rete). Per far fronte a questa situazione il Neighbor controlla periodicamente le sue tabelle per controllare quali entry abbiano raggiunto il timeout. Queste entry possono essere rimosse solo dopo aver controllato che nessun device driver avesse un header in cache sul quale stesse lavorando. Alcune di queste entry invece sono statiche e non raggiungono mai il timeout, ne possono essere aggiornate da messaggi ARP.
La ARP cache non può crescere all'infinito, esiste infatti un numero massimo di entry sopra al quale non si può andare. Se questo limite viene raggiunto, il Neighbor inserirà la nuova entry facendo spazio cancellando quella con il timestamp più vecchio.

4.1.5 ioctl SIOC*ARP

Il kernel fornisce una interfaccia comune per manipolare le ARP entry presenti nella cache. Per poter fare ciò dalla modalità utente è possibile utilizzare la funzione `ioctl()`.

Questa interfaccia, sebbene ancora supportata, è obsoleta ed è stata sostituita dalle socket della famiglia `AF_NETLINK` che vedremo nella prossima sezione.

Per poter effettuare una operazione di I/O control (ioctl), è necessario aprire una socket di tipo AF_INET. Nel file net/ipv4/af_inet.c troviamo la funzione di gestione generica per le ioctl su socket `AF_INET`. Questa funzione è chiamata `inet_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)` e al suo interno uno switch decide, a seconda del parametro `cmd` passatole, quale ioctl specifica richiamare. Nel nostro caso specifico troviamo:

```
case SIOCDARP:
case SIOCGARP:
case SIOCSARP:
   return(arp_ioctl(cmd,(void *) arg));
```

dato che la `arp_ioctl()` si occupa della gestione di tutti i comandi riguardanti ARP.

I valori dei comandi sono dichiarati all'interno del file include/linux/sockios.h

```
#define SIOCDARP  0x8953      /* delete ARP table entry  */
#define SIOCGARP  0x8954      /* get ARP table entry     */
#define SIOCSARP  0x8955      /* set ARP table entry     */
```

La funzione `arp_ioctl()` è dichiarata all'interno del file net/ipv4/arp.c e svolge la funzione di copiare dal livello utente la struttura `struct arpreq` (dichiarata in include/linux/if_arp.h) passata alla `ioctl()`.

A seconda del comando richiesto viene invocata la routine di gestione e viene restituito in modalità utente la struttura `arpreq` riempita dei dati corretti.

```
struct arpreq {
```

```
   struct sockaddr arp_pa;       /* protocol address       */
   struct sockaddr arp_ha;       /* hardware address       */
   int arp_flags;                /* flags                  */
   struct sockaddr arp_netmask;  /* netmask for proxy arps */
   char arp_dev[16];
};
```

4.1.5.1 Esempio di utilizzo di ioctl()

Per poter utilizzare correttamente una ioctl da livello utente è necessario aprire una socket `AF_INET` sulla quale operare la I/O control e riempire una struttura `arpreq` per comunicare al kernel su quale parte della ARP cache vogliamo operare. Ad esempio per reperire le informazioni riguardanti una entry, possiamo utilizzare il seguente codice:

```
int sock;
struct arpreq ar;
struct sockaddr_in *sin;

memset((char *)*ar, 0, sizeof(struct arpreq));
strncpy(ar.arp_dev, "eth0", sizeof(ar.arp_dev));
sin = (struct sockaddr_in *)&ar.arp_pa;
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = 0x0700a8c0;   /* 192.168.0.7 */

sock = socket(AF_INET, SOCK_DGRAM, 0);

if (ioctl(sock, SIOCGARP, &ar) == -1)
   printf("LL_ADDR not found !!\n");
else
   printf("LL_ADDR: %s\n", ar.arp_ha.sa_data);

close(sock);
```

Specificando l'indirizzo richiesto (in questo caso 192.168.0.7) all'interno della struttura `sockaddr_in`, la `ioctl()` e' in grado di fornire informazioni

riguardanti la entry nella tabella ARP relativa a quell'indirizzo. Tali informazioni sono memorizzate nella struttura `arpreq` passata come parametro.

4.1.6 Le socket AF_NETLINK

Alexey Kuznetsov ha proposto un nuovo modo di comunicare con il kernel per tutto quello che riguarda i parametri del networking. Questa nuova interfaccia va sotto il nome di socket Netlink [13].

Le socket Netlink sono usate per trasferire informazioni tra il livello kernel e i processi al livello utente. La comunicazione è bidirezionale. La struttura delle Netlink consiste di una interfaccia socket tradizionale per le applicazioni e di una serie di API per i moduli a livello kernel.

4.1.6.1 Inizializzazione

Prima di vedere come usare le socket Netlink, vediamo come sono inizializzate all'interno del kernel.

Come visto in precedenza allo startup del kernel le funzioni dichiarate come `__init` vengono eseguite. Una funzione speciale, chiamata `do_basic_setup()` (nel file /usr/src/linux/init/main.c), si prende cura di inizializzare, tra le altre cose, la parte di networking attraverso la chiamata a `sock_init()`.
Questa funzione in /usr/src/linux/net/socket.c inizializza i protocolli con la chiamata `proto_init()`. La lista dei protocolli registrati è conservata in un array in /usr/src/linux/net/protocols.c :

```
struct net_proto protocols[] = {
   . . .
#ifdef  CONFIG_NETLINK
   { "NETLINK",  netlink_proto_init  },
#endif
   . . .
}

struct net_proto {
   const char *name;       /* Protocol name */
   void (*init_func)(struct net_proto *);  /* Bootstrap */
```

```
};
```

Ogni elemento dell'array è una struttura `net_proto` la quale contiene un puntatore a funzione per la funzione di inizializzazione. Nel nostro caso è la funzione `netlink_proto_init()`.
Il compito di `proto_init()` è quello di inizializzare tutte queste strutture mediante un ciclo.
La funzione `netlink_proto_init()` registra la famiglia della socket attraverso `socket_register()`:

```
struct net_proto_family
{
   int family;
   int (*create)(struct socket *sock, int protocol);
   short   authentication;
   short   encryption;
   short   encrypt_net;
};

struct net_proto_family netlink_family_ops = {
   PF_NETLINK,
   netlink_create
};

void netlink_proto_init(struct net_proto *pro)
{
   . . .
   sock_register(&netlink_family_ops);
   . . .
}
```

La `sock_register()` crea una entry nell'array `net_family` e associa ad esso le operazioni da svolgere alla creazione di una socket.

```
int sock_register(struct net_proto_family *ops)
{
```

```
    . . .
    net_families[ops->family]=ops;
    . . .
}
```

4.1.6.2 Creazione di una socket

Una socket Netlink a livello utente può essere creata nel seguente modo:

```
sock_fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
```

La famiglia di socket è `AF_NETLINK`, il tipo è `SOCK_RAW` anche se le Netlink sono socket di tipo `SOCK_DGRAM`. Entrambi i valori SOCK_RAW e SOCK_DGRAM sono validi, ma non viene fatta nessuna distinzione tra questi.
Il protocollo serve per distinguere quale modulo netlink utilizzare:

- NETLINK_ROUTE riceve gli aggiornamenti dei percorsi di routing e può essere usato per modificare le tabelle di routing, gli indirizzi IP, i parametri della scheda di rete, le tabelle di neighbor, la politica delle code, ecc praticamente tutti i parametri di networking possono essere modificati tramite questo protocollo netlink.
- NETLINK_FIREWALL riceve pacchetti mandati dal firewall per IPv4
- NETLINK_ARPD viene usato per modificare le tabelle ARP dal livello utente (attualmente non implementato)
- NETLINK_ROUTE6 viene usato per la gestione delle tabelle di routing per IPv6
- NETLINK_IP6_FW ha lo stesso uso di NETLINK_FIREWALL ma per IPv6 (non implementato).

`Socket()` è una system call che viene gestita dal kernel tramite la tabella delle syscall. A fronte di una chiamata a `socket()`, il kernel invoca `sys_socketcall()`, la quale richiama a sua volta `sys_socket()`:

```
asmlinkage int sys_socket(int family, int type, int protocol)
{
   . . .
   retval = sock_create(family, type, protocol, &sock);
   . . .
   retval = get_fd(sock->inode);
   . . .
}

int sock_create(int family, int type, int protocol,
struct socket **res)
{
   . . .
   sock = sock_alloc();
   i = net_families[family]->create(sock, protocol);
   . . .
}
```

Come possiamo vedere dal codice, la `sock_create()` utilizza il puntatore a funzione registrato nell'array `net_families` (vedere sezione precedente). Questo puntatore è collegato alla funzione `netlink_create()` che si occupa di creare le socket di tipo netlink. Una volta creata la socket è possibile comunicare tramite le funzioni `recvmsg()` e `sendmsg()`.

4.1.6.3 All'interno del kernel

All'interno del kernel le socket netlink vengono create tramite la funzione `netlink_kernel_create`. Come esempio esaminiamo la creazione della socket che gestirà le richieste NETLINK_ROUTE.

```
__initfunc(void rtnetlink_init(void))
{
   rtnl = netlink_kernel_create(NETLINK_ROUTE, rtnetlink_rcv);
   if (rtnl == NULL)
      panic("rtnetlink_init: cannot initialize rtnetlink\n");
   register_netdevice_notifier(&rtnetlink_dev_notifier);
```

```
   rtnetlink_links[PF_UNSPEC] = link_rtnetlink_table;
   rtnetlink_links[PF_PACKET] = link_rtnetlink_table;
}
```

Questa funzione è chiamata dalla `sock_init()` all'inizializzazione delle socket. Essa crea una socket netlink nel kernel che processerà le richieste da userspace.
Il codice della `netlink_kernel_create()` è il seguente:

```
struct sock *
netlink_kernel_create(int unit,
      void (*input)(struct sock *sk, int len))
{
   . . .
   if (netlink_create(sock, unit) < 0) {
      sock_release(sock);
      return NULL;
   }

   sk = sock->sk;

   if (input)
      sk->data_ready = input;

   netlink_insert(sk);
   . . .
}
```

Questa funzione crea una socket netlink e una entry nella `nl_table`. Essendo creata durante l'inizializzazione sarà la prima entry nella tabella. Questa socket viene creata con pid = 0 ed è il motivo per cui tutte le socket netlink in userspace che vogliono comunicare con questa socket devono essere inizializzate con pid = 0.
Il puntatore passato alla `netlink_kernel_create()` è la funzione `rtnetlink_rcv()` e sarà l'entry point verso kernelspace.

Seguiamo ora come un pacchetto netlink passa dalla modalità utente fino a kernel attraverso la funzione `sendmsg()`.

La `sendmsg()` è mappata sulla `sys_sendmsg()` che a sua volta richiama la `netlink_sendmsg()`. Questa funzione identifica a quale socket netlink deve essere passato il pacchetto controllando i pid di tutte le socket all'interno della tabella `nl_table`. Trovata la entry, la funzione richiama il puntatore a funzione (`data_ready`) relativo, che nel nostro caso era stato inizializzato a `rtnetlink_rcv()` per il caso `NETLINK_ROUTE`.
Il codice relativo è

```
int netlink_unicast(struct sock *ssk, struct sk_buff *skb,
u32 pid, int nonblock)
{
   . . .
   for (sk = nl_table[protocol]; sk; sk = sk->next) {
      if (sk->protinfo.af_netlink.pid != pid)
         continue;
   . . .
   sk->data_ready(sk, len);
}
```

All'interno della funzione `rtnetlink_rcv()` la struttura `sk_buff` è tolta dalla coda e viene passata a `rtnetlink_rcv_skb()` che a sua volta richiama la `rtnetlink_rcv_msg()`. Questa funzione estrae il contenuto del messaggio spedito da userspace e richiama la corretta funzione `doit()` cercandola all'interno dell'array `rtnetlink_links`.

4.1.6.4 Formato dei messaggi

I messaggi passati alla socket netlink devono avere il seguente formato:

```
struct {
   struct nlmsghdr     netlink_header;
   struct rtmsg        rt_message;
   char            buffer[1024];
} request;

struct nlmsghdr {
```

```
    __u32  nlmsg_len;   /* Length of message including header */
    __u16  nlmsg_type;  /* Message content */
    __u16  nlmsg_flags; /* Additional flags */
    __u32  nlmsg_seq;   /* Sequence number */
    __u32  nlmsg_pid;   /* Sending process PID */
};

struct rtmsg {
    unsigned char  rtm_family;
    unsigned char  rtm_dst_len;
    unsigned char  rtm_src_len;
    unsigned char  rtm_tos;
    unsigned char  rtm_table;
    unsigned char  rtm_protocol;
    unsigned char  rtm_scope;
    unsigned char  rtm_type;
    unsigned char  rtm_flags;
};
```

Il buffer da 1024 char è riempito con degli `rtattribute`, ognuno dei quali consiste di una struttura `rtattr` seguita dal valore che assume. La lunghezza di `rta_len` comprende la dimensione del valore e della struttura stessa.

```
struct rtattr
{
    unsigned short  rta_len;
    unsigned short  rta_type;
};
```

4.1.6.5 Esempio di utilizzo netlink

Illustriamo come si possa inserire una entry nella ARP cache. È possibile utilizzare il seguente codice:

```
struct rtnl_handle rth;
```

```
struct {
   struct nlmsghdr   n;
   struct ndmsg      ndm;
   char              buf[256];
} req;

inet_prefix dst;

memset(&req, 0, sizeof(req));

req.n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ndmsg));
req.n.nlmsg_flags = NLM_F_REQUEST|NLM_F_CREATE|NLM_F_REPLACE;
req.n.nlmsg_type = RTM_NEWNEIGH;
req.ndm.ndm_family = AF_INET;
req.ndm.ndm_state = NUD_REACHABLE;

/* add the IP */

get_addr(&dst, "192.168.0.1", AF_INET);
addattr_l(&req.n, sizeof(req), NDA_DST, &dst.data, dst.bytelen);

/* add the link layer address */

addattr_l(&req.n, sizeof(req), NDA_LLADDR, "01:02:03:04:05:06",
            LL_ADDR_LEN);
req.ndm.ndm_ifindex = ll_name_to_index(iface));

/* open the netlink socket */

memset(rth, 0, sizeof(rth));

rth->fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE);

memset(&rth->local, 0, sizeof(rth->local));
rth->local.nl_family = AF_NETLINK;
```

```
bind(rth->fd, (struct sockaddr*)&rth->local,
         sizeof(rth->local));

/* send data */

struct sockaddr_nl nladdr;
struct iovec iov = { (void*)req.n, req.n->nlmsg_len };
struct msghdr msg = {
   (void*)&nladdr, sizeof(nladdr),
   &iov, 1,
   NULL, 0,
   0
};

memset(&nladdr, 0, sizeof(nladdr));
nladdr.nl_family = AF_NETLINK;

sendmsg(rth->fd, &msg, 0);
```

4.2 Possibili contromisure

L'implementazione attuale rispecchia il protocollo ARP come descritto nel capitolo 2. In questo scenario le risposte ARP sono inserite in cache per aggiornare eventuali mutamenti di indirizzi link layer. Ad ogni ARP entry nella tabella di cache è associato un flag, come descritto nella sezione 4.1.4. Se una ARP entry è contrassegnata dal flag *permanent* (o static), non viene aggiornata alla ricezione di una *gratuitous ARP*.

E' quindi possibile inserire nella tabella delle entry permanenti che non subiranno gli aggiornamenti indesiderati. Questa soluzione può rivelarsi utile all'interno di reti di piccole dimensioni e con assegnazione statica degli IP. Al contrario, se la rete è di grosse dimensioni, o gli IP sono assegnati dinamicamente tramite DHCP, è praticamente impossibile per l'amministratore di rete configurare manualmente tutte le entry permanenti.

Una soluzione fattibile, seppur parziale, all'ARP poisoning da locale a remoto è quella di inserire ARP entry permanenti per l'IP del gateway nelle tabelle degli host. In questo modo non potrà essere intercettata la connes-

sione dall'host al gateway, ma è possibile intercettare il traffico che proviene dal gateway all'host. Un campanello d'allarme potrebbe essere suonato da un IDS posto sul gateway, ma questo non previene l'attacco.

4.3 Secure Link Layer

Frank Hunleth ha proposto il Secure Link Layer [14] per far fronte al pericolo di attacchi in rete locale. Il suo obiettivo è quello di cifrare e autenticare tutte le connessioni a livello Link Layer. In questo senso il Secure Link Layer può essere visto come un Secure Socket Layer (SSL) [15] a livello più basso nello stack ISO/OSI. SLL si avvale di una Certification Authority (CA) che firma digitalmente tutti i certificati degli host appartenenti alla rete locale. Un certificato SLL è composto da MAC address, Expiration Time, Host Public Key e dalla firma della CA stessa. Nella sua implementazione attuale la CA crea le chiavi private dei singoli host. Questo perché si ritiene che, essendo la CA un elemento "trusted" della rete, possa svolgere questo compito in tutta sicurezza.

4.3.1 Esempio di utilizzo di SLL

Un utente che voglia connettersi ad una rete SLL deve seguire i seguenti passi:

- Installare la patch SLL e ricompilare il kernel
- Comunicare alla CA il MAC address della scheda di rete
- Ottenere dalla CA il certificato SLL e le chiavi pubblica e privata
- Eseguire il comando `sllconfig` per caricare le informazioni nel kernel
- Connettersi alla rete

A questo punto le connessioni possono avvenire in modo del tutto trasparente. L'implementazione attuale non consente di autenticare i messaggi in broadcast o in multicast, solo quelli unicast sono autenticati.

Tutti i servizi non hanno inizio finché non è stata ultimata la mutua autenticazione tra client e server. DHCP e ARP sono protocolli che usano broadcast, ma le risposte sono mandate in unicast, quindi SLL è in grado di autenticare anche questo tipo di protocolli.

4.3.2 Funzionamento di Secure Link Layer

4.3.2.1 Autenticazione

Lo scambio delle chiavi di sessione e la mutua autenticazione sono realizzate usando la crittografia basata sulle curve ellittiche (Elliptic Curve Criptography, ECC) [16]. In particolare il protocollo [17] Menezes-Qu-Vanstone (MQV) è utilizzato per lo scambio della chiave di sessione AES. Il protocollo MQV si svolge in tre fasi. Le prime due consistono nello scambio di due chiavi temporanee che vengono poi usate per generare una chiave segreta al terzo passo. Questo protocollo è simile a Diffie Hellman e in più aggiunge la mutua autenticazione delle parti ed è resistente agli attacchi man in the middle. Per verificare che un host è stato autorizzato a far parte della rete locale viene utilizzato il Digital Signature Algorithm (DSA). Le firme digitali sono create con l'algoritmo Secure Hash Algorithm (SHA-1) e poi firmate digitalmente con DSA.

4.3.2.2 Codifica

Dopo l'autenticazione tutti i dati che passano attraverso il Link Layer sono cifrati con l'algoritmo Rijndael (AES) a 128-bit di chiave e 128-bit per blocco. I frame sono cifrati con l'algoritmo AES in modalità Cipher Block Chaining.
Ogni frame Ethernet è cifrato indipendentemente in modo che la perdita di un frame non comprometta l'intero stream. Ethernet infatti non garantisce che tutti i frame siano consegnati, poiché esiste il problema delle collisioni sul mezzo fisico e della perdita di frame nello switch a causa delle politiche delle code.

4.3.2.3 Formato dei messaggi SLL

SLL definisce tre tipi di messaggi che sono utilizzati per autenticare le parti.

Nome	Direzione	Contenuto
Auth 1	Client -> Server	Certificato SLL del client Chiave pubblica del client per MQV
Auth 2	Server -> Client	Certificato SLL del server Chiave pubblica del server per MQV Un numero casuale (K)
Auth 3	Client -> Server	K cifrato con AES usando la chiave segreta di MQV

La figura 4.3 mostra come SLL gestisce una ARP request. Una volta ricevuta la request (in broadcast) il ricevente prepara la ARP reply. SLL mette in coda questa reply e inizia l'autenticazione. Terminato lo scambio dei tre messaggi l'ARP reply cifrata viene spedita. L'host che riceve l'ARP reply è in grado di decifrarla poiché possiede la chiave di sessione scambiata tramite l'MQV.

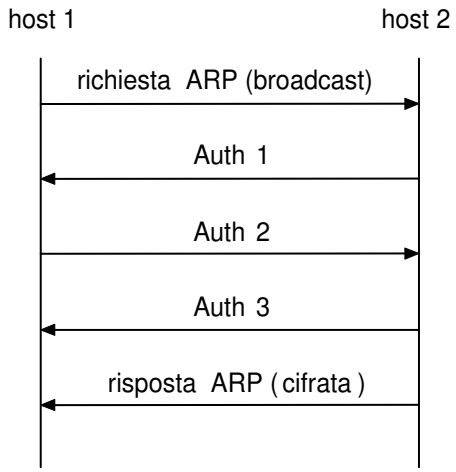

Figura 4.3: Flusso di messaggi iniziato da una ARP request

4.3.3 Implementazione di SLL

La versione attuale (ed unica implementazione) consiste in una patch al kernel di Linux versione 2.2.16. Sono stati incorporati nel kernel numerosi

nuovi file per la gestione dei numeri grandi (bignum) e per gli algoritmi ECC, MQV e DSA. Tutti questi nuovi file si trovano nella directory net/ipv4/.
Oltre ai file di appoggio è stato necessario aggiungere una `ioctl()` per configurare dal livello utente i parametri di SLL nel kernel. Una nuova voce è stata aggiunta al proc filesystem (/proc/net/sll) per visualizzare alcune statistiche riguardanti SLL. Infine una funzione SLL è stata aggiunta per intercettare i pacchetti in fase di spedizione/ricezione nello stack TCP/IP.

La procedura per la codifica/decodifica dei dati tramite AES è stata inserita nella funzione `bn_net()` nel file net/core/dev.c Questo fa sì che ogni dato che il kernel invia al driver di rete venga prima codificato dalla funzione di SLL.

Le funzioni per l'autenticazione sono state inserite nel layer ARP (net/ipv4/arp.c). Poiché ogni dato che attraversa lo stack TCP/IP verso il basso è obbligato a passare dal layer ARP, questo sembrava il luogo migliore per memorizzare i pacchetti finché l'autenticazione non fosse ultimata.

4.3.4 Benchmark

SLL ha due principali costi in termini di tempo per l'utente. Il primo è il tempo impiegato ad autenticare i due host per la prima volta o dopo un lungo periodo senza spedire pacchetti. Il secondo è il tempo impiegato a codificare/decodificare ogni frame Ethernet.
I risultati dei test [2] sono elencati nelle tabelle seguenti. I tempi in microsecondi per la fase di autenticazione :

Evento	**Pentium II 450**
Generazione Auth 1	28750
Elaborazione Auth 1 e generare Auth 2	25887
Elaborazione Auth 2 e generare Auth 3	92031
Elaborazione Auth 3	90786
Totale	**237454**

I tempi in microsecondi di overhead (codifica+decodifica) per pacchetti di varie dimensioni sono :

[2] Forniti dall'autore di SLL e non verificati

Dim. pacchetto	Pentium II 450
46	28.0
84	35.0
1500	290.5

I pacchetti da 46 byte sono tipicamente ARP request / reply. Quelli da 84 byte sono ICMP echo request / reply. 1500 è la lunghezza massima del pacchetto su Ethernet a pieno regime (trasferimento dati).

I tempi in microsecondi per trasferire un file da 1 MB via FTP sono i seguenti:

Modalità	Tempo (sec)	Throughput (MB/sec)	Slowdown
SLL disattivato	1.79	4.47	
SLL attivato	2.01	3.98	11 %

4.3.5 Osservazioni

Secure Link Layer è una buona soluzione in quanto utilizza algoritmi molto robusti e veloci. La crittografia a curve ellittiche è ancora oggetto di ricerca, ma ha dimostrato [16] di essere molto più robusta di RSA. E' stato calcolato che una chiave da 160 bit di ECC ha lo stesso grado di sicurezza di una chiave a 1024 bit di RSA. Rijndael (AES) è stato approvato come lo standard del XXI secolo [18], ed è veloce e sicuro.

Tuttavia riteniamo che si tratti di una soluzione non adatta al livello link. Esiste già Secure Socket Layer per le connessioni cifrate e opera a livello 4 (trasporto). Per il livello 3 è stato realizzato IPSEC. Non è necessario cifrare anche a questo livello. Ci sono messaggi di controllo ICMP che non necessitano affatto di essere protetti da occhi indiscreti.
SLL crea troppo overhead in modalità kernel. Tenendo presente che il kernel (almeno nella versione attuale) non è prelazionabile, i tempi di gestione delle chiavi sono proibitivi (237 msec). Questo significa che in quel lasso di tempo nessun altro task può essere eseguito. Ne consegue un notevole rallentamento della macchina che, in situazione di alto carico e molto traffico di rete, non riuscirebbe a gestire tutti gli interrupt generati.
Il kernel di un sistema operativo dovrebbe fornire solo le interfacce per

realizzare le operazioni, per le implementazioni il compito è spesso svolto a livello applicazione. Esempi di questo sono SSL e IPSEC i cui demoni di cifratura sono eseguiti in modalità utente.

Per difendersi dagli attacchi man in the middle è sufficiente l'autenticazione dei pacchetti ARP. La cifratura dei pacchetti può essere lasciata a livelli superiori, in modalità utente e non kernel, e solo per quelle applicazioni che trasportano dati sensibili.

Capitolo 5

Il protocollo Secure ARP

La crittografia a chiave pubblica ricopre un ruolo fondamentale per quanto riguarda l'autenticazione dei soggetti. In questo capitolo verrà presentata una breve introduzione alla firma digitale come supporto alla descrizione del protocollo Secure ARP. In particolare saranno descritti lo standard DSS, l'algoritmo DSA, la funzione di hash SHA-1 e lo scambio di chiavi Diffie Hellman.
In seguito, il capitolo descrive in dettaglio il protocollo Secure ARP, illustrando le procedure di scambio dei messaggi, il loro formato e la definizione dell'Authoritative Key Distributor.
Il capitolo si chiude con l'analisi di alcuni attacchi noti che Secure ARP riesce a fronteggiare.

5.1 Introduzione

Il problema più grosso illustrato nei capitoli precedenti riguardante il protocollo ARP è senza dubbio la mancanza di autenticazione dei messaggi. Infatti un host non è in grado di sapere chi sia l'effettivo mittente di tali messaggi. Esiste un campo "source address" nell'header Ethernet che dovrebbe assicurare la provenienza del frame, ma di fatto tale campo può assumere un qualsiasi valore (nessun controllo viene fatto sul mittente del frame). Questo porta ad una naturale diffidenza per tutte le risposte ARP che possono essere forgiate a piacimento da uno qualsiasi degli host della rete.

Una soluzione possibile a questo problema è rappresentata dalla possibilità di autenticare in maniera forte i mittenti dei messaggi. Tale operazione è ottenuta tramite l'uso di sistemi crittografici.

5.2 Autenticazione dei messaggi

Autenticare un messaggio tramite l'uso della crittografia significa mettere in grado il ricevente di verificare che il messaggio abbia l'origine dichiarata, non sia stato alterato e non possa essere ripudiato dal mittente.
Per poter fare ciò è necessario utilizzare algoritmi crittografici a chiave asimmetrica. Tali algoritmi prevedono due chiavi distinte per la fase di cifratura e quella di decifratura. Ogni coppia di chiavi deve essere scelta in modo che sia computazionalmente impossibile determinare la chiave privata possedendo la sua corrispondente pubblica.
Ogni individuo appartenente al gruppo di soggetti comunicanti possiede una sua chiave privata (tenuta segreta) e diffonde la corrispondente chiave pubblica.
Utilizzando la corrispondente chiave pubblica per decifrare è possibile verificare che un messaggio è stato effettivamente cifrato con la rispettiva chiave privata. Essendo la chiave privata segreta ed in possesso solo del mittente, il destinatario ha la certezza della provenienza del messaggio.

Questa operazione di cifratura è però molto costosa in termini di tempo macchina. Per risolvere questo problema si ricorre alla firma digitale. La firma non serve per ottenere la segretezza del messaggio ma solo la sua autenticazione.

5.2.1 Firma digitale

La firma digitale viene apposta mediante una sequenza di tre operazioni:

- Generazione di una impronta (o digest) del testo da firmare: al testo viene applicata una funzione di hash appositamente studiata che produce una stringa di lunghezza costante e piccola (128 - 160 bit). La funzione assicura l'unicità di tale stringa, nel senso che due testi diversi producono stringhe con hash diversi con probabilità $1/2^{160}$.

- Generazione della firma mediante cifratura dell'impronta.
- Apposizione della firma in coda al documento.

L'operazione di verifica della firma viene effettuata calcolando, con la stessa funzione di hash, il valore dell'impronta sul testo a disposizione e controllando che il valore così ottenuto coincida con quello generato dalla decodifica della firma stessa.

Le chiavi pubbliche di ciascun soggetto possono essere reperite tramite un "Key Distribution Center" che fornisce le chiavi pubbliche su richiesta delle parti comunicanti. Le chiavi pubbliche sono registrate presso un'Autorità di Certificazione (CA). La CA distribuisce certificati, che sono documenti contenenti la chiave pubblica del soggetto, una data di validità del certificato e la firma digitale della CA. Questo garantisce che una chiave pubblica sia effettivamente quella del mittente o destinatario richiesto. La firma della CA garantisce che il certificati sia integro e che sia stato effettivamente rilasciato dalla CA stessa. Se le parti comunicanti ritengono che la CA che ha firmato il certificato sia sicura, hanno la garanzia che il certificato appartiene effettivamente al proprio interlocutore.

La CA quindi garantisce che non ci sia un "man in the middle" nella catena di chiavi pubbliche e private. Se non esistesse la CA uno scenario di questo tipo potrebbe verificarsi:

- Alice e Bob vogliono comunicare tra loro.
- Bob chiede ad Alice la sua chiave pubblica.
- Eva intercetta la richiesta a spedisce la sua chiave pubblica a Bob facendo credere che si tratti di quella di Alice.
- Bob cifra un messaggio per Alice con la chiave pubblica di Eva.
- Eva è in grado di decifrare il messaggio (la chiave usata è la sua).
- Eva, dopo averlo letto, cifra il messaggio per Alice con la corretta chiave di Alice.
- Alice riceve il messaggio e non si accorge dell'intruso Eva.

Discorso analogo può essere fatto sulle firme digitali. Infatti anch'esse si basano sulla corretta associazione <individuo, chiave pubblica>.

5.2.1.1 Digital Signature Standard (DSS)

Nell'agosto del 1991, il National Institute of Standard and Technology (NIST) propose il Digital Signature Algorithm (DSA), da usare nel nuovo Digital Signature Standard (DSS) [19]. Questo standard specifica un algoritmo di firme digitali a chiave pubblica (DSA) usato per le applicazioni di firme digitali e l'algoritmo per la funzione di hash (SHA) [1].

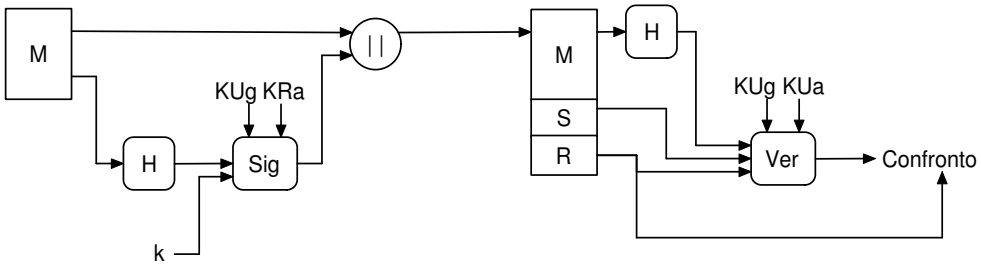

Figura 5.1: Firma digitale secondo DSS, usando SHA e DSA

La figura 5.1 mostra il funzionamento di DSS. Il digest del messaggio M fornito dalla funzione di hash SHA-1 (H in figura) viene dato in input ad una funzione di firma insieme ad un numero k generato casualmente (valido solo per questa firma). La funzione di firma dipende dalla chiave privata del mittente (KRa) e da una serie di parametri (p, q, g descritti in seguito) comuni ad un gruppo di soggetti comunicanti (KUg)[2]. Il risultato della funzione di firma è formato da due componenti: "r" e "s", apposte in fondo al messaggio M. Il ricevente calcola il digest SHA-1 sul messaggio M e lo mette in input, insieme ad r e s ad una funzione di verifica della firma. Questa funzione dipende dalla chiave pubblica del mittente (KUa) e dal set di parametri comuni. L'output della funzione di verifica deve essere identico alla componente "r" della firma.

Vediamo ora in dettaglio gli algoritmi utilizzati dallo standard DSS.

5.2.1.2 Secure Hash Algoritm (SHA-1)

L'algoritmo SHA è uno standard americano pubblicato nel documento FIPS PUB (Federal Information Processing Standards Publication) 180-1. Dal

[1] DSA è l'algoritmo; DSS è lo standard. Lo standard prevede l'uso dell'algoritmo e l'algoritmo è una parte dello standard.

[2] I parametri posso anche variare da soggetto a soggetto, non è richiesto che siano uguali per tutti

momento che tale documento modifica e sostituisce un precedente standard, l'identificazione corretta dell'algoritmo è data da SHA-1, mentre SHA si riferisce alla precedente versione [16].

Per un messaggio di lunghezza non superiore a 2^{64} bit, SHA-1 produce una rappresentazione condensata lunga 160 bit che prende il nome di digest. SHA è basato su MD4 [16] e il funzionamento è molto simile. Il messaggio di input è elaborato a blocchi di 512 bit.

SHA-1 è stato progettato in modo da soddisfare due proprietà essenziali:

- che risulti computazionalmente impraticabile determinare un messaggio che corrisponda ad un certo digest;
- che sia estremamente improbabile trovare due messaggi differenti che restituiscano lo stesso digest.

Usando un metodo esaustivo, la difficoltà di produrre un messaggio dato un digest è dell'ordine di 2^{160} operazioni. Quella per produrre due messaggi con lo stesso digest è di 2^{80} operazioni [16]. Quindi SHA-1 è considerato molto resistente agli attacchi di tipo brute force.

5.2.1.3 Digital Signature Algoritm (DSA)

La sicurezza dell'algoritmo DSA si basa sulla difficoltà del calcolo dei logaritmi discreti [16]. La figura 5.2 riassume i parametri necessari per il funzionamento dell'algoritmo. Ci sono tre parametri (p, q, g) che sono pubblici e possono essere comuni ad un gruppo di soggetti comunicanti; tuttavia non è condizione necessaria.

Il calcolo della chiave pubblica dell'utente a partire dalla privata è relativamente semplice, ma è ritenuto computazionalmente intrattabile il calcolo inverso.

Per la creazione della firma digitale di un messaggio M (corrispondente alla sua funzione hash H(M)) l'utente utilizza la sua chiave privata x e i parametri pubblici precedentemente introdotti secondo lo schema presentato in figura 5.3

Generalmente i valori di r, s e q sono di 160 bit, mentre i valori di p, g, y sono di 1024 bit.

Chiave Pubblica globale

p numero primo di L bit con $2^{L-1} < p < 2^L$, L deve essere un multiplo di 64 bit nell'intervallo [512, 1924]

q numero primo di 160 bit divisore primo di (p - 1), con $2^{159} < q < 2^{160}$

g = $h^{(p-1)/q}$ mod p con h numero intero tale che $1 \leq h \leq q-1$ e tale $h^{(p-1)/q}$ mod p > 1

Chiave Privata dell'utente

x numero random o pseudorandom con $0 < x < q$

Chiave Pubblica dell'utente

y = g^x mod p, con g > 1[5]

Parametro segreto per messaggio

k numero random o pseudorandom con $0 < k < q$

Figura 5.2: Parametri dell'algoritmo DSA

Per la verifica della firma digitale di un messaggio M vengono utilizzati i valori della chiave pubblica y dell'utente e i parametri pubblici p, q, g precedentemente introdotti secondo lo schema in figura 5.4

È da notare come il test di verifica sia su r, il quale non dipende affatto dal messaggio originale. r è una funzione di k e dei tre parametri globali p, q e g. Non è immediato come questo possa funzionare, ma esiste una dimostrazione che ne prova l'efficacia [16].

Data la complessità di calcolare algoritmi discreti [16], è impossibile risalire a k partendo da r o a x partendo da s.

Il costo maggiore in termini di tempo durante l'apposizione della firma DSA è il calcolo di g^k mod p. Questo valore, insieme a quelli di k^{-1}, non dipende dal messaggio che deve essere firmato, può quindi essere precalcolato in modo da accelerare le operazioni di firma.

Creazione della firma

r = $(g^k \bmod p) \bmod q$

kinv = $k^{-1} \bmod q$

s = $(kinv * (H(M) + xr)) \bmod q$

signature = (r,s)

Figura 5.3: Creazione della firma DSA.

Verifica della firma

w = $s^{-1} \bmod q$

u1 = $(H(M) * w) \bmod q$

u2 = $r * w \bmod q$

v = $((g^{u1} * y^{u2}) \bmod p) \bmod q$

TEST: v == r

Figura 5.4: Verifica della firma DSA.

5.2.1.4 L'algoritmo Diffie Hellman

Lo scopo di questo algoritmo è di permettere a due utenti di scambiarsi in modo sicuro una chiave segreta che possa essere usata successivamente per cifrare messaggi con algoritmi a chiave simmetrica.

Lo schema dell'algoritmo è mostrato in figura 5.5

Alla fine dello scambio, i due utenti sono in grado di calcolarsi la medesima chiave segreta K. Un eventuale attaccante, conoscendo solo q, α, Y_a e Y_b sarebbe costretto a calcolare il logaritmo discreto per ricavare K.

Questo algoritmo è soggetto ad attacchi di tipo "man in the middle". Infatti se un attaccante fosse in grado di intercettare e sostituire i parametri Y_a e Y_b che gli utenti si scambiano, potrebbe forzare la creazione di una chiave segreta che anch'egli conosce.

Elementi Pubblici globali

q numero primo

α tale che $\alpha <$ q e α è una radice prima di q. una radice prima è un numero che elevato a tutte le potenze da 1 a q-1 genera tutti gli interi da 1 a q-1 modulo q.

Generazione chiave utente A	**Generazione chiave utente B**
X_a numero segreto con $X_a <$ q	X_b numero segreto con $X_b <$ q
$Y_a = \alpha^{X_a} \bmod$ q (numero pubblico)	$Y_b = \alpha^{X_b} \bmod$ q (numero pubblico)

Generazione chiave segreta utente A	**Generazione chiave segreta utente B**
$\mathbf{K} = Y_b^{X_a} \bmod$ q	$\mathbf{K} = Y_a^{X_b} \bmod$ q

Figura 5.5: Parametri per l'algoritmo Diffie-Hellman

5.3 Il protocollo Secure ARP

Il protocollo Secure ARP è un estensione del protocollo ARP. Il formato di base dei messaggi, i timeout, la cache sono conformi alle specifiche del protocollo originale [4]. Secure ARP aggiunge una firma digitale alle risposte ARP in modo da autenticare il mittente. Potendo garantire la provenienza di un messaggio, gli attacchi visti nel capitolo 3.2 risulterebbero inefficaci. In questo modo l'attaccante non può forzare l'associazione di un MAC address con un determinato IP address poiché non è in possesso della chiave per firmare garantendo l'autenticità di tale accoppiamento. Per le richieste tale autenticazione non è necessaria, poiché non è necessario identificare i mittenti delle richieste di risoluzione.

L'estensione è realizzata grazie all'aggiunta di un header (denominato sarp) in coda alle risposte arp convenzionali. Questo particolare header verrà presentato nella sezione 5.3.5 e contiene le informazioni necessarie per autenticare i messaggi e per scambiare messaggi con l'Autoritative Key Distributor.

La scelta di accodare un header supplementare invece di integrare i nuovi

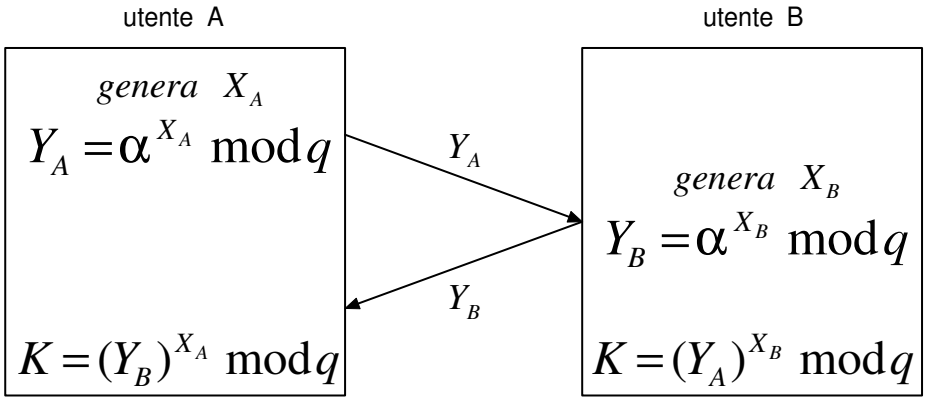

Figura 5.6: Scambio della chiave tramite Diffie Hellman

campi con quelli esistenti è motivata dalla volontà di mantenere compatibilità con gli host che non supportano tale estensione. Come verrà mostrato nel paragrafo 5.3.4, gli host che non supportano S-ARP saranno comunque in grado di analizzare la prima parte dei messaggi e di interpretarla come normali messaggi ARP.

5.3.1 Gestione delle chiavi crittografiche

Il protocollo Secure ARP si basa sulla crittografia asimmetrica descritta nel paragrafo 5.2.1.3. Per poter ottenere messaggi autenticati, ogni host della rete deve generare una coppia di chiavi (una pubblica e una privata). La chiave pubblica deve quindi essere messa a disposizione di tutti gli host del dominio di broadcast. Una soluzione è quella di eleggere un host della rete sia a Key Distributor che a Certification Autority.

La gestione delle chiavi crittografiche è affidata ad un host della rete locale denominato Authoritative Key Distributor (AKD). Poiché ARP è un protocollo locale, che funziona solo all'interno del dominio di broadcast, l'AKD deve trovarsi all'interno di tale rete. Ogni rete locale avrà quindi un host dedicato alla gestione delle chiavi, che può coincidere con il gateway di tale rete. Infatti, dato che ogni LAN (a meno che sia isolata) ha una macchina dedicata alla funzione di gateway, dedicarne un'altra per l'AKD comporterebbe uno spreco di risorse. Sia il gateway che l'AKD, al di là della topologia adottata, devono essere molto ben protetti, infatti la compromissione di uno dei due porterebbe ad un attacco di tipo man in the middle.

5.3.2 Configurazione iniziale

Per il corretto funzionamento del protocollo ogni host deve generare una coppia di chiavi pubblica/privata. La chiave pubblica deve essere memorizzata sull'Authoritative Key Distribtor attraverso un canale sicuro. Tipicamente viene memorizzata manualmente dall'amministratore di rete. Una volta che le chiavi sono state memorizzate il protocollo è pronto per iniziare.

5.3.2.1 L'Authoritative Key Distributor

L'AKD svolge i seguenti compiti:

- distribuisce le chiavi pubbliche verso gli host che le richiedono;
- certifica che tali chiavi siano corrette;
- fornisce agli host il timestamp al quale devono sincronizzare i timestamp dei messaggi.

La distribuzione delle chiavi avviene su esplicita richiesta dei singoli host della rete. Quando un host riceve una S-ARP reply, deve verificarne l'autenticità. Per svolgere tale compito ha bisogno della chiave pubblica associata al mittente. La chiave pubblica viene richiesta all'AKD il quale cerca nel suo database di chiavi e risponde alla richiesta.
Se la ricerca ha esito positivo la risposta conterrà la chiave pubblica del mittente, altrimenti l'host deve essere avvisato che tale chiave non è presente sull'AKD.
E' consigliato che gli host mantengano una cache nella quale memorizzare le chiavi pubbliche precedentemente chieste all'AKD in modo da ridurre il traffico di messaggi S-ARP.

La certificazione delle chiavi pubbliche consiste nel firmare con la propria chiave privata le risposte che l'AKD manda ai singoli host. In questo modo è garantita l'autenticità dei messaggi dell'AKD. Ogni host infatti possiede la chiave pubblica dell'AKD dallo startup del protocollo (vedi sezione 5.3.2) e può quindi verificarne la validità.
La chiave pubblica dell'AKD deve essere abbastanza robusta da garantirne

la non riproducibilità in tempi brevi. Infatti cambiare la chiave pubblica dell'AKD significherebbe dover aggiornare tutti gli host della rete per accettare messaggi firmati con la nuova chiave.

Il timestamp è fornito all'interno di ogni risposta proveniente dall'AKD. Ogni host deve mantenere questo timestamp come riferimento per impostare il timestamp di ogni sua risposta firmata. E' consigliabile memorizzare su ogni host la differenza (delta) tra il proprio timestamp e quello dell'AKD e utilizzare questo delta per aggiustare il proprio timestamp al momento della firma dei messaggi.
La sincronizzazione temporale degli host è parte integrante dell'infrastruttura di sicurezza del protocollo Secure ARP, quindi è stato necessario integrarla all'interno del protocollo stesso.
Vedremo in seguito (paragrafo 5.4.3) come la mancanza di timestamp possa portare ad un replay attack contro il protocollo.

5.3.2.2 Il database delle chiavi

L'AKD possiede un database delle chiavi statico. Il database viene popolato all'avvio dell'host e **non** può essere modificato dinamicamente da messaggi scambiati su canale insicuro (la rete locale).
Le chiavi sono salvate nell'AKD durante il setup del protocollo (sezione 5.3.2) e caricate in memoria. Per aggiungere una chiave al database è necessario copiarla fisicamente sull'AKD (tramite un canale sicuro) e forzare esplicitamente il caricamento in memoria tramite l'invio di un messaggio al demone che gestisce il database.

5.3.2.3 Il formato delle chiavi

Le chiavi utilizzate nel protocollo Secure ARP sono di tipo DSA descritte nel paragrafo 5.2.1.3. E' comunque possibile usare un qualsiasi algoritmo a chiave pubblica/privata come RSA o ECC (Elliptic Curve Criptography). L'unica differenza è nell'implementazione del protocollo e nei tempi di firma e verifica.
Nel caso si scelga di utilizzare DSA è consigliato usare una chiave forte (1024 bit) sull'AKD, mentre è sufficiente una chiave da 512 bit sui singoli host della rete. Questo accelera le operazioni di firma dei singoli host. Il tempo

richiesto per forzare una chiave DSA a 512 bit è notevolmente superiore al tempo medio di validità di una chiave, quindi anche una chiave a 512 bit può essere ritenuta più che sicura.

Le chiavi pubbliche degli host sono legate al loro indirizzo IP. Quindi un host è identificato in rete solo grazie a tale indirizzo. Se si decidesse di cambiare indirizzo IP ad un host, una nuova chiave dovrebbe essere generata e memorizzata nuovamente sull'AKD. Per situazioni in cui l'indirizzo IP è dinamico fare riferimento alla sezione 5.3.6.

Le chiavi pubbliche sono distribuite all'interno di un file speciale chiamato "certificato sarp". Questo facilita il compito di memorizzazione delle chiavi pubbliche degli host nel database dell'AKD e della chiave pubblica dell'AKD nei singoli host. Il formato di tale certificato è il seguente:

```
IP: <indirizzo ip dell'host>
LL: <indirizzo link layer dell'AKD, 00:00:00:00:00:00 se il
certificato e' di un host>
---BEGIN Secure ARP PUBLIC KEY---
<chiave pubblica DSA>
---END Secure ARP PUBLIC KEY---
<firma digitale del certificato stesso>
```

La chiave pubblica e la firma devono essere salvate in formato base64. Il certificato viene firmato da chi lo ha generato. In questo modo l'amministratore di rete è in grado di verificare che l'effettivo possessore della chiave privata corrispondente a quella pubblica contenuta nel certificato abbia generato il certificato stesso. La firma garantisce che il certificato non sia stato manomesso durante l'invio all'amministratore dell'AKD.

Ad esempio l'host 10.0.0.21 genera una chiave pubblica da memorizzare sull'AKD di questo tipo:

```
IP: 10.0.0.12
LL: 00:00:00:00:00:00
---BEGIN Secure ARP PUBLIC KEY---
MIIBoQKBgESoscuylPhC7HTACue6Z9DVCtM0b8iYoB8fEKGatOgcw8WmUvBHTkIU
DLMW5tLrFWBXpmMO6qqxLvQIHbS7zz4aJvd96Lw6LlL5KCVPFPUAWNXFD4iCkjdO
PBhw8aGg0Ud5i26z2ULo0o6onvBa8UfCc73IrUJ2fxPys8JzZ4dCAoGBAKGX+X9u
```

aR4ONpZyzTrxFiUqTAOz4ttJxYmdzZWi1r0Na5JUG25rNWriMbvt2UyNhvDbPOfZ
VoOF6JKAChzNLsi9ldDfo9gYe58AB5OPAwZOfSMd1cYGiNyf6+cxoSGUj3z8fRBq
SlnLwrJQcqpFuzp2eF8quE8VJNZuK5c7SoYpAhUAnfqmIwa2Y32Cl1kdd9QppvnL
bQkCgYBrmKcS5O/1xxCMMsnYbZywiEBZ6lHJ1EY/xxlCqsXJfhJIgn5GT/FVk3O5
B8gsj1ClE2pSXA7SLIZ/eqWoPafDCqIw9yl9qHGwFSMW+EpEqWQdDssXa4uqXvA/
Mz+Rwh2qBDDcAQx5vbbBIqfJbJZa1VmmgxeKkn1FdKMCPOXSDA==
---END Secure ARP PUBLIC KEY---
MCwCFHCvBMKs4+MbIrUwuuEbINbnsgNeAhQB1bogg12NUZRKM43d0UDqZXtX+w==

5.3.2.4 Validità delle chiavi

La validità di una chiave non è limitata da una expiration date come avviene per i certificati delle Certification Authority. Le chiavi sono valide finché non vengono cambiate. Ogni host può decidere ogni quanto generare una nuova coppia di chiavi. Per aggiornare la chiave pubblica è sufficiente comunicarla all'AKD.
Le chiavi mantenute in cache dagli host non hanno timeout. Al primo messaggio ricevuto in cui la firma non combacia con la chiave pubblica, l'host richiede nuovamente la chiave all'AKD. Se la chiave è cambiata l'host sarà in grado di verificare la firma, se la chiave è la stessa il messaggio sarà rifiutato.

5.3.3 Funzionamento generale del protocollo

Definizione degli oggetti:

AKD	Authoritative Key Distributor
H	Host
Rq(a)	Richiesta dell'oggetto a
Rp(a)	Risposta contenente l'oggetto a
$\mathbf{A}_h$	Indirizzo fisico dell'host h
C	Challenge random
T	Timestamp generico
$\mathbf{T}_g$	Timestamp globale
$\mathbf{T}_l$	Timestamp locale
D	Delta temporale
$\mathbf{K}_a$	Chiave pubblica dell'host a
$\mathbf{S}_a$	Firma dell'host a

Durante l'avvio degli host viene mandata una richiesta di sincronizzazione del timestamp. L'AKD risponde con il timestamp locale apponendogli la propria firma. L'AKD inoltre, deve inserire nella risposta un "challenge random" che l'host aveva mandato all'interno della richiesta. Questo serve all'host per controllare la validità della risposta in modo che non si possa effettuare un "replay attack" che desincronizzi il timestamp dell'host stesso. L'host che riceve un messaggio di sincronizzazione del timestamp, verifica la firma dell'AKD e la validità del "challenge" inviato. Se il "challenge" di ritorno non è identico a quello inviato, il messaggio verrà scartato. A questo punto l'host memorizza in una variabile (delta) la differenza tra il timestamp ricevuto e quello locale. In questo modo sarà in grado di verificare la validità dei timestamp delle risposte degli altri host. La somma del timestamp locale e del delta dà luogo al "timestamp globale".

H -> AKD :	Rq(T) || C
AKD -> H :	Rp(T) || C || S_{AKD}
H :	D = T - T_l

Figura 5.7: Sincronizzazione iniziale del timestamp

Quando un host necessita di risolvere un indirizzo IP nel suo corrispondente MAC address, una richiesta ARP viene spedita a tutti gli host della rete (vedere il capitolo 2 per il funzionamento di tale protocollo). Questa richiesta contiene come indirizzo di destinazione l'indirizzo di broadcast. Questo assicura che tutti gli host della rete ne ricevano una copia.

Alla ricezione di una richiesta ARP, l'host che possiede l'indirizzo IP contenuto nel campo Target Protocol Address dell'header ARP risponde con un messaggio autenticato. Questo messaggio contiene anche un timestamp che ne attesta il momento in cui è stato firmato.

$\mathbf{H}_i$ **-> broadcast** :	Rq(A_{H_j})
$\mathbf{H}_j$:	$T_g = T_l + D$
$\mathbf{H}_j$ **->** $\mathbf{H}_i$:	Rp(A_{H_j}) || T_g || S_{H_j}

Figura 5.8: Richiesta e risposta di risoluzione indirizzi S-ARP

L'host che ha fatto richiesta ARP riceverà un messaggio autenticato conte-

nente la risposta. I passi che devono essere svolti per accettare una risposta Secure ARP sono i seguenti:

- Controllare che il tipo di mezzo fisico e il protocollo siano supportati
- Verificare che si tratti di una risposta firmata
- Accertarsi che l'indirizzo contenuto nella risposta sia valido [3]
- Controllare che il timestamp sia valido [4].
- Verificare che la firma digitale corrisponda all'IP del mittente.

Se tutti i passi hanno esito positivo, la risposta può essere considerata valida e i dati in essa contenuti sono aggiunti alla cache di sistema.

Per poter verificare la validità di una firma, l'host deve essere in possesso della chiave pubblica del mittente. Se questa è già presente nella cache delle chiavi la verifica è immediata, altrimenti la chiave pubblica deve essere richiesta all'AKD.

Su richiesta di un host l'AKD deve inviare la chiave pubblica corrispondente all'IP contenuto nell'header della richiesta. La risposta deve essere firmata digitalmente per garantire che la provenienza della stessa.

$\mathbf{H}_i$ -> **AKD** :	$\mathrm{Rq}(\mathrm{K}_{H_j}) \mid\mid \mathrm{C}$
AKD -> $\mathbf{H}_i$:	$\mathrm{Rp}(\mathrm{K}_{H_j}) \mid\mid \mathrm{C} \mid\mid \mathrm{T} \mid\mid \mathrm{S}_{AKD}$
$\mathbf{H}_i$:	$\mathrm{D} = \mathrm{T}$ - T_l

Figura 5.9: Richiesta di una chiave all'AKD.

L'host riceve la chiave pubblica dell'IP associato alla risposta SARP pendente. Controlla che sia correttamente firmata dall'AKD. Verifica la validità della firma della risposta SARP pendente. Se tutte le operazioni di controllo vanno a buon fine, i dati sono inseriti nella cache ARP di sistema.
Se la chiave non è presente nel database dell'AKD un messaggio speciale informa che la chiave non è disponibile e le operazioni non possono essere

[3] Per essere valido deve appartenere alla rete locale

[4] La validità del timestamp può essere fissata arbitrariamente (dell'ordine di 30 secondi).

completate. In questo caso l'host si comporta come se la firma non fosse risultata valida.

Il timestamp è risincronizzato ad ogni messaggio proveniente dall'AKD.

5.3.3.1 Validità del timestamp

Una volta che tutti gli host possiedono la differenza del loro timestamp da quello dall'AKD, è possibile spedire messaggi con un timestamp globale. Sarà infatti sufficiente aggiungere tale delta al proprio timestamp per ottenere un valore a cui tutti gli host sono sincronizzati [5].

Avere messaggi autenticati contenenti un timestamp permette di non accettare messaggi troppo vecchi. Nella sezione 5.4.3 sarà spiegato il motivo per cui i messaggi con timestamp nel passato devono essere scartati prima di essere elaborati.

Uno scostamento accettato dal timestamp globale può essere dell'ordine di 30 secondi. Il valore di tale scostamento non è sottoposto a vincoli stretti, ogni implementazione può usarne uno differente. L'unica limitazione è che sia abbastanza piccolo.

5.3.4 Compatibilità all'indietro

Il protocollo è stato disegnato per essere interoperabile con la versione canonica di ARP. I messaggi di base sono sono stati alterati in modo tale da permettere agli host che non supportano S-ARP di interpretare i messaggi ricevuti. Infatti essendo l'header S-ARP semplicemente accodato al payload ARP, gli host che non sono predisposti per SARP si limiteranno a leggere il pacchetto ARP interpretando quello SARP come trailer da scartare.

Così facendo permettiamo ad host non che non supportano SARP di fare richieste e inserire le risposte nelle proprie cache, ma per poter comunicare, entrambi gli host devono avere in cache l'indirizzo dell'altro. Questo è un problema quando un host "SARP enabled" comunica con uno non abilitato. Per ovviare a questo problema l'implementazione può prevedere un file che contenga elenchi statici di IP e relativi MAC address che sono ritenuti fidati

[5] tale valore è il timestamp dell'AKD

da cui è possibile ricevere risposte non firmate. Ovviamente il MAC address inserito in cache non sarà quello ricevuto, ma quello contenuto nel file. Solo un controllo viene effettuato per verificare che la risposta corrisponda alla entry nel file.

I messaggi speciali (le richieste e risposte delle chiavi o dei timestamp) hanno nel campo `op` nell'header ARP il valore 0xff in modo che vengano ignorate dalle implementazioni del protocollo ARP (il quale supporta solo valori molto minori di tale soglia).

In questo modo la compatibilità con il protocollo ARP standard è assicurata. Gli host che non supportano SARP ricevono comunque le risposte ignorando l'header supplementare. Le richieste hanno lo stesso formato di quelle standard e i messaggi speciali sono ignorati.

5.3.5 Formato dei messaggi

Nel protocollo Secure ARP esistono tre gruppi di messaggi:

- risoluzione indirizzi
- richieste delle chiavi
- sincronizzazione temporale iniziale

ognuno dei quali è suddiviso in richiesta e risposta per un totale di 6 tipi di messaggi.
Le risoluzioni di indirizzi avvengono tra tutti gli host della rete (compreso l'AKD). Gli altri due tipi di messaggi sono scambiati solo con l'AKD.

L'header SARP viene accodato a normali pacchetti ARP come mostrato in figura 5.10

I messaggi ARP hanno lunghezza variabile, infatti gli indirizzi IP e MAC address possono cambiare a seconda del mezzo fisico utilizzato e della versioni di IP. L'header SARP contiene un campo chiamato "magic" che viene utilizzato per riconoscere l'header SARP. È necessario inserire questo campo poiché i pacchetti ARP hanno una dimensione di 42 byte e vengono estesi a 60 byte per motivi di lunghezza minima del frame Ethernet. Se non ci fosse il campo `magic` potremmo correre il rischio di interpretare il padding aggiunto come se fosse un header SARP.

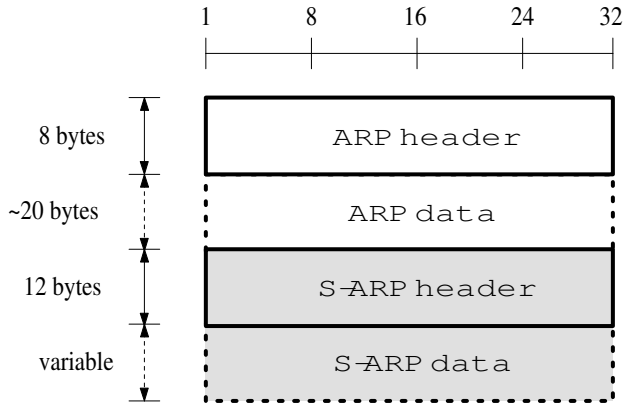

Figura 5.10: Header SARP accodato all'header ARP.

L'header SARP è descritto in figura 5.11

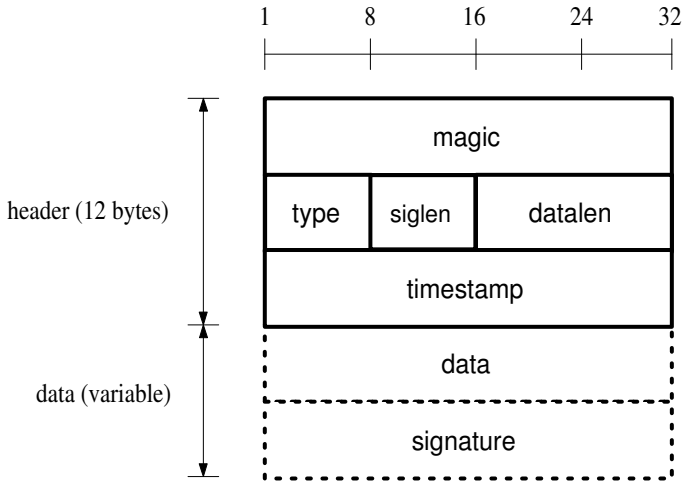

Figura 5.11: Header SARP

Il campo **magic** assume sempre il valore 0x7599e11e. Il campo **type** è il campo che discrimina i vari tipi di messaggi. I valori che può assumere questo campo rispecchiano le classi di messaggi descritte precedentemente:

```
#define SARPOP_SIGN             0x00     /* 0000 0000 */
#define SARPOP_TIME_REQUEST     0x01     /* 0000 0001 */
#define SARPOP_TIME_REPLY       0x02     /* 0000 0010 */
#define SARPOP_KEY_REQUEST      0x04     /* 0000 0100 */
#define SARPOP_KEY_REPLY        0x08     /* 0000 1000 */
```

Il campi **siglen** e **datalen** indicano rispettivamente la lunghezza dei dati e della firma DSA. **timestamp** è impostato al valore del timestamp globale al momento in cui è stata apposta la firma al pacchetto.

5.3.5.1 La firma

La firma da apporre ai messaggi che richiedono autenticazione è strutturata nel seguente modo. Si calcola l'hash (SHA-1) dell'header ARP + l'header SARP. Si firmano i 160 bit dell'hash con la chiave privata DSA, come spiegato nella sezione 5.2.1.3.

La firma deve essere apposta con il campo *siglen* uguale a zero. Successivamente questo campo assume il valore della reale lunghezza della firma. In ricezione quindi si dovrà tener conto che la firma deve essere controllata ponendo a zero il campo *siglen*.

5.3.5.2 S-ARP request

Le richieste di risoluzione di indirizzo non devono essere firmate a patto che non si utilizzino i dati in esse contenute per riempire la cache di sistema. E' quindi possibile utilizzare il vecchio header ARP senza aggiunte.
Per migliorare le performance è possibile firmare anche le richieste ed inserire in cache anche i dati ad esse relative.

5.3.5.3 S-ARP reply

Tutte le risposte a richieste di risoluzione di indirizzi devono essere firmate. La risposta consiste quindi di un pacchetto ARP convenzionale contenente i dati per la risposta, più un header supplementare per la firma.

Nell'header S-ARP il campo *type* è impostato a SARPOP_SIGN per indicare che è una risposta firmata. *datalen* viene posto a zero poiché non ci sono dati trasportati dal protocollo S-ARP. Infatti il pacchetto consiste solo in una firma dei dati dell'header ARP. Il *timestamp* è impostato con il valore del timestamp globale al momento della firma.

5.3.5.4 S-ARP KEY request

Questo tipo di messaggio è una sorta di ibrido tra gli header ARP e SARP. Infatti il campo *op* dell'header ARP viene impostato a 0xff in modo tale che non venga scambiato inavvertitamente per un messaggio ARP convenzionale.
Questo valore indica messaggi speciali da e per l'AKD. Il tipo di messaggio è indicato come sempre nel campo *type* dell'header SARP, in questo caso SARPOP_KEY_REQUEST. Il *timestamp* non è necessario poiché verrà ignorato dall'AKD.

La richiesta di una chiave non necessita firma da parte del mittente, quindi i campi *datalen* e *siglen* sono posti a zero.
Per sapere quale chiave cercare, l'AKD prende il valore contenuto nel campo Target Proto Address dell'header ARP e lo interpreta come l'IP da cercare nel proprio database. I campi Sender conterranno gli indirizzi hardware e proto del mittente della richiesta in modo che l'AKD possa rispondere al corretto destinatario.

5.3.5.5 S-ARP KEY reply

Ricevuta una SARPOP_KEY_REQUEST, l'AKD effettua la ricerca nel suo database interno e prepara una risposta da spedire al mittente della richiesta.
La chiave richiesta viene messa nel campo *data* e il rispettivo *datalen* aggiornato. Il *type* è SARPOP_KEY_REPLY e la signature viene apposta con le medesime modalità della firma dei messaggi SARPOP_SIGN.

Congiuntamente al valore SARPOP_KEY_REPLY è possibile sommare in binario (OR logico) i seguenti valori per indicare particolari risultati della ricerca della chiave nel database dell'AKD.

```
#define SARPOP_KEY_NOTFOUND   0x10     /* 0001 0000 */
#define SARPOP_KEY_INVALID    0x20     /* 0010 0000 */
```

In questo modo l'host riceve informazioni supplementari dall'AKD relative allo stato della chiave richiesta.

E' importante impostare il campo *timestamp* con il timestamp locale dell'AKD poiché l'host si sincronizzerà su di esso alla ricezione del messaggio.

5.3.5.6 S-ARP TIME request

La richiesta del timestamp dell'AKD non necessita firma da parte del mittente, quindi i campi *datalen* e *siglen* sono posti a zero. Il *type* è SARPOP_TIME_REQUEST. Il *timestamp* ovviamente è ininfluente.

5.3.5.7 S-ARP TIME reply

Ricevuta una SARPOP_TIME_REQUEST, l'AKD risponde impostando il timestamp locale nel campo *timestamp*. *data* e *datalen* sono eguagliati a zero, il *type* è SARPOP_TIME_REPLY e la firma viene apposta con le medesime modalità della firma dei messaggi SARPOP_SIGN.

5.3.6 Assegnazione dinamica degli IP

Abbiamo visto nella sezione 5.3.2.3 che le chiavi del protocollo Secure ARP sono legate staticamente all'indirizzo IP dell'host che le ha generate. Questo diventa un problema nell'ambito di reti in cui gli indirizzi IP sono assegnati dinamicamente tramite un server DHCP.

In presenza di un DHCP server ad ogni host viene assegnato un indirizzo IP differente ogni volta che si connette alla rete. Nasce quindi la necessità di associare dinamicamente gli indirizzi IP alle chiavi degli host.
In questo caso possiamo definire una strategia tra il server DHCP e l'AKD in modo tale che possa essere cambiata l'associazione <chiave, IP> in maniera trusted.

Per poter effettuare questa operazione è necessario che l'AKD conosca a priori la chiave pubblica del server DHCP. Il server DHCP a sua volta deve essere in grado di supportare il protocollo SARP. Chiameremo SDHCP un server con tale supporto.

Definizione degli oggetti:

L'host che richiede l'assegnazione di indirizzo dinamico, effettua la richiesta al DHCP server secondo il protocollo standard accodando alla richiesta un messaggio di identificazione. Questo messaggio consiste nell'hash della sua chiave pubblica e nella firma di tale hash. La firma viene apposta per

AKD	Authoritative Key Distributor
SDHCP	Secure ARP enabled DHCP server
H	Host
SHA(a)	Hash SHA-1 del messaggio a
$\mathbf{A}_H$	Indirizzo IP dell'host H
$\mathbf{P}_H$	Chiave pubblica dell'host H
$\mathbf{S}_H$	Signature dell'host H

verificare che il mittente della richiesta sia l'effettivo possessore della chiave privata corrispondente all'hash.

H -> SDHCP : DHCP request || SHA(P_H) || S_H

L'hash dovrà essere utilizzato per identificare velocemente la chiave pubblica dell'host che fa richiesta di assegnazione di indirizzo IP. Ovviamente tale chiave pubblica dovrà già essere presente nel database dell'AKD.
Per verificare che tale host sia autorizzato a far parte della rete locale, il DHCP ha bisogno di effettuare una richiesta all'AKD per la verifica della chiave inviatagli. Quindi il SDHCP spedisce un messaggio all'AKD contenente il messaggio di identificazione dell'host, l'indirizzo IP che gli verrà assegnato e la firma del SDHCP per autenticare il messaggio.

SDHCP -> AKD : SHA(P_H) || S_H || A_H || S_{SDHCP}

L'AKD verifica che il messaggio sia correttamente firmato dal SDHCP, identifica la chiave pubblica dell'host H, ne controlla la validità mediante la firma dell'hash SHA-1 e se tutte le operazioni sono completate con successo, aggiorna l'IP associato a quella chiave con l'IP proposto dall'SDHCP.
Sia in caso di verifica positiva, sia in caso negativo, l'AKD manda un messaggio al SDHCP per avvisarlo del risultato della verifica.

AKD -> SDHCP : (ACK / NACK) || S_{AKD}
SDHCP -> H : DHCP reply || IP_H || S_{SDHCP}

Se la verifica è stata positiva, il SDHCP può procedere con l'assegnazione dell'indirizzo IP proposto, altrimenti la procedura è interrotta e all'host H non verrà assegnato nessun IP.

5.4 Attacchi contro S-ARP

In questa sezione verranno presentati gli attacchi noti per i quali Secure ARP è stato progettato. Al momento non si conoscono attacchi che riescano a rendere inefficace la protezione offerta da Secure ARP, in futuro l'aumento della potenza di calcolo potrebbe inficiare la sicurezza degli algoritmi crittografici utilizzati.

I principali attacchi a cui Secure ARP resiste sono illustrati nei paragrafi successivi.

5.4.1 Spoofing dei messaggi

Lo spoofing dei messaggi è il problema del protocollo ARP che lo rendeva altamente insicuro. Infatti, come visto nel capitolo 3, la mancanza di autenticazione del mittente può portare alla compromissione delle tabelle di ARP degli host.
Secure ARP aggiunge la firma digitale ai messaggi e ne garantisce la provenienza. In questo modo non è possibile spedire messaggi con falsa identità.

Alcuni messaggi del protocollo Secure ARP non sono firmati (time request e key request) ma questo nulla toglie alla sicurezza del protocollo stesso. Infatti essendo messaggi contenenti richieste, anche se spoofate, non intaccano nessuna informazione sensibile.

5.4.2 Replay attack

Il replay attack consiste nel catturare un messaggio spedito da un host e rispedirlo senza alcuna modifica più avanti nel tempo. I messaggi critici di S-ARP sono firmati e di conseguenza non possono essere modificati senza che il ricevente se ne accorga. L'algoritmo di hash infatti garantisce l'integrità dei dati.
Se un attaccante decidesse di utilizzare un pacchetto precedentemente catturato per inviare una risposta più avanti nel tempo, non farebbe altro che spedire un pacchetto valido. Infatti le informazioni in esso contenute sono vere e firmate dal reale mittente. La richiesta di risoluzione ARP è legata all'IP che a sua volta è associato ad una chiave. Lo scopo dell'attaccante

sarebbe quello di mandare una risposta nella quale l'indirizzo IP sia quello richiesto, ma il MAC address corrisponda a quello dell'attaccante. In questo modo riuscirebbe a portare a termine l'attacco. L'integrità dei dati garantisce che l'attaccante non possa modificare i dati contenuti nella risposta.

Esiste però un tipo di replay attack che potrebbe essere portato a termine se non ci fossero i timestamp all'interno delle risposte.

5.4.3 Replay attack con delay

Supponiamo uno scenario di questo tipo. L'attaccante intercetta una risposta dell'host 192.168.0.1 con MAC address associato 01:01:01:01:01:01. Aspetta fino a che l'host 192.168.0.1 è spento o comunque non raggiungibile. A questo punto imposta il suo MAC address a 01:01:01:01:01:01 e spedisce il messaggio precedentemente catturato per rispondere alla richiesta di risoluzione ARP verso l'indirizzo 192.168.0.1. Questa risposta è perfettamente valida poiché è stata firmata dal reale proprietario della chiave privata. In questo modo l'attaccante è riuscito ad intercettare i frame destinati all'indirizzo 192.168.0.1.

L'introduzione dei timestamp nell'header Secure ARP impedisce questo tipo di attacco. Infatti la risposta non conterrà un timestamp valido al momento del suo riutilizzo. Gli host che riceveranno tale risposta la scarteranno senza inserire la relativa entry nella cache di sistema.

5.4.4 Protezione fornita da S-ARP

Gli attacchi su una rete locale, supponendo che sia non raggiungibile da Internet, possono essere suddivisi in due categorie. La prima raggruppa gli attacchi portati da agenti esterni all'organizzazione cui la rete locale appartiene, ad esempio persone munite di computer portatile che si collegano fisicamente ad una borchia della rete; la seconda raggruppa gli attacchi porati da agenti interni: macchine già collegate alla rete che vengono utilizzate da persone maleintezionate. Vediamo quali conseguenze portano entrambe le classi.

5.4.4.1 Agenti Esterni

Un agente esterno che si collega alla rete dalla propria macchina possiede permessi di root su di essa e può quindi effettuare attacchi a qualsiasi livello dello stack TCP/IP. Questo tipo di attacco è il più pericoloso per quanto riguarda la compromissione della rete locale. Infatti il requisito primario per portare a termine attacchi di grande portata è avere il pieno controllo di una macchina della rete. Se l'attaccante riesce a collegare una propria macchina, viene da sé che la sicurezza dell'intera rete può essere compromessa in modo grave. Sebbene non facilmente realizzabile in ambito aziendale, un simile attacco è di facile realizzazione in ambienti nei quali gli utenti possono collegarsi liberamente alla rete tramite macchine personali, quali biblioteche o università.

Gli agenti esterni possono essere facilmente arginabili, poiché per entrare a far parte di una rete Secure ARP è necessario inserire nel database dell'AKD la propria chiave pubblica. Essendo questa operazione svolta manualmente dal gestore della rete, difficilmente l'agente esterno potrà comunicare con gli altri host della rete, salvo che non riesca a violare l'AKD.

5.4.4.2 Agenti interni

Una seconda minaccia è rappresentata da macchine che già fanno parte della rete locale ma che sono state compromesse dall'esterno o dagli utenti che hanno regolare accesso ad esse. Questo caso è più pericoloso del precedente poiché la macchina è ritenuta trusted dagli altri host della rete e la sua chiave pubblica è già presente nel database dell'AKD.
In questo caso, non si può impedire che la macchina compromessa comunichi con le altre, ma il protocollo Secure ARP garantisce che la macchina compromessa non possa forgiare false risposte ARP per cercare di portare a termine un attacco di tipo man in the middle.

Capitolo 6

Implementazione del protocollo

In questo capitolo verrà descritto il modo in cui Secure ARP è stato implementato per Linux con kernel 2.4.xx. Verrà illustrata la separazione dei compiti tra modalità kernel e modalità utente, facendo riferimento al codice del modulo kernel e del demone sarpd. Le principali strutture dati e scelte implementative saranno descritte in dettaglio.

6.1 Scelte implementative

Il kernel 2.4.xx non è prelazionabile, questo significa che quando una funzione è in esecuzione in modalità kernel non può essere interrotta dallo scheduler. Di conseguenza le funzioni devono essere molto veloci, poiché il resto del kernel è in attesa durante la loro esecuzione. Inserire le fasi di autenticazione nel kernel, implicherebbe sovraccaricare il kernel di operazioni durante le quali altre funzioni non possono essere eseguite. Considerando che la crittografia a chiave pubblica comporta tempi di calcolo molto elevati, si è preferito demandare questo compito ad un demone in modalità utente.

L'implementazione del protocollo è stata quindi divisa in due parti: un modulo kernel e un demone utente. Il primo atto a modificare solo i parametri strettamente necessari all'interno del kernel, la seconda implementa tutti i controlli che richiedono molto tempo CPU per essere effettuati.

Durante la fase di implementazione è stata fatta attenzione nel modificare il meno possibile i sorgenti del kernel, per evitare la ricompilazione forzata del kernel. Non è quindi necessario ricompilare il kernel per abilitare il

protocollo Secure ARP. Il modulo può essere attivato in qualsiasi momento tramite il comando `insmod sarp.o` . Una volta caricato il modulo kernel, il demone deve essere attivato altrimenti l'host non sarebbe più in grado di comunicare via rete.

6.2 Descrizione dell'architettura

Il livello ARP all'interno del kernel è suddiviso in due parti indipendenti: una è composta dal sistema di neighbor che contiene tutte le tuple relative alle risoluzioni degli indirizzi; l'altra è formata dalle funzioni di ricezione pacchetti e si preoccupa di inserire i risultati delle richieste nelle tabelle di neighbor. Le funzioni di ricezione si registrano all'interno del kernel attraverso la funzione `dev_add_pack()`. In questo modo il kernel può elaborare un pacchetto di tipo ARP passando la struttura `sk_buff` ricevuta dal device di rete alla funzione registrata nella lista. Se il tipo di un pacchetto non viene trovato nella lista, il kernel non sarà in grado di elaborare il messaggio.

Il modulo kernel disabilita la ricezione dei messaggi ARP nel kernel, lasciando completamente operativo il supporto per la ARP cache e le tabelle di gestione dei neighbor con tutti i timeout relativi. Questa operazione è effettuata tramite la funzione `dev_remove_pack()`. Una volta disabilitata la ricezione dei pacchetti ARP, il kernel ingnorerà tali messaggi. Il sistema di neighbor non viene modificato dal modulo S-ARP. Lo stack TCP/IP continuerà a richiedere al livello ARP la generazione di messaggi per la risoluzione degli indirizzi. Questi messaggi vengono accodati dal kernel nella coda di trasmissione e successivamente spediti. In questa situazione il livello ARP lavora solo in fase di spedizione. La ricezione è affidata al demone sarpd, il quale comunica al kernel le tuple da inserire nelle tabelle del sistema di neighbor.

La figura mostra la struttura generale dell'implementazione del protocollo S-ARP. La spedizione delle richieste ARP è gestita dal kernel, mentre la ricezione è elaborata dal demone sarpd che, una volta effettuati i controlli relativi all'autenticazione, inserisce nel kernel le tuple relative (freccia tratto-punto).

Il demone sarpd deve quindi sopperire alla mancata ricezione del kernel dei messaggi ARP e al loro inserimento nelle tabelle di sistema. Il demone

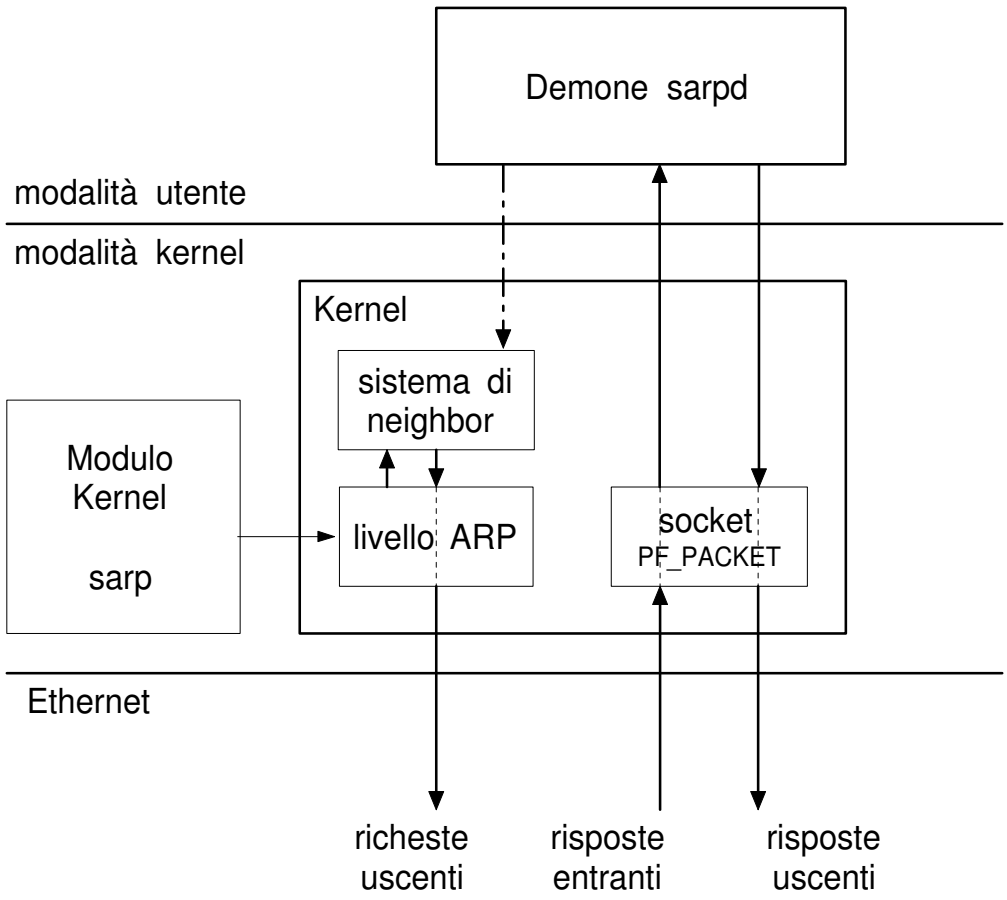

Figura 6.1: Architettura dell'implementazione Secure ARP

quindi, è posto in ascolto su tutte le interfacce a livello datalink e verifica la validità dei messaggi ricevuti. Se tutti i test hanno esito positivo la relativa entry viene inserita nella cache di sistema tramite apposite socket netlink.

6.3 Modulo Kernel

Per facilitare la modifica dei parametri del kernel senza però modificare i sorgenti del kernel è possibile utilizzare un modulo kernel che può essere caricato in qualsiasi momento. Un modulo kernel non è direttamente eseguibile ma è un codice oggetto che deve essere collegato al kernel a tempo di esecuzione. Le operazioni di caricamento e rimozione sono svolte dai comandi `insmod` e `rmmod` . Durante queste due fasi il kernel utilizza due funzioni che tutti i moduli caricabili dinamicamente devono implementare; una di inizializzazione e una di terminazione.

Il modo in cui queste funzioni sono implementate varia a seconda della versione del kernel presa in considerazione.

Il modulo kernel è molto semplice. Durante la fase di inizializzazione il modulo crea una file virtuale nel filesystem proc (/proc/sys/net/ipv4/sarp). Attraverso questo file è possibile abilitare o disabilitare la ricezione da parte del kernel dei messaggi ARP. Il demone sarpd utilizzerà questo file per disabilitare i messaggi ARP nel kernel durante la fase di inizializzazione.

6.3.1 init e cleanup

Il modulo kernel di Secure ARP può essere utilizzato indifferentemente su kernel linux 2.2.xx o 2.4.xx. Questo è reso possibile dalla dichiarazione condizionale delle funzioni di inizializzazione e terminazione.

```
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
   module_init(sarp_init);
   module_exit(sarp_exit);
#elif LINUX_VERSION_CODE < KERNEL_VERSION(2,4,0)
   void cleanup_module(void)
   {
      sarp_exit();
   }

   int init_module(void)
   {
      return sarp_init();
   }
#endif
```

Infatti nei kernel 2.4.xx le macro module_init() e module_exit() vengono utilizzate per definire le funzioni da utilizzare durante lo startup e la terminazione del modulo. Nel kernel 2.2.xx è necessario definire le due funzioni init_module() e cleanup_module().
Utilizzando questo tipo di compilazione condizionale, si rende il modulo utilizzabile su entrambe le versioni del kernel Linux.

6.3.2 Entry nel proc filesystem

Durante la fase di avvio, il modulo crea il file virtuale "sys/net/ipv4/sarp" nel filesystem virtuale denominato "proc" e assegna ad esso gli handler per la lettura e la scrittura. Questo file è un file speciale. Quando un'applicazione richiede la lettura o la scrittura su questo file, il kernel invoca gli handler che sono stati impostati; `secure_arp_proc_read()` per la `read()` e `secure_arp_proc_write()` per la `write()`.

```
struct proc_dir_entry *sarp_entry;

sarp_entry = create_proc_entry("sys/net/ipv4/sarp", 0644, NULL);

sarp_entry->read_proc = (read_proc_t *)&secure_arp_proc_read;
sarp_entry->write_proc = (write_proc_t *)&secure_arp_proc_write;
```

Le istruzioni sopra illustrate fanno in modo che ad ogni accesso (in lettura o scrittura) del file sys/net/ipv4/sarp vengano eseguite le relative funzioni.

Questo file virtuale è utilizzato per visualizzare e modificare lo stato del modulo dalla modalità utente. Conterrà infatti il valore 1 se il modulo è attivo, 0 al contrario. Questo valore può essere controllato facilmente tramite il comando :

```
cat /proc/sys/net/ipv4/sarp
```

Per abilitare o meno la funzionalità Secure ARP all'interno del kernel[1] è sufficiente usare il comando:

```
echo 1 > /proc/sys/net/ipv4/sarp
```

Al contrario per disabilitare il supporto sarà sufficiente scrivere 0 all'interno del file. Queste operazioni di lettura e scrittura provocano l'invocazione degli handler impostati per il file virtuale. I valori scritti nel file vengono direttamente passati alla funzione `secure_arp_proc_read()` che li gestisce come se fossero parametri.
Il file virtuale /proc/sys/net/ipv4/sarp viene creato con permessi 0644, il che significa che solo root può scrivere all'interno del file pur lasciando libera la lettura ad un qualsiasi utente del sistema.

[1] Il modulo deve essere già stato caricato attraverso il comando insmod

6.3.3 Modifica dei parametri del kernel

Il compito del modulo kernel è di inibire la ricezione di pacchetti ARP da parte del kernel. Infatti la gestione di tali pacchetti sarà demandata al demone S-ARP che utilizzerà le socket netlink per comunicare al kernel le entry da inserire nella cache di sistema.

Lo scopo del modulo è annullare la registrazione dei puntatori a funzione contenuti nella lista `dev_base`, in modo che il kernel non riceva più pacchetti ARP.
La funzione messa a disposizione dal kernel per effettuare questo annullamento è :

```
void dev_remove_pack(struct packet_type *pt);
```

Il problema principale è dato dal fatto che il kernel non è stato pensato per permettere la deregistrazione della struttura `arp_packet_type` da parte di un modulo, quindi non esporta il simbolo relativo. Questo simbolo è necessario per il corretto funzionamento delle funzioni `dev_remove_pack()` e `dev_add_pack()` le quali utilizzano tale indirizzo come valore di hash per la tabella di registrazione.

Un modo per risolvere questo problema consiste nel reperire l'indirizzo cercato nel file System.map. Questo file contiene gli indirizzi di tutti i simboli del kernel e viene creato durante la fase di installazione del kernel per facilitare le operazioni di debug. È possibile cercare all'interno di questo file il nome della struttura `arp_packet_type` per poterne ricavare l'indirizzo.

Il modulo kernel ha bisogno di questo indirizzo al momento della compilazione. Sarebbe possibile anche ricavarlo in fase di esecuzione, ma questo comporterebbe la ricerca manuale dell'indirizzo della struttura da parte del modulo. Una volta caricato, il modulo è parte integrante del kernel e non è consigliato aprire file dal kernel, anche perché il file potrebbe non esistere o essere stato creato da una versione precedente del kernel.
L'indirizzo di tale struttura viene passato al modulo, servendoci del Makefile. All'interno del sorgente definiamo una struttura di questo tipo:

```
struct packet_type *arp_packet_type = ARP_PACKET_TYPE_ADDR;
```

la costante ARP_PACKET_TYPE_ADDR verrà passata al momento della compilazione, come risultato della ricerca del valore di arp_packet_type nel file /boot/System.map correntemente in uso.

```
CFLAGS = -DARP_PACKET_TYPE_ADDR=‘grep -w arp_packet_type
              /boot/System.map | awk {’print "0x"$$1’}

all:
    $(CC) -c $(CFLAGS) sarp.c -o sarp.o
```

In questo modo la costante ARP_PACKET_TYPE_ADDR conterrà un valore del tipo 0xc02ddf10 che rappresenta l'indirizzo kernel della struttura desiderata.
È necessario fare attenzione che il System.map sia aggiornato al kernel correntemente utilizzato. poiché ogni kernel ha indirizzi diversi, assegnare un indirizzo non valido a tale struttura potrebbe causare errori indefiniti.

6.4 Demone in modalità utente

Poiché il kernel non può più ricevere i pacchetti ARP, nell'architettura di Secure ARP, il demone sarpd deve fornire questa funzionalità a livello utente. Il suo scopo principale è quindi quello di effettuare controlli sui pacchetti ricevuti e passarli al kernel solo se correttamente autenticati.

Il demone è configurabile tramite un file di configurazione che contiene alcune informazioni su dove reperire la chiave pubblica dell'AKD o le chiavi da inserire nel database, in caso venga eseguito in modalità AKD.

Il demone si pone in ascolto, attraverso le socket PF_PACKET, su tutte le interfacce presenti nel sistema a livello datalink. Attraverso queste socket analizza tutti i pacchetti ricevuti dalla scheda di rete. Questi messaggi vengono elaborati e le firme digitali sono verificate. Poiché il kernel non riceve più i messaggi ARP, non risponderà più alle richieste. Questa operazione viene svolta dal demone che prepara le risposte apponendovi la firma digitale.

Il demone mantiene la cache delle chiavi DSA degli host che hanno già comunicato con l'host sul quale gira il demone. Se il demone riceve un

messaggio inviato da un host la cui chiave pubblica non è in cache, inserisce il messaggio in una coda di attesa e invia una richiesta all'AKD per ricevere la chiave pubblica relativa. All'arrivo della chiave pubblica inviata dall'AKD, la coda di attesa viene analizzata in cerca di messaggi pendenti; la chiave pubblica viene inserita nella cache e il messaggio verificato. I messaggi che superano la verifica della firma digitale, vengono inseriti nelle tabelle di sistema mediante socket netlink.

6.4.1 Avvio del demone

Il demone allo startup carica la configurazione esaminando il file di configurazione e carica le chiavi in memoria (vedere il paragrafo relativo).

Completata la verifica della configurazione, il demone disabilita la ricezione dei pacchetti ARP da parte del kernel mediante la scrittura nel file virtuale creato dal modulo kernel.

```
if ((fd = open("/proc/sys/net/ipv4/sarp", O_WRONLY)) == -1)
   ERROR_MSG("can't open /proc/sys/net/ipv4/sarp");

if (write(fd, "1", 1) != 1)
   ERROR_MSG("can't enable secure arp");
```

Da questo momento in poi il kernel non riceverà più messaggi ARP.

Il passo successivo è quello di cancellare tutte le entry non STATIC presenti nelle tabelle di sistema. La funzione `neigh_flush_all_tables()` esegue un ciclo su tutte le interfacce e cancella le relative entry mediante la funzione `neigh_flush_table(char *iface)`

```
pcap_if_t *curdev, *nextdev;

for (curdev = (pcap_if_t *)GBL_PCAP->ifs;
                curdev != NULL; curdev = nextdev) {
   nextdev = curdev->next;

   neigh_flush_table(curdev->name);
}
```

la funzione `neigh_flush_table()` utilizza le socket netlink per cancellare le entry della ARP cache di una data interfaccia. Solo le entry che passano il filtro

```
filter.state = ~(NUD_PERMANENT|NUD_NOARP);
```

vengono cancellate. Questo permette di mantenere inalterate le entry STATIC (NUD_PERMANENT) impostate manualmente dall'utente o quelle che sono state impostate in modo da non richiedere risoluzione ARP (NUD_NOARP).

Le tabelle di ARP vengono cancellate solo dopo aver attivato il modulo kernel. Questo garantisce che nuove entry (non autenticate) non siano inserite mentre il demone effettua le operazioni di startup.

6.4.2 Cattura dei pacchetti a livello Link Layer

Il demone deve preoccuparsi di ricevere i pacchetti ARP e Secure ARP che viaggiano in rete. Le librerie pcap permettono di passare al demone i pacchetti catturati su tutte le interfacce.

Durante la fase di inizializzazione le librerie pcap aprono una socket che riceve solo pacchetti che contengono nel "proto" dell'header Ethernet il valore 0x0806.

```
#define PCAP_IFACE "any"
#define PCAP_FILTER "arp"
#define PCAP_PROMISC 0
#define PCAP_TIMEOUT 0

pd = pcap_open_live(PCAP_IFACE, GBL_PCAP->snaplen, PCAP_PROMISC,
      PCAP_TIMEOUT, pcap_errbuf);
ON_ERROR(pd, "%s", pcap_errbuf);

if (pcap_lookupnet(PCAP_IFACE, &net, &mask, pcap_errbuf) == -1)
   ERROR_MSG("%s", pcap_errbuf);

if (pcap_compile(pd, &bpf, PCAP_FILTER, 1, mask) < 0)
```

```
   ERROR_MSG("%s", pcap_errbuf);

if (pcap_setfilter(pd, &bpf) == -1)
   ERROR_MSG("pcap_setfilter");
```

PCAP_IFACE è impostato a "any". Questo garantisce che la socket riceva pacchetti da tutte le interfacce di rete. Il demone infatti disabilita la ricezione di pacchetti ARP da tutte le interfacce. Non è possibile discriminare su quale interfaccia riceve e su quale no a meno di impostare filtri mediante l'uso degli hook del firewall "net filter".

Questo tipo di socket (aperta su interfaccia any) non ritorna nei suoi header l'informazione relativa all'interfaccia dal quale il pacchetto è ricevuto. Per questo scopo è stata implementata la funzione

```
int belong_to_iface(u_int32 addr, char **iface);
```

la quale torna l'interfaccia a cui appartiene l'indirizzo "addr".

I pacchetti ricevuti sono passati al demone grazie alla callback sarp_get()

```
pcap_loop(GBL_PCAP->fd, -1, sarp_get, NULL);
```

Il demone è quindi bloccante. Si trova sempre in stato di idle in attesa dell'arrivo di pacchetto. Quando la callback attiva la funzione, tutte le operazioni del caso vengono eseguite per poi tornare in stato di idle.

La funzione sarp_get() si occupa di esaminare il contenuto del pacchetto ed intraprendere la strada opportuna. Un case sul tipo di messaggio distingue le funzioni che devono essere usate per gestire il messaggio.

6.4.3 Spedizione messaggi a livello Link Layer

Per scrivere a livello data link sono state utilizzate le librerie libnet 1.1.0. È stata implementata un'unica funzione che viene utilizzata per spedire i messaggi necessari.

```
int send_to_wire(char *iface, struct libnet_arp_hdr *arp)
{
   . . .
   l = libnet_init(LIBNET_LINK_ADV, iface, errbuf);
   ON_ERROR(l, "libnet fail: %s", errbuf);
   . . .
   len = sizeof(struct sarp_auth_msg) +
         sarp->siglen +
         ntohs(sarp->datalen);
   . . .
   if ((c = libnet_write(l)) == -1)
      ERROR_MSG("Write error: %s\n", libnet_geterror(l));

   libnet_destroy(l);
   SAFE_FREE(packet);

   return c;
}
```

La funzione ha come parametri l'interfaccia su cui spedire e un puntatore all'inizio del pacchetto SARP. Il calcolo sulla lunghezza del pacchetto viene effettuato automaticamente esaminando i campi "datalen" e "siglen" dell'header sarp.

6.4.4 Manipolazione ARP cache

Per la manipolazione delle tabelle contenenti le ARP entry, il kernel mette a disposizione due interfacce: la IOCTL e le socket NETLINK (come spiegato nel capitolo 4).
Si è preferito usare le socket netlink poiché l'interfaccia attraverso la ioctl() è ritenuta obsoleta. A tale scopo sono state usate le librerie libnetlink che forniscono una buona astrazione per le complesse strutture dei messaggi netlink.

Sono state implementate alcune funzioni "wrapper" all'interno del demone per facilitare ulteriormente la manipolazione dell'ARP cache.

```
void neigh_flush_table(char *iface);
```

Cancella tutte le entry nella tabella relativa all'interfaccia "iface" che non sono marcate con il flag NUD_PERMANENT.

```
void neigh_add(char *ll_addr, u_int32 ip, char *iface, int nud);
```

Inserisce la coppia <ip, ll_addr> nella tabella della interfaccia "iface" con il flag "nud"

```
void neigh_remove(char *ll_addr, u_int32 ip, char *iface);
```

Rimuove la coppia <ip, ll_addr> dall'interfaccia "iface"

```
int ipneigh_modify(int cmd, int flags, int nud, char *ll_addr,
u_int32 ip, char *iface);
```

è la funzione che viene richiamata da neigh_add() e neigh_remove(). A seconda del comando "cmd" passatole (RTM_NEWNEIGH o RTM_DELNEIGH) effettua l'inserimento o la cancellazione della coppia <ip, ll_addr>.
I "flags" per l'inserimento sono NLM_F_CREATE | NLM_F_REPLACE che istruiscono la funzione a creare una nuova entry se non era già presente e a sostituirla in caso contrario.

Questa funzione prepara un messaggio RTNL di questo tipo :

```
struct rtnl_handle rth;
struct {
   struct nlmsghdr   n;
   struct ndmsg      ndm;
   char              buf[256];
} req;

inet_prefix dst;

memset(&req, 0, sizeof(req));

req.n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ndmsg));
req.n.nlmsg_flags = NLM_F_REQUEST | flags;
```

```
req.n.nlmsg_type = cmd;
req.ndm.ndm_family = AF_INET;

get_addr(&dst, inet_ntoa(*(struct in_addr *)&ip), AF_INET);
addattr_l(&req.n, sizeof(req), NDA_DST, &dst.data, dst.bytelen);
addattr_l(&req.n, sizeof(req), NDA_LLADDR, ll_addr, LL_ADDR_LEN);
```

e lo trasmette al kernel attraverso una socket netlink.

```
if (rtnl_open(&rth, 0) < 0)
   ERROR_MSG("rtnl_open()");

if (rtnl_talk(&rth, &req.n, 0, 0, NULL, NULL, NULL) < 0)
   ERROR_MSG("rtnl_talk()");
```

6.4.5 Gestione delle chiavi

6.4.5.1 Negli host

Quando un host deve controllare l'autenticità di un messaggio, ha bisogno di avere la chiave pubblica del mittente per poter controllare la firma. Le chiavi sono distribuite dall'AKD e vengono messe in una cache per minimizzare le richieste verso l'AKD stesso. Infatti se la chiave non è cambiata non è necessario richiederla ad ogni scambio di messaggi, è sufficiente utilizzare quella contenuta nella cache.

La cache delle chiavi viene manipolata attraverso le funzioni:

```
int key_add( u_int32 ip_addr, DSA *key);
int get_key( u_int32 ip_addr, DSA **key);
void key_request( u_int32 ip_addr);
```

le quali rispettivamente aggiungono una chiave alla cache, recuperano la chiave dalla cache ed effettuano una richiesta all'AKD per ricevere la chiave relativa all'IP specificato.

La cache è implementata mediante una hash table di 2^5 elementi. La dichiarazione della hash table e dei suoi elementi è la seguente.

```
SLIST_HEAD (, keys_entry) keys_head[HASH_SIZE];

struct keys_entry {
   u_int32 ip_addr;
   DSA *key;
   SLIST_ENTRY (keys_entry) next;
};
```

La funzione di hash utilizzata è la FNV [20] e le collisioni sono risolte tramite chaining a liste.

Oltre alla cache delle chiavi per gli host, il demone memorizza in una apposita lista le chiavi pubbliche degli AKD a cui è connesso (uno per ogni interfaccia).

```
SLIST_HEAD (, ca_entry) ca_head;

struct ca_entry {
   u_char *iface;
   u_int32 ip_addr;
   u_char ll_addr[LL_ADDR_LEN];
   DSA *key;
   int ca_time_delta;   /* used for time sincronization */
   SLIST_ENTRY (ca_entry) next;
};
```

Ogni elemento della lista contiene l'IP, il MAC address, la chiave pubblica e il delta relativo all'AKD di una determinata interfaccia di rete.

6.4.5.2 Nell'AKD

L'AKD utilizza la cache delle chiavi per memorizzare le chiavi di tutti gli host della rete. Questa tabella è riempita durante la fase di startup caricando da file le chiavi pubbliche degli host.
In questo caso non saranno effettuate richieste se la chiave non è presente nel database ma si considererà l'host non facente parte del protocollo SARP.

La lista delle chiavi degli AKD resterà vuota poiché l'host stesso è l'AKD.

6.4.6 Firma digitale

Le firme crittografiche utilizzate dal protocollo SARP si appoggiano a chiavi DSA. Per implementare le routine di firma e verifica sono state utilizzate le librerie OpenSSL.

Durante la fase di startup (crypto_init), il demone inizializza le librerie crittografiche fornendo al PRNG 1024 byte del file /dev/urandom. La funzione che effettua questa operazione è

```
RAND_load_file("/dev/urandom", 1024);
```

sempre durante questa fase, vengono anche precalcolati i parametri DSA per le firma attraverso la chiamata crypto_precomp() che effettua una :

```
DSA_sign_setup(GBL_CRYPTO_KEY, NULL,
               &GBL_CRYPTO_KEY->kinv,
               &GBL_CRYPTO_KEY->r);
```

Questa operazione vale per una sola firma. La funzione crypto_precomp() deve essere nuovamente eseguita al termine di una operazione di firma. poiché questa operazione richiede alcuni millisecondi, il demone è programmato per effettuare questo calcolo solo dopo che il pacchetto è stato processato e spedito sulla rete. In pratica la crypto_precomp() viene eseguita appena prima che il demone torni nello stato di idle.

Le API per aggiungere o verificare una firma DSA sono fornite dalle funzioni:

```
void crypto_sign(u_char *data, int datalen, u_char *sign,
                 u_int32 *siglen);
int crypto_verify_sign(u_char *data, int datalen,
                 u_char *sig, int siglen, DSA *key);
```

Queste implementano gli algoritmi di hash del messaggio e firma DSA dell'hash SHA-1.

```
u_char sha[SHA_DIGEST_LENGTH];
```

```
SHA1(data, datalen, sha);

DSA_sign(0, sha, SHA_DIGEST_LENGTH, sign, siglen,
         (DSA *)GBL_CRYPTO_KEY)
```

6.4.7 La coda di processing

Il demone, per poter controllare la validità di una firma, deve possedere la chiave pubblica del mittente. In caso la chiave non sia presente nella propria cache, deve essere inoltrata una richiesta all'AKD. In questo caso il demone non può essere bloccante sulla risposta dell'AKD poiché altre SARP request/reply potrebbero arrivare nel frattempo. Il messaggio pendente viene quindi messo in una lista di processing e il demone torna in stato di idle in attesa di nuovi pacchetti. In questo stato è in grado di gestire qualsiasi tipo di messaggio.
All'arrivo di un messaggio di tipo SARPOP_KEY_REPLY la chiave viene inserita nella cache e la coda di processing viene analizzata alla ricerca di messaggi dipendenti dalla chiave appena inserita.

La lista è così definita:

```
SLIST_HEAD (, queue_entry) queue_head;

struct queue_entry {
   u_char *data;
   time_t timestamp;    /* used for timouted entries removal */
   SLIST_ENTRY (queue_entry) next;
};
```

Il campo `data` contiene il pacchetto così come è stato ricevuto dalla callback di capture. In questo modo potrà essere analizzato con la stessa funzione di parsing una volta ricevuta la chiave relativa. Il campo `timestamp` contiene il timestamp (in secondi) del momento in cui il pacchetto è stato inserito nella coda di processing. Periodicamente i messaggi troppo vecchi e non ancora processati vengono cancellati dalla lista per impedire "memory leak".

Capitolo 7

Valutazione delle prestazioni

In questo capitolo vengono presentate le misurazioni effettuate sul protocollo Secure ARP. È stato misurato l'overhead introdotto durante il primo scambio di messaggi tra due host, comparandolo al tempo impiegato per compiere la medesima operazione utilizzando il protocollo ARP originale. I test sono stati suddivisi in categorie per meglio analizzare il problema ed individuare i passaggi che richiedono più tempo di calcolo. Saranno presentai anche test specifici per le misurazioni di calcolo e verifica della firma DSA.

7.1 Introduzione

Progettando protocolli di sicurezza quali Secure ARP, è necessario avere sempre un occhio di riguardo per le prestazioni. È necessario trovare il giusto compromesso tra numero di controlli e perdita di prestazioni.
Il protocollo Secure ARP prevede un massiccio uso di crittografia a chiave pubblica e questo impatta notevolmente sulle prestazioni. Tuttavia, poiché il traffico ARP non è così frequente come quello generato da altri protocolli di livello più alto, un ritardo sui messaggi ARP risulta accettabile anche nell'ordine dei decimi di secondo. Le entry nelle cache ARP hanno infatti un timeout di 20 minuti circa, questo comporta quindi che si andrebbero a perdere alcuni decimi di secondo ogni 20 minuti, che risultano trascurabili. È comunque necessario limitare il tempo di risposta di messaggi Secure ARP poiché la prima volta che due host comunicano impiegheranno il tempo maggiore per elaborare i messaggi scambiati. Infatti nessuno dei due host

possiede la chiave pubblica dell'altro ed entrambe devono essere richieste all'AKD introducendo ulteriore overhead. Questo primo scambio non deve comunque richiedere troppo tempo per non scoraggiare l'utente dall'usare il protocollo.
Per misurare le prestazioni del protocollo ARP è possibile misurare il tempo che intercorre tra l'invio di un ICMP echo request e la ricezione della risposta. Il protocollo ICMP è quello a priorità maggiore quindi gli overhead del kernel possono essere trascurati. In ogni caso, se tutte le misurazioni tengono conto di questo ritardo, i risultati sono comparabili. Vedremo in seguito in dettaglio come questo test viene effettuato.
Confrontando i tempi ottenuti utilizzando ARP con quelli utilizzando SARP possiamo renderci conto dell'overhead determinato dai controlli aggiuntivi di SARP.

Un altro test effettuato è stato quello di misurare il tempo che la funzione di firma DSA impiega a calcolare la firma digitale. Questo test misura il tempo impiegato dalla funzione di hash (quasi trascurabile) e quello ben più rilevante per la verifica (o il calcolo) della firma. In seguito sono riportati i test che riassumono questi risultati.

7.1.1 Ambienete sperimentale

I test sono stati effettuati su una rete di 3 host collegati tramite un HUB Ethernet 10 Mb/s. L'AKD è un AMD Athlon 4 1.0 GHz con 256 Mb RAM e scheda di rete 10/100 Mb/s; i due host utilizzati per le comunicazioni sono Intel Pentium IV 1.6 GHz con 128 Mb RAM e schede di rete 10/100 Mb/s. Il kernel montato su tutte le macchine è Linux 2.4.18. L'AKD utilizza Gentoo Linux 1.4 mentre gli host sono Debian Linux 3.0.

Tutti i test sono stati svolti con le tabelle ARP inizialmente vuote, altrimenti nessuna richiesta ARP sarebbe stata effettuata. Questo può essere facilmente ottenuto tramite il comando:

```
ip neigh flush dev eth0
```

7.2 Misurazioni con ICMP

Per effettuare il test tramite un ICMP echo request è necessario che le tabelle di ARP siano vuote. In questo modo è possibile misurare il tempo necessario al sistema per risolvere la coppia <IP, MAC address>.
Una volta eseguito il comando `ping hostname` il kernel inizia la procedura di risoluzione ARP. Il comando ping ripete n volte la richiesta di un ICMP echo e misura il tempo di round trip in base all'istante in cui riceve la risposta. I passi compiuti sono i seguenti:

- l'utente dalla macchina 10.0.0.1 digita il comando `ping 10.0.0.2`
- il sistema di routing calcola che si tratta di un indirizzo locale e richiede al kernel di risolvere l'indirizzo.
- il kernel di 10.0.0.1 effettua una richiesta ARP verso l'indirizzo 10.0.0.2
- 10.0.0.2 riceve la richiesta ARP
- il kernel di 10.0.0.2 memorizza la coppia <10.0.0.1, 01:01:01:01:01:01> nella sua cache ARP
- 10.0.0.2 risponde inviando il MAC address 02:02:02:02:02:02
- il kernel di 10.0.0.1 memorizza la coppia <10.0.0.2, 02:02:02:02:02:02> nella sua cache ARP
- 10.0.0.1 può ora mandare un ICMP echo request a 10.0.0.2
- 10.0.0.2 risponde con un ICMP echo reply a 10.0.0.1

Tutte queste operazioni sono misurate dal comando `ping` che visualizza il tempo di Round Trip. I successivi messaggi di ICMP echo request sono spediti direttamente poiché la ARP cache è già popolata con l'informazione necessaria.

```
> ping 10.0.0.2

PING 10.0.0.2 (10.0.0.2): 56 octets data
64 octets from 10.0.0.2: icmp_seq=0 ttl=255 time=0.8 ms
64 octets from 10.0.0.2: icmp_seq=1 ttl=255 time=0.4 ms
64 octets from 10.0.0.2: icmp_seq=2 ttl=255 time=0.4 ms
```

Si nota come la prima risposta richieda un tempo superiore dovuto alla risoluzione degli indirizzi di rete. Il ritardo dovuto alla risoluzione degli indirizzi è dovuto al fatto che devono essere scambiati 2 messaggi in più rispetto al normale ICMP echo request/reply, cioè i due messaggi ARP. Infatti supponendo un tempo di 0.2 ms per messaggio, i conti sono presto fatti: 0.4 ms per due messaggi e 0.8 ms per 4 messaggi. Il tempo misurato quindi è quello necessario ai frame Ethernet per viaggiare da un host all'altro. Nessun calcolo deve essere effettuato dal kernel.

Il comando ping è stato eseguito 1000 volte, svuotando la cache ARP prima di ogni esecuzione. Come si vede dai valori riportati nella tabella 7.1, il tempo richiesto dal primo ICMP ha una maggiore variabilità del tempo richiesto dai successivi. Questo perché il primo ICMP provoca lo scambio di 2 messaggi ARP.

	primo ICMP	**successivi**
massimo	**0.8**	**0.5**
minimo	**0.6**	**0.4**
media	**0.70**	**0.46**
deviazione	**0.050**	**0.047**

Tabella 7.1: Risultati con ICMP

Dai risultati dei test effettuati risulta che l'overhead medio introdotto dalla risoluzione ARP è di 0.27 ms.

7.3 Misurazioni con protocollo Secure ARP

Il protocollo Secure ARP introduce un notevole overhead dovuto all'uso di crittografia a chiave pubblica. Le operazioni effettuate nello scenario descritto al paragrafo precedente aumentano sensibilmente.

I test effettuati si dividono in due gruppi: senza le chiavi pubbliche in cache e con le chiavi precedentemente memorizzate nella cache delle chiavi. Il primo scenario è il meno comune e si verifica solo se due host non hanno mai comunicato dal'avvio del demone SARP fino a quel momento. In questo caso infatti gli host non conoscono la chiave pubblica dell'interlocutore e dovranno chiederla all'AKD durante la fase di autenticazione. Questo scenario si può presentare anche nel caso in cui uno dei due host abbia cambiato la

propria chiave pubblica. In questo caso l'host che riscontra una discrepanza tra firma e chiave pubblica, chiederà all'AKD la nuova chiave pubblica del suo interlocutore.

Nel secondo scenario le chiavi si trovano già nella cache del demone SARP. Ciò si verifica nel caso in cui due host abbiano comunicato precedentemente almeno una volta. In questo caso i ritardi saranno dovuti solo alla fase di autenticazione, poiché le chiavi non devono essere richieste all'AKD.

7.3.1 Chiavi non in cache

Nel caso in cui due host non abbiano mai comunicato in precedenza, la procedura di autenticazione è molto costosa in termini di tempo. Infatti i passi compiuti da due host che vogliano scambiarsi un messaggio ICMP echo sono i seguenti:

1 l'utente dalla macchina 10.0.0.1 esegue il comando `ping 10.0.0.2` ;

2 il kernel di 10.0.0.1 effettua una richiesta ARP verso l'indirizzo 10.0.0.2;

3 10.0.0.2 riceve la richiesta ARP;

4 10.0.0.2 risponde inviando il MAC address 02:02:02:02:02:02 all'interno di una risposta SARP;

5 il kernel di 10.0.0.1 riceve la risposta SARP e cerca la chiave nella propria cache;

6 il demone S-ARP di 10.0.0.1 manda una KEY_REQUEST per 10.0.0.2 all'AKD;

7 l'AKD risponde con un KEY_REPLY contenente la chiave di 10.0.0.2;

8 10.0.0.1 controlla la firma dell'AKD;

9 10.0.0.1 verifica la validità della firma e memorizza la coppia <10.0.0.2, 02:02:02:02:02:02>;

10 10.0.0.1 può ora mandare un ICMP echo request a 10.0.0.2;

11 il kernel di 10.0.0.2 effettua una richiesta ARP verso l'indirizzo 10.0.0.1;

12 10.0.0.1 riceve la richiesta ARP;

13 10.0.0.1 risponde inviando il MAC address 01:01:01:01:01:01 all'interno di una risposta SARP;

14 il kernel di 10.0.0.2 riceve la risposta SARP e cerca la chiave nella propria cache;

15 il demone S-ARP di 10.0.0.2 manda una KEY_REQUEST per 10.0.0.1 all'AKD;

16 l'AKD risponde con un KEY_REPLY contenente la chiave di 10.0.0.1;

17 10.0.0.2 controlla la firma dell'AKD;

18 10.0.0.2 verifica la validità della firma e memorizza la coppia <10.0.0.1, 01:01:01:01:01:01>;

19 10.0.0.2 può ora mandare un ICMP echo reply a 10.0.0.1;

Le operazioni che costano di più in termini di tempo sono le 4 verifiche delle firme DSA. Per il calcolo delle firme il tempo impiegato è trascurabile visto che i parametri sono precalcolati come spiegato nel paragrafo 5.2.1.3 e dimostrato dai test nel paragrafo 7.4.

Sono stati effettuati 20 test con chiavi DSA a 512 e a 1024 bit. I risultati sono in riportati nella seguente tabella.

	Chiave a 512 bit	**Chiave a 1024 bit**
massimo	18.1	48.8
minimo	17.7	48.0
media	17.86	48.49
deviazione	0.12	0.22

Tabella 7.2: Chiavi non presenti in cache

7.3.2 Chiavi già presenti in cache

La fase di autenticazione in questo caso è più snella. Infatti, non dovendo gli host richiedere la chiave all'AKD, le operazioni 6, 7, 8, 15, 16, 17 descritte nel paragrafo precedente non vengono effettuate. Questo comporta un risparmio di 2 operazioni di firma e 2 di verifica.

	Chiave a 512 bit	**Chiave a 1024 bit**
massimo	9.3	24.4
minimo	8.8	23.6
media	8.96	24.00
deviazione	0.13	0.20

Tabella 7.3: Chiavi già presenti in cache

I tempi misurati in questo caso e riportati in tabella 7.3 con chiavi da 512 e 1024 bit sono quelli che ci si può aspettare in media dal protocollo Secure ARP. Infatti una volta richieste le chiavi all'AKD e memorizzate in cache, tutte le successive comunicazioni beneficeranno dell'utilizzo di tale cache.

7.4 Tempi di firma e verifica SHA1 + DSA

I ritardi introdotti dal protocollo Secure ARP sono dovuti principalmente alla verifica delle firme DSA. I tempi di calcolo della firma e di verifica sono stati misurati separatamente.

```
dsa = DSA_generate_parameters(atoi(argv[1]), NULL, 0,
         &counter, &h, NULL, NULL);
DSA_generate_key(dsa);

for (i = 0; i < 50; i++) {

   DEBUG_TIME_INIT();
   DSA_sign_setup(dsa, NULL, &dsa->kinv, &dsa->r);
   DEBUG_TIME_FINI();

   usleep(100000);

   for(h=0; h<sizeof(data); h++)
      data[h] = (char)rand();

   DEBUG_TIME_INIT();
   SHA1(data, sizeof(data), sha);
   DSA_sign(0, sha, SHA_DIGEST_LENGTH, sign, &siglen, dsa);
```

```
   DEBUG_TIME_FINI();

   usleep(100000);

   DEBUG_TIME_INIT();
   SHA1(data, sizeof(data), sha);
   DSA_verify(0, sha, SHA_DIGEST_LENGTH, sign, siglen, dsa);
   DEBUG_TIME_FINI();
}
```

La misurazione dei tempi viene effettuata dalle macro DEBUG_TIME_INIT() DEBUG_TIME_FINI(). La prima inizializza un contatore tramite la gettimeofday() e la seconda calcola la differenza in microsecondi del tempo trascorso e memorizza i valori che saranno poi stampati in seguito.

```
#define DEBUG_TIME_INIT() gettimeofday(&init_time, 0)
#define DEBUG_TIME_PRINT(f) do { \
   gettimeofday(&fini_time, 0); \
   printf("%ld\n", (fini_time.tv_usec - init_time.tv_usec) ); \
} while (0)
```

Una volta calcolati i parametri DSA, l'operazione di firma si riduce a circa 36 microsecondi sia per chiavi a 512 che 1024 bit. Questo dipende dal fatto che la firma non è dipendente nè dal messaggio, nè dalla lunghezza della chiave. Non bisogna includere il tempo necessario al pre-calcolo nelle funzioni di misurazione della firma poiché si presuppone che la `DSA_sign_setup()` venga eseguita quando il demone è in stato di idle.

Il buffer "data" occupa lo spazio di memoria equivalente alla massima lunghezza di un pacchetto Secure ARP e contiene dati random. In questo modo simuliamo il caso peggiore in dimensione. La funzione di hash SHA-1 è comunque talmente veloce da non permettere di rilevare differenze apprezzabili.

La usleep(100000) è stata introdotta per impedire allo scheduler di interferire con le misurazioni. Infatti introducendo un delay tra le operazioni di firma, il quanto di tempo dedicato al programma di test non scade durante la computazione. Le misurazioni risultano così più omogenee.

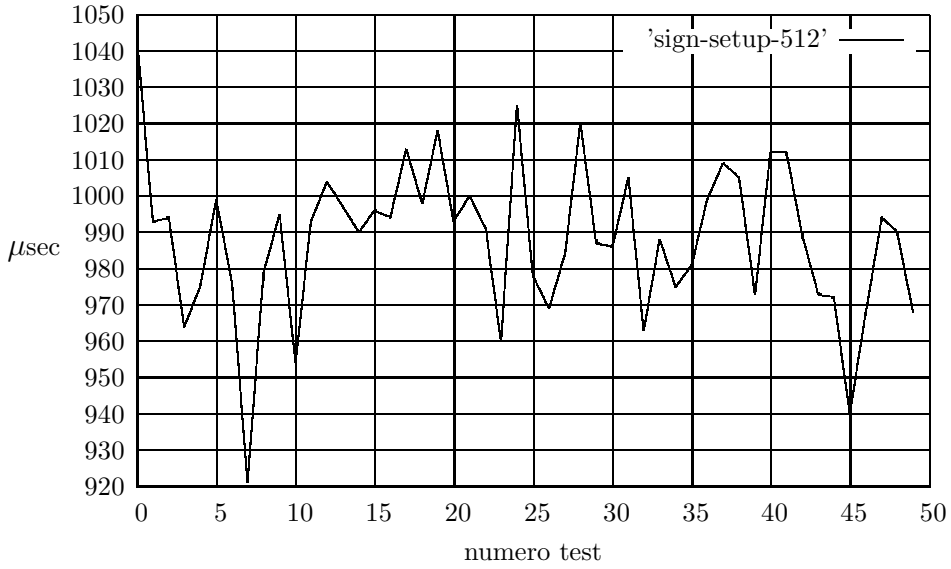

Figura 7.1: Calcolo dei parametri indipendenti di DSA (chiave 512 bit)

Le figure 7.3 e 7.4 evidenziano la differenza dei tempi di verifica utilizzando chiavi a 512 e a 1024 bit. Questo test risulta cruciale per avere un'idea dell'overhead introdotto dal protocollo Secure ARP. Infatti l'operazione di verifica e' la più lenta. Maggiore è il tempo impiegato da questo test, maggiore sarà il tempo misurato al primo ping durante lo scambio di messaggi SARP.

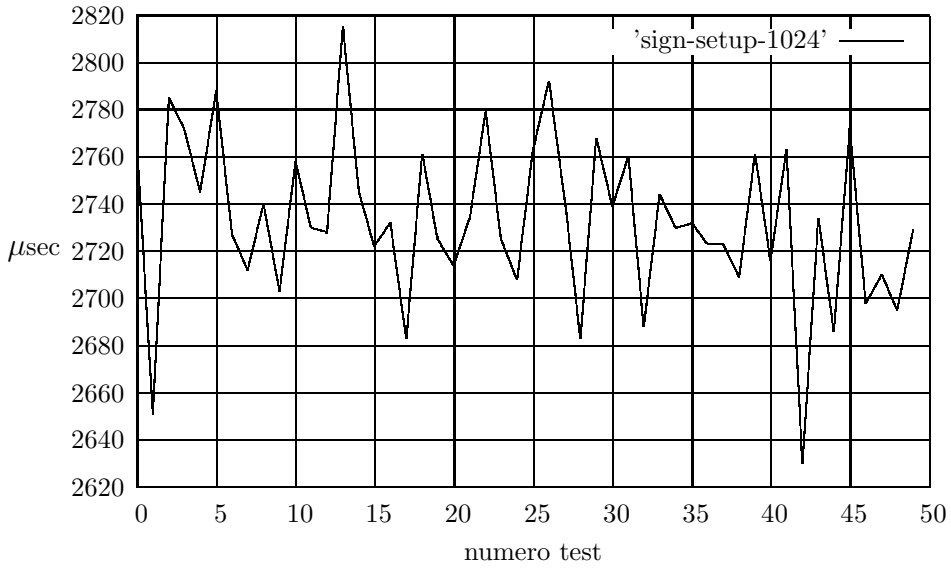

Figura 7.2: Calcolo dei parametri indipendenti di DSA (chiave 1024 bit)

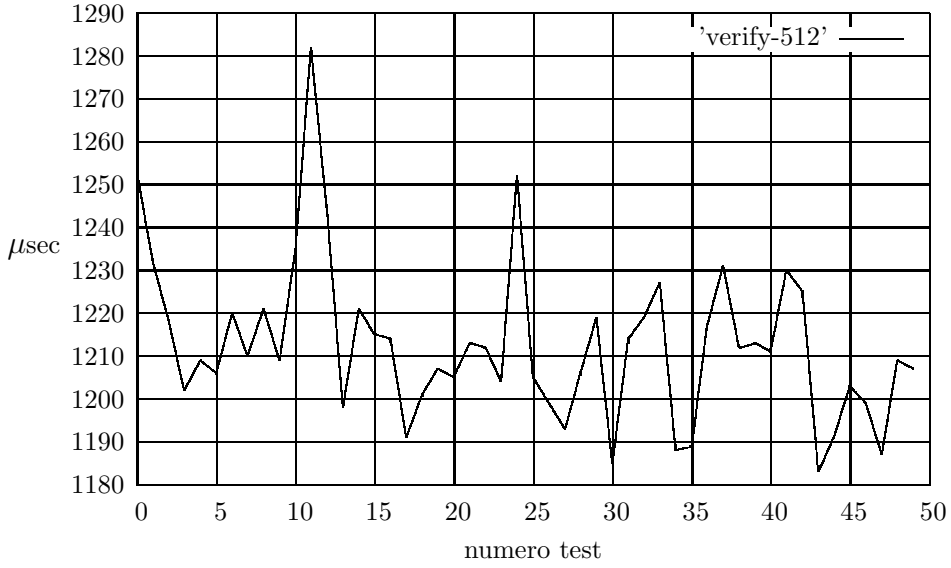

Figura 7.3: Test di verifica firma DSA (512 bit)

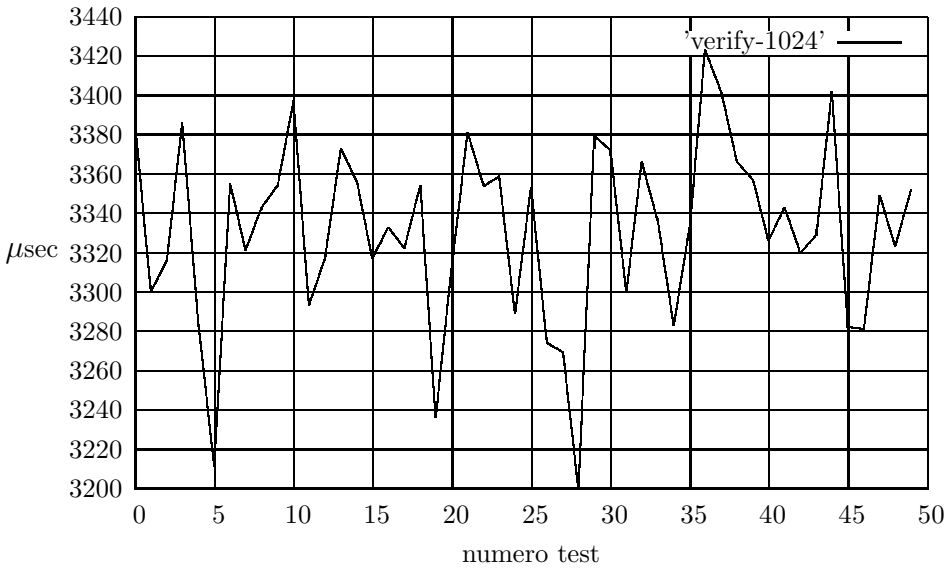

Figura 7.4: Test di verifica firma DSA (1024 bit)

Capitolo 8

Conclusioni e sviluppi futuri

In questa tesi è stato realizzato un protocollo di risoluzione degli indirizzi a livello 2 sicuro. Secure ARP è stato realizzato come una estensione del protocollo ARP già esistente. Il suo principale obiettivo è quello di impedire attacchi di tipo ARP poisoning. Attraverso l'uso dell'autenticazione dei mittenti dei messaggi ARP questo scopo è stato raggiunto.
Il protocollo è stato sviluppato per operare sia in ambito di configurazioni statiche degli indirizzi, sia in reti dove questi vengono assegnati dinamicamente da un server DHCP. Per effettuare quest'ultima operazione è necessario che i client e il server DHCP siano opportunamente modificati per supportare il protocollo Secure ARP.

L'implementazione del protocollo è richiede la presenza di un host nel dominio di broadcast della rete che assuma il ruolo di Authoritative Key Distributor. Questo comporta l'inserimento di un "single point of failure" nella rete locale. Per risolvere questo problema si potrebbe mettere un secondo host in hot-swap che si sostituisca al primo in caso di guasto di quest'ultimo. Basterebbe replicare il database delle chiavi su entrambi gli host ed attivare il demone "slave" quando il "master" non rispondesse più agli heartbeat.

Il bisogno di avere un AKD è dettato dalla necessità di avere un certificatore per le chiavi pubbliche degli host della rete. La scelta di utilizzare un meccanismo di autenticazione a chiave pubblica DSA è stata arbitraria. Un qualsiasi altro meccanismo di autenticazione forte dei soggetti può essere utilizzato per ottenere lo stesso risultato.

Nella configurazione e nella messa in opera di una rete che utilizza il protocollo Secure ARP è fondamentale il ruolo del sysadmin. Egli, infatti,

rappresenta il canale di comunicazione sicuro attraverso il quale inserire le chiavi pubbliche nel database dell'AKD. Ogni host della rete deve generare una coppia di chiavi pubblica/privata e consegnare la parte pubblica all'amministratore di rete il quale la inserirà nel database dell'AKD.

La distribuzione delle chiavi pubbliche in maniera affidabile attraverso un canale non sicuro è un problema ancora aperto che offre molti spunti di ricerca. Una possibile estensione a questo protocollo potrebbe essere rappresentato dall'implementazione di un sistema di scambio di chiavi pubbliche senza dover ricorrere allo scambio fisico tramite l'amministratore di rete. Un altro filone di ricerca è rappresentato dalla possibilità o meno di eliminare l'AKD. Questo infatti comporterebbe l'eliminazione di un palese "single point of failure" nell'architettura della rete, aumentando la robustezza della stessa.

I tempi di overhead misurati sull'implementazione realizzata durante la tesi, sono risultati accettabili. Infatti questi ritardi sono dell'ordine di qualche decina di millesimi di secondo. I test effettuati sono stati misurati su macchine di fascia medio-alta (Athlon 1.0 GHz), l'uso massiccio della crittografia a chiave pubblica potrebbe appesantire maggiormente macchine un po' datate. Si calcola comunque che il ritardo possa rimanere accettabile poiché questo si verifica solo la prima volta che due host comunicano. Dopo la prima comunicazione i risultati della richiesta sono inseriti in una cache di sistema che prolunga i tempi di validità della risposta ARP.

Questi problemi di calcolo crittografico potrebbero essere risolti in futuro quando i calcolatori saranno provvisti di un chip crittografico di supporto al processore. Così come il coprocessore matematico è stato introdotto negli anni 80 per accelerare l'esecuzione di operazioni in virgola mobile, si può immaginare che in un futuro non troppo lontano venga introdotto un coprocessore crittografico. Questo diminuirebbe di molto il carico della cpu e permetterebbe anche l'integrazione di protocolli che utilizzano pesantemente la crittografia (come ad esempio SARP) all'interno di apparati di rete dedicati. Si potrebbe così implementare in hardware l'AKD all'interno di gateway, router o firewall e munire le schede di rete di un chip crittografico contenente già chiave pubblica e privata (come se fosse una smart card) in modo da rendere l'autenticazione trasparente al sistema operativo.

Appendice A

Il demone SARPD

A.1 Librerie richieste

La compilazione del demone richiede che nel sistema siano installate le seguenti librerie:

- libpcap 0.7.1 o superiori
- libnet 1.1.0 o superiori
- libnetlink 2.4.7 o superiori
- libcrypto 0.9.6 o superiori

La compilazione è automatica tramite i comandi:

```
./configure
make
make install
```

che provvedono a compilare e installare il demone. Per una descrizione più completa del processo di configurazione e compilazione il file INSTALL contiene tutte le informazioni a riguardo.

A.2 Come usare il demone

Il demone può essere attivato o deattivato in qualsiasi momento. Ovviamente il modulo kernel deve essere già attivo prima dell'inizializzazione del demone stesso. Questo perchè il demone cerca di scrivere 1 nel file /proc/sys/net/ipv4/sarp per attivare il protocollo SARP. Se il modulo kernel non fosse già attivo, questa operazione fallirebbe. è consigliato che il demone venga lanciato all'avvio del sistema e resti attivo per tutto l'uptime dello stesso.

Durante la fase di avvio il demone cancella tutte le tuple presenti della cache di sistema tranne quelle marcate come STATIC. Con questa operazione si cancellano eventuali tracce di attacchi subiti prima dell'entrata in funzione del demone.

Il demone carica le informazioni relative alle chiavi attraverso il parsing del suo file di configurazione. Questa operazione viene effettuata solo una volta. Per permettere al demone di rileggere la configurazione a runtime (se per esempio si è cambiata qualche chiave) è necessario inviare un segnale di HUP al demone tramite il comando :

```
kill -HUP ‘pidof sarpd‘
```

A.3 Demone per l'AKD

Il protocollo SARP prevede che un host sia eletto ad Authoritative Key Distributor. Per utilizzare il demone in modalità AKD è necessario usare il seguente comando:

```
sarpd -akd-mode
```

Il demone lanciato in modalità AKD svolge sia le normali funzioni di host sia quelle di AKD, non è quindi necessario lanciare due demoni sulla macchina AKD.

Il demone è stato programmato per ascoltare su tutte le interfacce di rete. Una sola istanza è sufficiente per permettere di gestire tutte le schede ethernet del sistema.
L'unica limitazione in questo senso è che un host con più di una interfaccia

di rete, deve essere connesso solo a reti Secure ARP. Non è possibile abilitare SARP solo su una interfaccia.

A.4 Distribuzione delle chiavi

Le chiavi pubbliche in formato .sarp generate dagli host devono essere copiate nella directory delle chiavi dell'AKD e la chiave pubblica dell'AKD installata su ciascun host.

A.5 File di configurazione

Il file di configurazione del demone sarp è utilizzato per reperire le informazioni necessarie al suo funzionamento. All'interno di questo file vengono specificati i nomi assoluti dei file delle chiavi.

Ogni riga è organizzata secondo un formato standard:

```
Parametro[interfaccia]:  valore
```

che associa il "valore" al "parametro" sull'interfaccia di rete "interfaccia".

I parametri presenti nel file di configurazione sono :

```
CAKey[eth0]:  /etc/sarpd/id_sarp_ca.sarp
```

Che specifica il file contenente la chiave pubblica dell'AKD sull'interfaccia eth0.

```
MYKey[all]:  /etc/sarpd/id_sarp
```

Indica il file dove è memorizzata la chiave dell'host corrente.

```
DHCPKey[eth0]:  /etc/sarpd/id_sarp_dhcp.sarp
```

```
KEYDir[all]:  /etc/sarpd/hosts
```

Utilizzati solo se il demone viene lanciato con l'opzione –akd-mode, specificano rispettivamente la chiave pubblica del server DHCP e la directory dove sono memorizzate le chiavi degli host della rete.

Appendice B

Il comando SARP

Il comando `sarp` è una utility di supporto per l'amministratore della macchina. Con questo comando è possibile generare le chiavi DSA a 512 o a 1024 bit. Il comando può anche essere utilizzato per verificare che una coppia di chiavi pubblica / privata sia valida e per la generazione dei file di chiavi .sarp descritti nella sezione 5.3.2.3

Le opzioni disponibili per questo comando sono:

```
Usage: sarp [OPTIONS]

General options:
   -b, --bitlen               number of bit for the key
   -g, --genkey               generate a keypair
   -s, --sign                 generate a signed info file
   -o, --outfile <FILE>       specify the output filename
   -c, --check                check if the keypair is valid
   -i, --infile <FILE>        specify the input filename

   -v, --verbose              be verbose during operations
   -V, --version              prints the version and exit
   -h, --help                 this help screen
```

Esempi di utilizzo:

```
./sarp -gv -o prova
```

crea una coppia di file chiamati `prova` e `prova.pub`. Questi file contengono rispettivamente la chiave privata e la chiave pubblica generate. Il flag -v è utilizzato per stampare a video le operazioni effettuate.

```
./sarp -s -i prova
```

crea un file `prova.sarp` a partire da una coppia di file denominata `prova` e `prova.pub`. Il file è creato in formato sarp per poter essere memorizzato nel database dell'AKD. Saranno richiesti l'IP a cui associare la chiave pubblica e l'eventuale MAC address se si tratta della chiave pubblica da utilizzare per l'AKD.

```
./sarp -c -i prova
```

controlla che la coppia `prova` e `prova.pub` sia corretta.

```
./sarp -gvs -b 512
```

genera una tripla di file `id_sarp`, `id_sarp.pub` e `id_sarp.sarp`. Rispettivamente chiave privata, pubblica e certificato sarp. Le chiavi sono generate di 512 bit (come specificato dall'opzione -b).

Appendice C

Licenza per Documentazione Libera GNU

Version 1.1, Marzo 2000

Preambolo

Lo scopo di questa licenza è di rendere un manuale, un testo o altri documenti scritti "liberi" nel senso di assicurare a tutti la libertà effettiva di copiarli e redistribuirli, con o senza modifiche, a fini di lucro o no. In secondo luogo questa licenza prevede per autori ed editori il modo per ottenere il giusto riconoscimento del proprio lavoro, preservandoli dall'essere considerati responsabili per modifiche apportate da altri.

Questa licenza è un "copyleft": ciò vuol dire che i lavori che derivano dal documento originale devono essere ugualmente liberi. è il complemento alla Licenza Pubblica Generale GNU, che è una licenza di tipo "copyleft" pensata per il software libero.

Abbiamo progettato questa licenza al fine di applicarla alla documentazione del software libero, perché il software libero ha bisogno di documentazione

libera: un programma libero dovrebbe accompagnarsi a manuali che forniscano la stessa libertà del software. Ma questa licenza non è limitata alla documentazione del software; può essere utilizzata per ogni testo che tratti un qualsiasi argomento e al di là dell'avvenuta pubblicazione cartacea. Raccomandiamo principalmente questa licenza per opere che abbiano fini didattici o per manuali di consultazione.

C.1 Applicabilità e Definizioni

Questa licenza si applica a qualsiasi manuale o altra opera che contenga una nota messa dal detentore del copyright che dica che si può distribuire nei termini di questa licenza. Con "Documento", in seguito ci si riferisce a qualsiasi manuale o opera. Ogni fruitore è un destinatario della licenza e viene indicato con "voi".

Una "versione modificata" di un documento è ogni opera contenente il documento stesso o parte di esso, sia riprodotto alla lettera che con modifiche, oppure traduzioni in un'altra lingua.

Una "sezione secondaria" è un'appendice cui si fa riferimento o una premessa del documento e riguarda esclusivamente il rapporto dell'editore o dell'autore del documento con l'argomento generale del documento stesso (o argomenti affini) e non contiene nulla che possa essere compreso nell'argomento principale. (Per esempio, se il documento è in parte un manuale di matematica, una sezione secondaria non può contenere spiegazioni di matematica). Il rapporto con l'argomento può essere un tema collegato storicamente con il soggetto principale o con soggetti affini, o essere costituito da argomentazioni legali, commerciali, filosofiche, etiche o politiche pertinenti.

Le "sezioni non modificabili" sono alcune sezioni secondarie i cui titoli sono esplicitamente dichiarati essere sezioni non modificabili, nella nota che indica che il documento è realizzato sotto questa licenza.

I "testi copertina" sono dei brevi brani di testo che sono elencati nella nota che indica che il documento è realizzato sotto questa licenza.

Una copia "trasparente" del documento indica una copia leggibile da un calcolatore, codificata in un formato le cui specifiche sono disponibili pub-

blicamente, i cui contenuti possono essere visti e modificati direttamente, ora e in futuro, con generici editor di testi o (per immagini composte da pixel) con generici editor di immagini o (per i disegni) con qualche editor di disegni ampiamente diffuso, e la copia deve essere adatta al trattamento per la formattazione o per la conversione in una varietà di formati atti alla successiva formattazione. Una copia fatta in un altro formato di file trasparente il cui markup è stato progettato per intralciare o scoraggiare modifiche future da parte dei lettori non è trasparente. Una copia che non è trasparente è "opaca".

Esempi di formati adatti per copie trasparenti sono l'ASCII puro senza markup, il formato di input per Texinfo, il formato di input per LaTeX, SGML o XML accoppiati ad una DTD pubblica e disponibile, e semplice HTML conforme agli standard e progettato per essere modificato manualmente. Formati opachi sono PostScript, PDF, formati proprietari che possono essere letti e modificati solo con word processor proprietari, SGML o XML per cui non è in genere disponibile la DTD o gli strumenti per il trattamento, e HTML generato automaticamente da qualche word processor per il solo output.

La "pagina del titolo" di un libro stampato indica la pagina del titolo stessa, più qualche pagina seguente per quanto necessario a contenere in modo leggibile, il materiale che la licenza prevede che compaia nella pagina del titolo. Per opere in formati in cui non sia contemplata esplicitamente la pagina del titolo, con "pagina del titolo" si intende il testo prossimo al titolo dell'opera, precedente l'inizio del corpo del testo.

C.2 Copie Letterali

Si può copiare e distribuire il documento con l'ausilio di qualsiasi mezzo, per fini di lucro e non, fornendo per tutte le copie questa licenza, le note sul copyright e l'avviso che questa licenza si applica al documento, e che non si aggiungono altre condizioni al di fuori di quelle della licenza stessa. Non si possono usare misure tecniche per impedire o controllare la lettura o la produzione di copie successive alle copie che si producono o distribuiscono. Però si possono ricavare compensi per le copie fornite. Se si distribuiscono

un numero sufficiente di copie si devono seguire anche le condizioni della sezione 3.

Si possono anche prestare copie e con le stesse condizioni sopra menzionate possono essere utilizzate in pubblico.

C.3 Copiare in notevoli quantità

Se si pubblicano a mezzo stampa più di 100 copie del documento, e la nota della licenza indica che esistono uno o più testi copertina, si devono includere nelle copie, in modo chiaro e leggibile, tutti i testi copertina indicati: il testo della prima di copertina in prima di copertina e il testo di quarta di copertina in quarta di copertina. Ambedue devono identificare l'editore che pubblica il documento. La prima di copertina deve presentare il titolo completo con tutte le parole che lo compongono egualmente visibili ed evidenti. Si può aggiungere altro materiale alle copertine. Il copiare con modifiche limitate alle sole copertine, purché si preservino il titolo e le altre condizioni viste in precedenza, è considerato alla stregua di copiare alla lettera.

Se il testo richiesto per le copertine è troppo voluminoso per essere riprodotto in modo leggibile, se ne può mettere una prima parte per quanto ragionevolmente può stare in copertina, e continuare nelle pagine immediatamente seguenti.

Se si pubblicano o distribuiscono copie opache del documento in numero superiore a 100, si deve anche includere una copia trasparente leggibile da un calcolatore per ogni copia o menzionare per ogni copia opaca un indirizzo di una rete di calcolatori pubblicamente accessibile in cui vi sia una copia trasparente completa del documento, spogliato di materiale aggiuntivo, e a cui si possa accedere anonimamente e gratuitamente per scaricare il documento usando i protocolli standard e pubblici generalmente usati. Se si adotta l'ultima opzione, si deve prestare la giusta attenzione, nel momento in cui si inizia la distribuzione in quantità elevata di copie opache, ad assicurarsi che la copia trasparente rimanga accessibile all'indirizzo stabilito fino ad almeno un anno di distanza dall'ultima distribuzione (direttamente o attraverso rivenditori) di quell'edizione al pubblico.

È caldamente consigliato, benché non obbligatorio, contattare l'autore del

documento prima di distribuirne un numero considerevole di copie, per metterlo in grado di fornire una versione aggiornata del documento.

C.4 Modifiche

Si possono copiare e distribuire versioni modificate del documento rispettando le condizioni delle precedenti sezioni 2 e 3, purché la versione modificata sia realizzata seguendo scrupolosamente questa stessa licenza, con la versione modificata che svolga il ruolo del "documento", così da estendere la licenza sulla distribuzione e la modifica a chiunque ne possieda una copia. Inoltre nelle versioni modificate si deve:

- A. Usare nella pagina del titolo (e nelle copertine se ce ne sono) un titolo diverso da quello del documento, e da quelli di versioni precedenti (che devono essere elencati nella sezione storia del documento ove presenti). Si può usare lo stesso titolo di una versione precedente se l'editore di quella versione originale ne ha dato il permesso.
- B. Elencare nella pagina del titolo, come autori, una o più persone o gruppi responsabili in qualità di autori delle modifiche nella versione modificata, insieme ad almeno cinque fra i principali autori del documento (tutti gli autori principali se sono meno di cinque).
- C. Dichiarare nella pagina del titolo il nome dell'editore della versione modificata in qualità di editore.
- D. Conservare tutte le note sul copyright del documento originale.
- E. Aggiungere un'appropriata licenza per le modifiche di seguito alle altre licenze sui copyright.
- F. Includere immediatamente dopo la nota di copyright, un avviso di licenza che dia pubblicamente il permesso di usare la versione modificata nei termini di questa licenza, nella forma mostrata nell'addendum alla fine di questo testo.
- G. Preservare in questo avviso di licenza l'intera lista di sezioni non modificabili e testi copertina richieste come previsto dalla licenza del documento.

- H. Includere una copia non modificata di questa licenza.
- I. Conservare la sezione intitolata "Storia", e il suo titolo, e aggiungere a questa un elemento che riporti al minimo il titolo, l'anno, i nuovi autori, e gli editori della versione modificata come figurano nella pagina del titolo. Se non ci sono sezioni intitolate "Storia" nel documento, createne una che riporti il titolo, gli autori, gli editori del documento come figurano nella pagina del titolo, quindi aggiungete un elemento che descriva la versione modificata come detto in precedenza.
- J. Conservare l'indirizzo in rete riportato nel documento, se c'è, al fine del pubblico accesso ad una copia trasparente, e possibilmente l'indirizzo in rete per le precedenti versioni su cui ci si è basati. Questi possono essere collocati nella sezione "Storia". Si può omettere un indirizzo di rete per un'opera pubblicata almeno quattro anni prima del documento stesso, o se l'originario editore della versione cui ci si riferisce ne dà il permesso.
- K. In ogni sezione di "Ringraziamenti" o "Dediche", si conservino il titolo, il senso, il tono della sezione stessa.
- L. Si conservino inalterate le sezioni non modificabili del documento, nei propri testi e nei propri titoli. I numeri della sezione o equivalenti non sono considerati parte del titolo della sezione.
- M. Si cancelli ogni sezione intitolata "Riconoscimenti". Solo questa sezione può non essere inclusa nella versione modificata.
- N. Non si modifichi il titolo di sezioni esistenti come "miglioria" o per creare confusione con i titoli di sezioni non modificabili.

Se la versione modificata comprende nuove sezioni di primaria importanza o appendici che ricadono in "sezioni secondarie", e non contengono materiale copiato dal documento, si ha facoltà di rendere non modificabili quante sezioni si voglia. Per fare ciò si aggiunga il loro titolo alla lista delle sezioni immutabili nella nota di copyright della versione modificata. Questi titoli devono essere diversi dai titoli di ogni altra sezione.

Si può aggiungere una sezione intitolata "Riconoscimenti", a patto che non contenga altro che le approvazioni alla versione modificata prodotte da

vari soggetti–per esempio, affermazioni di revisione o che il testo è stato approvato da una organizzazione come la definizione normativa di uno standard.

Si può aggiungere un brano fino a cinque parole come Testo Copertina, e un brano fino a 25 parole come Testo di Retro Copertina, alla fine dell'elenco dei Testi Copertina nella versione modificata. Solamente un brano del Testo Copertina e uno del Testo di Retro Copertina possono essere aggiunti (anche con adattamenti) da ciascuna persona o organizzazione. Se il documento include già un testo copertina per la stessa copertina, precedentemente aggiunto o adattato da voi o dalla stessa organizzazione nel nome della quale si agisce, non se ne può aggiungere un altro, ma si può rimpiazzare il vecchio ottenendo l'esplicita autorizzazione dall'editore precedente che aveva aggiunto il testo copertina.

L'autore/i e l'editore/i del "documento" non ottengono da questa licenza il permesso di usare i propri nomi per pubblicizzare la versione modificata o rivendicare l'approvazione di ogni versione modificata.

C.5 Unione di documenti

Si può unire il documento con altri realizzati sotto questa licenza, seguendo i termini definiti nella precedente sezione 4 per le versioni modificate, a patto che si includa l'insieme di tutte le Sezioni Invarianti di tutti i documenti originali, senza modifiche, e si elenchino tutte come Sezioni Invarianti della sintesi di documenti nella licenza della stessa.

Nella sintesi è necessaria una sola copia di questa licenza, e multiple sezioni invarianti possono essere rimpiazzate da una singola copia se identiche. Se ci sono multiple Sezioni Invarianti con lo stesso nome ma contenuti differenti, si renda unico il titolo di ciascuna sezione aggiungendovi alla fine e fra parentesi, il nome dell'autore o editore della sezione, se noti, o altrimenti un numero distintivo. Si facciano gli stessi aggiustamenti ai titoli delle sezioni nell'elenco delle Sezioni Invarianti nella nota di copiright della sintesi.

Nella sintesi si devono unire le varie sezioni intitolate "storia" nei vari documenti originali di partenza per formare una unica sezione intitolata "storia"; allo stesso modo si unisca ogni sezione intitolata "Ringraziamenti", e ogni

sezione intitolata "Dediche". Si devono eliminare tutte le sezioni intitolate "Riconoscimenti".

C.6 Raccolte di documenti

Si può produrre una raccolta che consista del documento e di altri realizzati sotto questa licenza; e rimpiazzare le singole copie di questa licenza nei vari documenti con una sola inclusa nella raccolta, solamente se si seguono le regole fissate da questa licenza per le copie alla lettera come se si applicassero a ciascun documento.

Si può estrarre un singolo documento da una raccolta e distribuirlo individualmente sotto questa licenza, solo se si inserisce una copia di questa licenza nel documento estratto e se si seguono tutte le altre regole fissate da questa licenza per le copie alla lettera del documento.

C.7 Raccogliere insieme a lavori indipendenti

Una raccolta del documento o sue derivazioni con altri documenti o lavori separati o indipendenti, all'interno di o a formare un archivio o un supporto per la distribuzione, non è una "versione modificata" del documento nella sua interezza, se non ci sono copiright per l'intera raccolta. Ciascuna raccolta si chiama allora "aggregato" e questa licenza non si applica agli altri lavori contenuti in essa che ne sono parte, per il solo fatto di essere raccolti insieme, qualora non siano però loro stessi lavori derivati dal documento.

Se le esigenze del Testo Copertina della sezione 3 sono applicabili a queste copie del documento allora, se il documento è inferiore ad un quarto dell'intero aggregato i Testi Copertina del documento possono essere piazzati in copertine che delimitano solo il documento all'interno dell'aggregato. Altrimenti devono apparire nella copertina dell'intero aggregato.

C.8 Traduzioni

La traduzione è considerata un tipo di modifica, e di conseguenza si possono distribuire traduzioni del documento seguendo i termini della sezione 4.

Rimpiazzare sezioni non modificabili con traduzioni richiede un particolare permesso da parte dei detentori del diritto d'autore, ma si possono includere traduzioni di una o più sezioni non modificabili in aggiunta alle versioni originali di queste sezioni immutabili. Si può fornire una traduzione della presente licenza a patto che si includa anche l'originale versione inglese di questa licenza. In caso di discordanza fra la traduzione e l'originale inglese di questa licenza la versione originale inglese prevale sempre.

C.9 Termini

Non si può applicare un'altra licenza al documento, copiarlo, modificarlo, o distribuirlo al di fuori dei termini espressamente previsti da questa licenza. Ogni altro tentativo di applicare un'altra licenza al documento, copiarlo, modificarlo, o distribuirlo è deprecato e pone fine automaticamente ai diritti previsti da questa licenza. Comunque, per quanti abbiano ricevuto copie o abbiano diritti coperti da questa licenza, essi non ne cessano se si rimane perfettamente coerenti con quanto previsto dalla stessa.

C.10 Revisioni future di questa licenza

La Free Software Foundation può pubblicare nuove, rivedute versioni della Licenza per Documentazione Libera GNU volta per volta. Qualche nuova versione potrebbe essere simile nello spirito alla versione attuale ma differire in dettagli per affrontare nuovi problemi e concetti. Si veda http://www.gnu.org/copyleft.

Ad ogni versione della licenza viene dato un numero che distingue la versione stessa. Se il documento specifica che si riferisce ad una versione particolare della licenza contraddistinta dal numero o "ogni versione successiva", si ha la possibilità di seguire termini e condizioni sia della versione specificata che di ogni versione successiva pubblicata (non come bozza) dalla Free Software Foundation. Se il documento non specifica un numero di versione particolare di questa licenza, si può scegliere ogni versione pubblicata (non come bozza) dalla Free Software Foundation.

ADDENDUM: Come usare questa licenza nei vostri documenti

Per applicare questa licenza ad un documento che si è scritto, si includa una copia della licenza nel documento e si inserisca il seguente avviso subito dopo la pagina del titolo:

> Copyright ©ANNO VOSTRO NOME. È garantito il permesso di copiare, distribuire e/o modificare questo documento seguendo i termini della Licenza per Documentazione Libera GNU, Versione 1.1 o ogni versione successiva pubblicata dalla Free Software Foundation; con le Sezioni Non Modificabili ELENCARNE I TITOLI, con i Testi Copertina ELENCO, e con i Testi di Retro Copertina ELENCO. Una copia della licenza è acclusa nella sezione intitolata "Licenza per Documentazione Libera GNU".

Se non ci sono Sezioni non Modificabili, si scriva "senza Sezioni non Modificabili" invece di dire quali sono non modificabili. Se non c'è Testo Copertina, si scriva "nessun Testo Copertina" invece di "il testo Copertina è ELENCO"; e allo stesso modo si operi per il Testo di Retro Copertina.

Se il vostro documento contiene esempi non banali di programma in codice sorgente si raccomanda di realizzare gli esempi contemporaneamente applicandovi anche una licenza di software libero di vostra scelta, come ad esempio la Licenza Pubblica Generale GNU, al fine di permetterne l'uso come software libero.

Elenco delle figure

Bibliografia

[1] Computer Emergency Rensponse Team. http://www.cert.org.

[2] University of Southern California. Transmission Control Protocol . (RFC0793), September 1981.

[3] IEEE 802.3 CSMA/CD. http://grouper.ieee.org/groups/802/3/.

[4] David C. Plummer. An Ethernet Address Resolution Protocol. (RFC0826), November 1982.

[5] Andrew S. Tanenbaum. Computer Networks. ISBN 88-7750-453-6.

[6] ISO 7498, Open System Interconnection Model . http://www.acm.org/sigcomm/standards/iso_stds/OSI_MODEL/.

[7] Tcpdump a network analizer. http://www.tcpdump.org.

[8] Dug Song. http://monkey.org/~dugsong/dsniff.

[9] Alberto Ornaghi e Marco Valleri. http://ettercap.sf.net.

[10] Wired Equivalent Privacy. http://grouper.ieee.org/groups/802/11/documents/documentarchives/1994_docs/1194249.doc.

[11] Extensible Authentication Protocol. http://www.ietf.org/html.charters/eap-charter.html.

[12] Marco Cesati e Daniel P. Bovet. Understanding the Linux kernel. ISBN 0-596-00002-2.

[13] Netlink Sockets Overview. http://qos.ittc.ukans.edu/netlink/html/netlink.html.

[14] Frank Hunleth. Secure link layer. http://www.cs.wustl.edu/ fhunleth/projects/projects.html.

[15] Secure Socket Layer. http://www.openssl.org.

[16] William Stallings. Criptography and Network Security. ISBN 0-13-869017-0.

[17] Vanstone Menesez, Qu. An efficient protocol for authenticated key management. http://cacr.math.uwaterloo.ca/articles/1998/corr98-05.ps.

[18] RSA security. http://www.rsasecurity.com.

[19] Federal Information Processing Standards Publication 186. Digital Signature Standard (DSS). http://www.itl.nist.gov/fipspubs/fip186.htm, 1994.

[20] Landon Curt Noll. http://www.isthe.com/chongo/.

Indice analitico

Questa tesi e' stata redatta utilizzando LaTeX e vim.
(c) Alberto Ornghi 3 febbraio 2003