marcomas@libero.it bluemailto:marcomas@libero.it e Gianluigi Spagnuolo
kirash@interfree.it bluemailto:kirash@interfree.it

Linux netfilter Hacking HOWTO

Rusty Russell e Harald Welte, mailing list netfilter@lists.samba.org $Revision: 580 $ $Date: 2004-06-03 22:17:07 +0200 (Thu, 03 Jun 2004) $

Indice

1 Introduzione

Salve ragazzi.

Questo documento è un viaggio che in alcune parti sarà molto comodo mentre in altre vi farà sentire abbandonati a voi stessi. Il miglior consiglio che vi posso dare è di prendere una grossa, intima tazza di caffè o di cioccolata calda, di procurarvi una confortevole sedia e, prima di avventurarvi nel mondo a volte pericoloso del network hacking, di assorbirne il contenuto.

Per comprendere meglio come utilizzare l'infrastruttura presente al di sopra del framework netfilter, raccomando di leggere il Packet Filtering HOWTO e il NAT HOWTO. Per informazioni riguardanti la programmazione del kernel suggerisco la Rusty's Unreliable Guide to Kernel Hacking e la Rusty's Unreliable Guide to Kernel Locking.

1.1 Che cos'è netfilter?

netfilter è un framework per il manipolamento dei pacchetti, esterno alla normale interfaccia socket Berkeley. Consta di 4 parti. Prima parte, ogni protocollo definisce degli hook (IPv4 ne definisce 5) i quali sono punti ben definiti nella traversata dei pacchetti nel protocol stack. In ciascuno di questi punti, il protocollo richiamerà il framework netfilter fornendo il pacchetto e il numero dell'hook.

Seconda parte, porzioni del kernel possono registrarsi per ascoltare, per ogni protocollo, differenti hook. Perciò quando un pacchetto è passato al framework netfilter, esso controlla se qualcuno si è registrato per quel determinato protocollo e hook; se sì, a ciascuno di essi è data, in ordine, una chance per esaminare (e possibilmente alterare) il pacchetto, per scartarlo (`NF_DROP`), per lasciarlo proseguire (`NF_ACCEPT`), o per chiedere a netfilter di accodarlo per lo userspace (`NF_QUEUE`).

Terza parte, i pacchetti che sono stati accodati sono accumulati (dal driver ip_queue) per essere inviati allo userspace; questi pacchetti sono gestiti in modo asincrono.

La parte finale consiste in splendidi commenti sul codice e nella documentazione. Questa è strumentale per ogni progetto sperimentale. Il motto di netfilter (rubato spudoratamente a Cort Dougan) è:

```
``Orbene... quanto è meglio di KDE?''
```

(Questo motto si affianca a 'Frustami, colpiscimi, fammi utilizzare ipchains').

In aggiunta a questo framework grezzo sono stati realizzati vari moduli che forniscono funzionalità simili ai kernel precedenti (pre-netfilter), in particolare un sistema NAT e uno di filtraggio dei pacchetti (iptables) entrambi estendibili.

1.2 Che cos'è che non va con la 2.0 e la 2.2?

1. Nessuna infrastruttura per il passaggio dei pacchetti allo userspace:
    - Programmare il kernel è complicato
    - Il codice per il kernel deve essere sviluppato in C/C++
    - Le politiche per il filtraggio dinamico non sono collocate nel kernel
    - 2.2 ha introdotto la copia dei pacchetti verso lo userspace attraverso netlink, ma la reintroduzione è lenta e soggetta a controlli di 'sanità'. Ad esempio, non è possibile avere pacchetti reimmessi che sostengono di arrivare da un'interfaccia esistente.
2. Il proxy trasparente è un accrocchio:
    - **ogni** pacchetto è controllato per stabilire se esiste un legame socket con questo indirizzo
    - All'utente root è consentito di collegarsi ad indirizzi estranei
    - Non è possibile redirigere i pacchetti generati localmente
    - REDIRECT non gestisce le risposte UDP: redirigere pacchetti UDP named verso la porta 1153 non funziona in quanto alcuni client non gradiscono risposte provenienti da una porta che non sia la 53.
    - REDIRECT non è coordinata con l'allocazione delle porte tcp/udp: un utente può ottenere una porta shadow attraverso una regola REDIRECT.
    - E' risultato corrotto almeno due volte durante la serie 2.1.
    - Il codice è estremamente intrusivo. Si considerino le statistiche del numero di #ifdef CONFIG_IP_TRANSPARENT_PROXY presenti nella 2.2.1: 34 occorrenze in 11 file. Le si confrontino con CONFIG_IP_FIREWALL, il quale ha 10 occorrenze in 5 file.
3. Creare regole di filtraggio dei pacchetti indipendenti dagli indirizzi delle interfacce non è possibile:
    - E' necessario conoscere gli indirizzi locali delle interfacce per distinguere pacchetti generati localmente o destinati localmente, da quelli in transito.
    - Non è sufficiente però nei casi di redirezione o mascheramento.
    - La catena forward ha solo l'informazione riguardante l'interfaccia di uscita, ciò significa che è necessario immaginare da dove arriva un pacchetto in base alle conoscenze sulla topografia della rete.
4. Il mascheramento è incluso nel filtraggio dei pacchetti:interazioni tra filtraggio dei pacchetti e mascheramento rendono complessa la realizzazione del firewall:
    - Al filtraggio in input, i pacchetti in risposta appaiono destinati alla box stessa
    - Al filtraggio nella forward, i pacchetti demascherati non sono visti del tutto
    - Al filtraggio in output, i pacchetti appaiono provenienti dalla box locale
5. manipolazione TOS, redirezione, ICMP unreachable e marcamento (che riguarda port forwarding, instradamento e QoS) sono collocati assieme al codice di filtraggio.

6. il codice di ipchains non è né modulare, né estendibile (per esempio filtraggio per indirizzo, filtraggio in base alle opzioni, ecc.).

7. La mancanza di un'infrastruttura adeguata ha portato alla proliferazione di differenti tecniche:

   - Mascheramento, più moduli per protocollo
   - Fast static NAT attraverso il codice di instradamento (senza gestione per protocollo)
   - Port forwarding, redirezione, auto forwarding
   - Linux NAT e progetti Virtual Server.

8. Incompatibilità tra CONFIG_NET_FASTROUTE e filtraggio dei pacchetti:

   - Pacchetti in transito attraversano in ogni caso tre catene
   - Nessun modo per segnalare se queste catene possono essere oltrepassate.

9. L'ispezione dei pacchetti scartati a causa della protezione di instradamento (es. Source Address Verification) non è possibile.

10. Nessun modo per leggere atomicamente i contatori delle regole di filtraggio dei pacchetti.

11. L'opzione CONFIG_IP_ALWAYS_DEFRAG è da selezionare in fase di compilazione, ciò rende difficile la vita alle distribuzioni che desiderano un kernel con funzionalità generiche.

1.3 Chi sei?

Sono l'unico tanto insensato disponibile a farlo. Come coautore di ipchains e attuale manutentore del Linux Kernel IP Firewall ho conosciuto molti dei problemi incontrati dalle persone con l'attuale sistema, in aggiunta mi sono anche esposto a cercare di comprendere cosa cercavano di fare.

1.4 Perché si pianta?

Woah! Avreste dovuto vederlo la **scorsa** settimana!

Siccome non sono un così grande programmatore, come noi tutti potremmo desiderare, sicuramente non ho esaminato tutti gli scenari, a causa di mancanza di tempo, attrezzatura e/o ispirazione. C'è anche una suite di test alla quale vi incoraggio a contribuire.

2 Dove si può reperire l'ultima versione?

Esiste un server CVS su netfilter.org che contiene gli ultimi HOWTO, i tool userspace e la testsuite. Per visite saltuarie, si può utilizzare

l'interfaccia Web http://cvs.netfilter.org/ .

Per prelevare gli ultimi sorgenti, si possono seguire le seguenti fasi:

1. Accedere al server CVS di netfilter come anonimo:

   ```
   cvs -d :pserver:cvs@pserver.netfilter.org:/cvspublic login
   ```

2. Quando viene richiesta la password digitare 'cvs'.

3. Controllare il codice che si utilizza:

```
# cvs -d :pserver:cvs@pserver.netfilter.org:/cvspublic co netfilter/userspace
```

4. Per aggiornare all'ultima versione utilizzare

```
cvs update -d -P
```

3 L'architettura di netfilter

Netfilter consiste semplicemente in una serie di hook collocati in vari punti del protocol stack (in questo momento, IPv4, IPv6 e DECnet). Il diagramma (idealizzato) di attraversamento nel caso dell'IPv4 assomiglia al seguente:

```
Pacchetto che attraversa il sistema Netfilter:

   --->[1]--->[Instradamento]--->[3]--->[4]--->
                    |                  ^
                    |                  |
                    |          [Instradamento]
                    v                  |
                   [2]                [5]
                    |                  ^
                    |                  |
                    v                  |
```

I pacchetti arrivano sulla sinistra: dopo aver passato il semplice controllo di sanità (ossia, no troncature, IP checksum OK, ricezione non confusa) sono passati all'hook NF_IP_PRE_ROUTING [1] del framework netfilter.

Successivamente entrano nel codice di routing, il quale decide se il pacchetto è destinato ad un'altra interfaccia o ad un processo locale. Il codice di routing potrebbe scartare i pacchetti non instradabili.

Se è destinato alla box stessa, il framework netfilter, prima che il pacchetto sia passato al processo (se presente), è chiamato nuovamente per l'hook NF_IP_LOCAL_IN [2].

Se è invece destinato ad un'altra interfaccia il framework netfilter è chiamato per l'hook NF_IP_FORWARD [3].

Il pacchetto poi, prima di essere immesso nuovamente nel cavo, passa all'hook finale, l'hook NF_IP_POST_ROUTING [4].

L'hook NF_IP_LOCAL_OUT [5] è chiamato per i pacchetti creati localmente. Si può qui notare che il codice di routing viene eseguito dopo che questo hook è stato chiamato: di fatto, il codice di routing è chiamato prima (per comprendere l'indirizzo IP sorgente e alcune opzioni IP): se si vuole alterare il routing bisogna modificare il campo 'skb->dst', come è fatto nel codice che gestisce il NAT.

3.1 Fondamenti di Netfilter

Ora segue un esempio riguardante netfilter per IPv4, si potrà notare quando ciascun hook è attivato. Questa è l'essenza di netfilter.

I moduli del kernel possono registrarsi per ascoltare qualsiasi hook. Un modulo che registra una funzione deve specificare la sua priorità all'interno dell'hook. Quando poi l'hook di netfilter è invocato dal codice del nucleo di rete, ciascun modulo registrato per questo punto è chiamato seguendo l'ordine definito dalle priorità, ed è libero di manipolare il pacchetto. Il modulo può inoltre specificare a netfilter di effettuare una delle seguenti cinque cose:

1. NF_ACCEPT: continua la traversata normalmente.
2. NF_DROP: scarta il pacchetto; non continuare la traversata.
3. NF_STOLEN: ho prelevato il pacchetto; non continuare la traversata.
4. NF_QUEUE: accoda il pacchetto (di solito per la gestione in userspace).
5. NF_REPEAT: chiama di nuovo questo hook.

Le altre parti di netfilter (gestione dei pacchetti accodati, commenti) saranno trattate più avanti nella sezione riguardante il kernel.

Subito dopo questi concetti base, si possono realizzare complesse manipolazioni dei pacchetti, come descritto nei prossimi due paragrafi.

3.2 Selezione dei pacchetti: IP Tables

Un sistema di selezione dei pacchetti, denominato IP Tables, è stato realizzato al di sopra del framework netfilter. E' un diretto discendente di ipchains (che proviene da ipfwadm, che a sua volta deriva da ipfw IIRC dei sistemi BSD) con in più l'estendibilità. I moduli del kernel possono registrare una nuova tabella e richiedere che un determinato pacchetto attraversi la tabella indicata. Questo metodo di selezione dei pacchetti è utilizzato per il filtraggio dei pacchetti (tabella 'filter'), per il Network Address Translation (tabella 'nat'), e per il manipolamento generico dei pacchetti prima dell'instradamento (tabella 'mangle').

Gli hook registrati con netfilter sono i seguenti (in figura sono indicate in ordine, per ogni hook, le funzioni chiamate):

```
--->PRE------>[ROUTE]--->FWD---------->POST------>
    Conntrack    |       Mangle   ^    Mangle
    Mangle       |       Filter   |    NAT (Src)
    NAT (Dst)    |                |    Conntrack
    (QDisc)      |             [ROUTE]
                 v                |
                 IN Filter       OUT Conntrack
                 |  Conntrack     ^  Mangle
                 |  Mangle        |  NAT (Dst)
                 v                |  Filter
```

3.2.1 Filtraggio dei pacchetti

Questa tabella, 'filter', non deve mai alterare i pacchetti: solo filtrarli.

Uno dei vantaggi del filtro iptables rispetto a ipchains consiste nel fatto che è più compatto e veloce, inoltre si aggancia in netfilter nei punti NF_IP_LOCAL_IN, NF_IP_FORWARD e NF_IP_LOCAL_OUT. Ciò significa che per ogni pacchetto c'è un (e solo un) punto possibile per il filtraggio. Le cose in questo modo sono molto più semplici. Inoltre, il framework netfilter provvede sia un'interfaccia di input che di output per l'hook NF_IP_FORWARD, ciò implica la possibilità di avere diversi tipi, e anche piuttosto semplici, di filtraggio.

Nota: ho implementato le porzione del kernel riguardanti ipchains e ipfwadm come moduli sopra netfilter, consentendo così di poter utilizzare i vecchi tool userspace ipfwadm e ipchains senza la necessità di un aggiornamento.

3.2.2 NAT

Questo è il regno della tabella 'nat', che riceve i pacchetti da due hook di netfilter: per i pacchetti non locali NF_IP_PRE_ROUTING e NF_IP_POST_ROUTING sono perfetti per la modifica rispettivamente della destinazione e della sorgente. Se è definito CONFIG_IP_NF_NAT_LOCAL, gli hook NF_IP_LOCAL_OUT e NF_IP_LOCAL_IN sono usati per alterare la destinazione dei pacchetti locali.

Questa tabella è leggermente differente rispetto alla tabella 'filter', in questa solo il primo pacchetto di una nuova connessione attraversa la tabella: il risultato di questa traversata sarà poi applicata a tutti i pacchetti futuri appartenenti alla stessa connessione.

Mascheramento, Port Forwarding, Proxy Trasparente Ho suddiviso il NAT in Source NAT (dove al primo pacchetto viene alterata la sorgente) e Destination NAT (al primo pacchetto viene alterata la destinazione).

Il mascheramento è una forma speciale di Source NAT: port forwarding e il proxy trasparente sono invece forme speciali di Destination NAT. Queste ora utilizzano tutte il framework NAT, invece di essere entità indipendenti.

3.2.3 Packet Mangling

La tabella per il packet mangling (tabella 'mangle') è usata per modificare le informazioni del pacchetto.I target TOS e TCPMSS sono un esempio di utilizzo. La tabella mangle si aggancia a tutti e cinque gli hook di netfilter. (questo cambiamento è avvenuto con la versione 2.4.18 del kernel. Nei precedenti kernel mangle non si agganciava a tutti gli hook.)

3.3 Connection Tracking

Il Connection tracking (tracciamento delle connessioni) è fondamentale per il NAT, tuttavia è implementato come modulo; ciò per consentire di estendere il codice di filtraggio dei pacchetti, permettendo l'utilizzo semplice e pulito del connection tracking (modulo 'state').

3.4 Altre aggiunte

La nuova flessibilità fornisce l'opportunità di realizzare cose davvero incredibili, oltre che di apportare miglioramenti o di realizzare completi rimpiazzi da combinare e adattare.

4 Informazioni per i programmatori

Voglio confidarvi un segreto: il mio criceto preferito ha realizzato tutto il codice. Io sono solo un tramite, una facciata se si vuole, appartenente al grande piano del mio animale. Perciò, non mi si rimproveri se ci sono dei bachi. Incolpate lo svelto, l'impellicciato.

4.1 Comprendere ip_tables

iptables provvede semplicemente un array di regole in memoria (da qui il nome 'iptables') e informazioni su dove i pacchetti da ciascun hook dovrebbero cominciare la traversata. Dopo che una tabella è stata registrata, lo userspace può leggere e sostituirne il contenuto utilizzando getsockopt() e setsockopt().

iptables non si registra con nessun hook di netfilter: rilascia ad altri moduli questo compito, provvede quindi solo a passare i pacchetti in modo appropriato; un modulo deve registrare hook e ip_tables separatamente, e fornire il meccanismo per chiamare ip_tables quando viene raggiunto l'hook.

4.1.1 Strutture dati ip_tables

Per convenienza viene utilizzata, per rappresentare una regola sia nello userspace sia nel kernel, la stessa struttura dati, sebbene qualche campo sia utilizzato solo nel kernel.

Ogni regola consiste delle seguenti parti:

1. Una 'struct ipt_entry'.
2. Zero o più strutture 'struct ipt_entry_match', ognuna con un ammontare variabile (0 o più byte) di dati allegati.
3. Una struttura 'struct ipt_entry_target' con un ammontare variabile (0 o più byte) di dati allegati.

La natura variabile della regola dà un'enorme disponibilità di flessibilità per le estensioni, come si vedrà, in particolare su come ciascun match (corrispondenza) oppure target (obiettivo) può trasportare un quantitativo arbitrario di dati. Ciò comporta comunque alcune trappole: è necessario prestare attenzione all'allineamento. Ciò avviene assicurandosi che le strutture 'ipt_entry', 'ipt_entry_match' e 'ipt_entry_target' siano convenientemente dimensionate, e che tutti i pacchetti siano confinati nell'allineamento massimo della macchina utilizzando la macro IPT_ALIGN().

La 'struct ipt_entry' ha i seguenti campi:

1. Una 'struct ipt_ip', contenente la specificazione dell'intestazione IP che deve essere soddisfatta.
2. Un bitfield 'nf_cache' che mostra quali parti del pacchetto questa regola ha esaminato.
3. Un campo 'target_offset' che indica l'offset da cui, a partire dall'inizio di questa regola, la struttura ipt_entry_target comincia. Questo dovrebbe essere sempre allineato correttamente (attraverso la macro IPT_ALIGN).
4. Un campo 'next_offset' che indica la dimensione totale di questa regola, inclusi i match e i target. Anche questo campo dovrebbe essere allineato correttamente utilizzando la macro IPT_ALIGN.
5. Un campo 'comefrom' utilizzato dal kernel per tracciare la traversata dei pacchetti.
6. Un campo 'struct ipt_counters' contenente i contatori del numero e dei byte riguardanti i pacchetti che hanno soddisfatto questa regola.

Le strutture 'struct ipt_entry_match' e 'struct ipt_entry_target' sono molto simili, in quanto contengono un campo di lunghezza totale (IPT_ALIGN'alizzato) (rispettivamente 'match_size' e 'target_size') e una union che mantiene il nome del match o del target (per userspace) e un puntatore (per il kernel).

A causa della complicata natura della struttura dati delle regole sono state previste alcune routine di aiuto:

ipt_get_target()
: Questa funzione inline restituisce un puntatore al target di una regola.

IPT_MATCH_ITERATE()
: Questa macro invoca la funzione specificata per ogni soddisfazione della regola data. Il primo argomento della funzione è la 'struct ipt_match_entry', altri argomenti (se presenti) sono quelli forniti alla macro IPT_MATCH_ITERATE(). Questa funzione ritorna zero in caso di iterazione, o un valore diverso da zero in caso di stop.

IPT_ENTRY_ITERATE()

Questa funzione richiede un puntatore ad una entry, la dimensione totale della tabella delle entry e una funzione da invocare. Il primo argomento della funzione è la 'struct ipt_entry', altri argomenti (se presenti) sono quelli forniti alla macro IPT_ENTRY_ITERATE(). Questa funzione ritorna zero in caso di iterazione, o un valore diverso da zero in caso di stop.

4.1.2 ip_tables dallo Userspace

Lo userspace ha quattro funzioni: può leggere la tabella corrente, leggere le informazioni (posizione degli hook e dimensione della tabella), sostituire la tabella (ed ottenere i vecchi contatori) e aggiungere nuovi contatori.

Ciò permette la simulazione, attraverso lo userspace, di qualsiasi operazione atomica: ciò è ottenuto attraverso la libreria libiptc, la quale provvede per i programmi una comoda semantica add/delete/replace.

Siccome queste tabelle sono trasferite nello spazio del kernel, l'allineamento diventa un problema per quelle macchine che possiedono tipi di regole userspace e kernelspace differenti (es. Sparc64 con userland a 32-bit). Questi casi sono gestiti sovrascrivendo, per queste piattaforme, la definizione di IPT_ALIGN nel file 'libiptc.h'.

4.1.3 Uso di ip_tables e traversata

Il kernel comincia ad esaminare dalla locazione indicata dal particolare hook. La regola è esaminata se gli elementi della 'struct ipt_ip' sono soddisfatti, ciascuna 'struct ipt_entry_match' è poi controllata a turno (la funzione associata con quella soddisfatta è invocata). Se la funzione corrispondente ritorna 0, le iterazioni sono fermate su questa regola. Se il parametro 'hotdrop' è impostato a 1, il pacchetto sarà immediatamente scartato (è utilizzata per alcuni pacchetti sospetti, come nella funzione match tcp).

Se l'iterazione continua verso la fine, i contatori sono incrementati e la 'struct ipt_entry_target' è esaminata: se è un target (obiettivo) standard allora viene letto il campo 'verdict' (valore negativo indica verdetto del pacchetto, positivo indica un offset a cui saltare). Se la risposta è positiva e l'offset non corrisponde a quello della regola successiva, la variabile 'back' è impostata, e il valore 'back' precedente è collocato nel campo 'comefrom' di questa regola.

Per i target non-standard viene chiamata la funzione target: essa restituisce un verdetto (target non standard non possono saltare, in quanto si potrebbe infrangere il codice statico di determinazione dei loop). Il verdetto può corrispondere anche a IPT_CONTINUE per continuare con la regola successiva.

4.2 Estendere iptables

Siccome sono pigro, `iptables` è abbastanza estendibile. Questo è sostanzialmente un tentativo di passare il lavoro ad altre persone, e rappresenta proprio ciò che è l'Open Source dopo tutto (vedi Free Software, ciò che RMS dichiara a riguardo della parola freedom, e io ero presente ad uno di questi discorsi quando ho scritto ciò).

Estendere `iptables` potenzialmente coinvolge due parti: estensione del kernel, con la scrittura di un nuovo modulo, e possibilmente estensione del programma userspace `iptables`, con la realizzazione di una nuova libreria condivisa.

4.2.1 Il Kernel

Realizzare un modulo per il kernel è di per sé abbastanza semplice, come si può notare dagli esempi. Una cosa da sapere è che il codice deve essere rientrante: ci può essere un solo pacchetto in arrivo dallo userspace mentre un altro giunge su un interrupt. Di fatto in SMP ci può essere un pacchetto su un interrupt per CPU in 2.3.4 e oltre.

Le funzioni che è necessario conoscere sono:

init_module()
: Questa funzione è il punto di entrata del modulo. Restituisce un numero di errore negativo, oppure 0 se riesce a registrarsi con successo con netfilter.

cleanup_module()
: Questo è il punto di uscita del modulo; dovrebbe eliminare la registrazione con netfilter.

ipt_register_match()
: Utilizzata per registrare un nuovo tipo di match (corrispondenza). Si passerà ad essa una 'struct ipt_match' di solito dichiarata come variabile statica (file-scope).

ipt_register_target()
: Utilizzata per registrare un nuovo tipo. Si passerà ad essa una 'struct ipt_target' di solito dichiarata come variabile statica (file-scope).

ipt_unregister_target()
: Utilizzata per rimuovere la registrazione del proprio target.

ipt_unregister_match()
: Utilizzata per rimuovere la registrazione del proprio match.

Un avvertimento riguardo la realizzazione di cose complicate (come ad esempio provvedere dei contatori) nello spazio extra del proprio match o del proprio target. Sulle macchine SMP, l'intera tabella è duplicata usando memcpy per ciascuna CPU: se davvero si desidera mantenere informazioni in modo centralizzato, si dovrebbe dare un'occhiata al metodo utilizzato con il match 'limit'.

Nuove funzioni match Nuove funzione match sono di solito realizzate come moduli a sé stanti. E' possibile estendere questi moduli successivamente, sebbene solitamente non necessario. Un modo potrebbe essere quello di utilizzare la funzione 'nf_register_sockopt' del framework netfilter, per consentire agli utenti di comunicare direttamente con i propri moduli. Un'altra soluzione potrebbe essere quella di esportare i simboli per altri moduli affinché si registrino allo stesso modo di netfilter e iptables.

Il nocciolo della propria funzione match sarà la struttura ipt_match che sarà passata a 'ipt_register_match()'. Questa struttura ha i seguenti campi:

list
: Questo campo è impostabile con qualsiasi robaccia, facciamo '{ NULL, NULL }'.

name
: Questo campo specifica il nome della funzione match, come riferito allo userspace. Il nome, affinché l'auto-caricamento funzioni, deve corrispondere al nome del modulo (ossia, se il nome è mac, il modulo dovrà essere ipt_mac.o).

match

Questo campo è un puntatore ad una funzione match che prende skb, i puntatori ai dispositivi in ed out (uno dei quali potrebbe essere NULL, a seconda dell'hook), un puntatore ai dati match della regola che è risultata soddisfatta, la dimensione della regola, l'offset IP (non zero significa un frammento non di testa), un puntatore all'intestazione del protocollo (ossia, giusto l'intestazione IP), la lunghezza dei dati (ossia la dimensione del pacchetto meno l'intestazione IP) e infine un puntatore ad una variabile 'hotdrop'. Dovrebbe restituire un valore non-zero se il pacchetto la soddisfa, e può impostare 'hotdrop' a 1 se restituisce 0, per segnalare che questo pacchetto deve essere scartato immediatamente.

checkentry

Questo campo è un puntatore ad una funzione la quale controlla le specificazioni di una regola; se restituisce 0, allora la regola dell'utente non sarà accettata. Ad esempio, il tipo match tcp accetterà solo pacchetti tcp, quindi se la 'struct ipt_ip', parte della regola, non specifica che il protocollo è tcp, uno zero è restituito. L'argomento tablename consente al match di controllare con quali tabelle può essere utilizzato, mentre la 'hook_mask' è una bitmask di hook da cui questa regola può essere chiamata: se il match non ha senso per qualche hook di netfilter, si può evitare ciò in questo punto.

destroy

Questo campo è un puntatore ad una funzione la quale è invocata quando una entry, che utilizza questo match, è cancellata. Ciò consente di allocare dinamicamente delle risorse nella checkentry e di rilasciarle qui.

me

Questo campo è da impostare a 'THIS_MODULE', il quale fornisce un puntatore al modulo. Esso comporta l'aumento e la diminuzione dell'usage-count a seconda che le regole di questo tipo siano create o distrutte. Ciò previene che un utente rimuova il modulo (e che quindi cleanup_module() sia invocata) quando esiste una regola riferita ad esso.

Nuovi target Se il proprio target modifica il pacchetto (es. l'intestazione o il corpo), bisogna chiamare la funzione skb_unshare() per copiare il pacchetto nel caso che sia clonato: altrimenti qualsiasi raw socket che ha un clone di skbuff vedrà le modifiche (es. le persone potrebbero vedere cose arcane in tcpdump).

Nuovi target sono di solito realizzati a loro volta come moduli a sé stanti. La discussione riguardante il capitolo 'Nuove funzioni match' può essere ugualmente utilizzata anche qui.

Il nocciolo del proprio nuovo target è la struct ipt_target che sarà poi passata alla ipt_register_target(). La struttura ha i seguenti campi:

list

Questo campo è impostabile con qualsiasi robaccia, facciamo '{ NULL, NULL }'.

name

Questo campo specifica il nome della funzione target, come riferito allo userspace. Il nome, affinché l'auto-caricamento funzioni, deve corrispondere al nome del modulo (ossia, se il nome è REJECT, il modulo dovrà essere ipt_REJECT.o).

target

Questo è un puntatore alla funzione target, la quale richiede skbuff, il numero di hook, i puntatori ai device input ed output (uno di essi potrebbe essere NULL), un puntatore ai dati target, la dimensione dei dati target, e la posizione della regola nella tabella. La funzione target restituisce o IPT_CONTINUE (-1) se la traversata deve continuare, oppure uno dei verdetti di netfilter verdict (NF_DROP, NF_ACCEPT, NF_STOLEN etc.).

checkentry

Questo campo è un puntatore ad una funzione la quale controlla le specificazioni di una regola; se restituisce 0, allora la regola dell'utente non sarà accettata.

destroy

Questo campo è un puntatore ad una funzione che è invocata quando una entry con questo target è cancellata. Ciò consente di allocare dinamicamente le risorse nella checkentry e di rilasciarle qui.

me

Questo campo è da impostare a 'THIS_MODULE', il quale fornisce un puntatore al modulo. Esso comporta l'aumento e la diminuzione dell'usage-count a seconda che le regole di questo tipo siano create o distrutte. Ciò previene che un utente rimuova il modulo (e che quindi cleanup_module() sia invocata) quando esiste una regola riferita ad esso.

Nuove tabelle Se desiderato si può creare una nuova tabella con scopi specifici. Per crearla, si deve chiamare 'ipt_register_table()' fornendo una 'struct ipt_table', la quale ha i seguenti campi:

list

Questo campo è impostabile con qualsiasi robaccia, facciamo '{ NULL, NULL }'.

name

Questo campo specifica il nome della funzione target, come riferito allo userspace. Il nome, affinché l'auto-caricamento funzioni, deve corrispondere al nome del modulo (ossia, se il nome è nat, il modulo dovrà essere ipt_nat.o).

table

Questa è una 'struct ipt_replace' completamente popolata, proprio come utilizzata nello userspace per sostituire una tabella. Il puntatore 'counters' può essere impostato a NULL. Questa struttura dati può essere dichiarata '__initdata', in questo modo dopo il boot sarà eliminata.

valid_hooks

Questa è una bitmask di hook IPv4 di netfilter, con cui si accederà alla tabella: questa è usata per verificare che quelle entry point siano valide, e per calcolare i possibili hook per le funzioni 'checkentry()' di ipt_match e ipt_target.

lock

Questo campo è l'interruttore lettura-scrittura dell'intera tabella; la si inizializzi a RW_LOCK_UNLOCKED.

private

Questo campo è utilizzato internamente dal codice di ip_tables.

4.2.2 Tool userspace

Ora dopo aver realizzato uno splendente modulo per il kernel, si potrebbe desiderare di controllare le opzioni dallo userspace. Piuttosto che avere una versione derivata di `iptables` per ogni estensione, io utilizzo l'ultimissima tecnologia degli anni 90: i furbies. Scusate, intendevo le librerie condivise.

Nuove tabelle generalmente non richiedono alcuna estensione di `iptables`: l'utente può utilizzare l'opzione '-t' per far sì che sia possibile utilizzare la nuova tabella.

La libreria condivisa dovrebbe avere una funzione '_init()', la quale sarà chiamata automaticamente appena caricata: è l'equivalente della funzione 'init_module()' per i moduli del kernel. Questa dovrebbe poi chiamare 'register_match()' o 'register_target()', a seconda che la libreria provveda un nuovo match o un nuovo target.

E' necessario fornire una libreria condivisa: può essere usata per inizializzare parte della struttura, o fornire opzioni addizionali. Insisto nel consigliare la creazione della libreria condivisa anche se non deve fare niente, ma solo per ridurre i problemi che si verificano quando le librerie sono mancanti.

Esistono funzioni molto utili descritte nel file header 'iptables.h', in particolare:

check_inverse()

controlla se un argomento è attualmente '!', e in tal caso imposta il flag 'invert' se non già impostato. Se restituisce vero, si può incrementare optind, come fatto negli esempi.

string_to_number()

converte una stringa in un numero dell'intervallo dato, restituisce -1 se malformato o fuori intervallo. 'string_to_number' si appoggia a 'strtol' (vedi la pagina del manuale), questo significa che uno 0x in testa indica che il numero è in base esadecimale, e uno 0 indica che è in base ottale.

exit_error()

dovrebbe essere invocata se si incontra un errore. Di solito il primo argomento è 'PARAMETER_PROBLEM', il quale specifica che l'utente non ha utilizzato correttamente la linea comando.

Nuove funzioni match La funzione _init() della libreria condivisa passa a 'register_match()' un puntatore ad una 'struct iptables_match' statica, che ha i seguenti campi:

next

Questo puntatore è utilizzato per realizzare una lista linkata di match (come quelle utilizzate per visualizzare le regole). Dovrebbe essere inizialmente impostata a NULL.

name

Nome della funzione match. Questa dovrebbe corrispondere al nome della libreria (es. tcp per 'libipt_tcp.so').

version

Di solito impostata con la macro IPTABLES_VERSION: questa è utilizzata per assicurarsi che l'eseguibile `iptables` non utilizzi per sbaglio le librerie condivise errate.

size

Dimensione dei dati match per questo match; si dovrebbe utilizzare la macro IPT_ALIGN() per assicurarsi che sia correttamente allineato.

userspacesize

Per alcuni match, il kernel modifica alcuni campi internamente (il target 'limit' è uno di questi casi). Ciò significa che una semplice 'memcmp()' è insufficiente per comparare due regole (richiesto per la funzionalità delete-matching-rule). Se questo è il caso, si sistemino tutti i campi che non cambiano all'inizio della struttura, e qui si metta la loro dimensione. Di solito questa ha lo stesso valore del campo 'size'.

help

Funzione che visualizza le informazioni sull'uso delle opzioni.

init

Questa può essere utilizzata per inizializzare lo spazio extra (se presente) della struttura ipt_entry_match, e per impostare qualsiasi bit nfcache; se si sta esaminando qualcosa non esprimibile utilizzando il contenuto di 'linux/include/netfilter_ipv4.h', allora si faccia semplicemente un OR con i bit NFC_UNKNOWN. Sarà chiamata prima di 'parse()'

parse

Questa funzione è chiamata quando un'opzione non conosciuta è presente nella linea comando: dovrebbe restituire non-zero se l'opzione è effettivamente della propria libreria. 'invert' è vera se un '!' è già stato incontrato. Il puntatore 'flags' è di esclusivo utilizzo per la propria libreria match, e di solito è utilizzato per memorizzare una bitmask di opzioni che sono state specificate. Ci si assicuri di aver aggiustato il campo nfcache. Riallocando si può estendere, se necessario, la dimensione della struttura 'ipt_entry_match', ma poi è necessario assicurarsi che la dimensione sia passata attraverso la macro IPT_ALIGN.

final_check

Questa è chiamata dopo che la linea di comando è stata analizzata, inoltre viene passato l'intero 'flags' riservato per la propria libreria. Ciò dà la possibilità di controllare che tutte le opzione obbligatorie siano state specificate, quindi si invochi 'exit_error()' se è il caso.

print

Utilizzata dal codice di visualizzazione della catena per stampare (sullo standard output) le informazioni match extra (se presenti) di una regola. L'opzione numeric viene impostata se l'utente specifica il flag '-n'.

save

Questa funzione è il contrario della parse: è utilizzata da 'iptables-save' per riprodurre le opzioni usate per creare la regola.

extra_opts

Questa è una lista di opzioni extra, terminata con un NULL, offerte dalla propria libreria. E' fusa con le opzioni correnti e passata alla getopt_long; consultare le pagine del manuale per i dettagli. Il codice di ritorno della getopt_long diventa poi il primo argomento ('c') della funzione 'parse()'.

Ci sono altri elementi extra alla fine di questa struttura, utilizzati internamente da `iptables`: non è necessario impostarli.

Nuovi target La funzione _init() della propria libreria condivisa passa a 'register_target()' un puntatore ad una 'struct iptables_target' statica, la quale ha campi simili alla struttura iptables_match vista prima.

4.2.3 Utilizzare 'libiptc'

`libiptc` è la libreria di controllo di iptables, progettata per visualizzare e manipolare le regole nel modulo iptables del kernel. Anche se il suo utilizzo corrente riguarda il programma iptables, consente di scrivere altri tool in modo molto semplice. E' necessario essere root per utilizzare queste funzioni.

Le tabelle del kernel sono semplici tabelle di regole e un insieme di numeri che rappresentano gli entry point. I nomi delle catene (INPUT, ecc.) sono forniti come astrazioni dalla libreria. Le catene definite dall'utente sono etichettate inserendo un nodo di errore prima dell'inizio della catena definita dall'utente, la quale contiene nella sezione dei dati extra del target, il nome della catena (le posizioni delle catene incorporate sono definite attraverso gli entry point delle tre tabelle).

Sono forniti i seguenti target standard: ACCEPT, DROP, QUEUE (che sono tradotti rispettivamente in NF_ACCEPT, NF_DROP, e NF_QUEUE,), RETURN (che è tradotto in un valore speciale IPT_RETURN gestito da ip_tables), e JUMP (che è tradotto dal nome della catena nel reale offset all'interno della tabella).

Quando 'iptc_init()' è invocata, la tabella, inclusi i contatori, è letta. Questa tabella è manipolabile attraverso le funzioni 'iptc_insert_entry()', 'iptc_replace_entry()', 'iptc_append_entry()', 'iptc_delete_entry()', 'iptc_delete_num_entry()', 'iptc_flush_entries()', 'iptc_zero_entries()', 'iptc_create_chain()' 'iptc_delete_chain()', e 'iptc_set_policy()'.

I cambiamenti alla tabella non saranno apportati fino a quando non sarà chiamata la funzione 'iptc_commit()'. Ciò significa che è possibile che due librerie utenti, operanti sulla stessa catena, concorrano una con l'altra; per prevenire queste situazioni sarebbe necessario il locking, al momento non effettuabile.

Non esiste concorrenza per quanto riguarda i contatori; i contatori sono sommati dopo nel kernel in un modo tale che i loro incrementi, tra il tempo di lettura e scrittura della tabella, siano ancora visibili nella nuova tabella.

Ci sono diverse funzioni di aiuto:

iptc_first_chain()

Questa funzione restituisce il nome della prima catena della tabella.

iptc_next_chain()

Questa funzione restituisce il nome della catena successiva della tabella: NULL indica che non ci sono altre catene.

iptc_builtin()

Restituisce true (vero) se il nome della catena fornito corrisponde al nome di una catena definita da iptables.

iptc_first_rule()

Questa funzione restituisce un puntatore alla prima regola della catena avente il nome dato: NULL indica catena vuota.

iptc_next_rule()

Questa restituisce un puntatore alla regola successiva della catena: NULL indica fine della catena.

iptc_get_target()

Questa funzione permette di ottenere il target di una data regola. Se si tratta di un target estensione viene restituito il nome del target. Se corrisponde ad un salto ad un'altra catena viene restituito il nome della catena. Se è un verdetto (es. DROP) ne viene restituito il nome. Se non ha un target (regola accounting-style) viene restituita una stringa vuota.

Si noti che questa funzione dovrebbe essere utilizzata al posto della consultazione diretta del valore del campo 'verdict' nella struttura ipt_entry, dato che offre le sopraindicate ulteriori interpretazioni del verdetto standard.

iptc_get_policy()

Questa consente di ottenere la policy (tattica) di una catena incorporata, e la sua statistica di utilizzo attraverso l'argomento 'counters'.

iptc_strerror()

Questa funzione restituisce una ancor più eloquente spiegazione riguardo un codice di fallimento della libreria iptc. Se una funzione fallisce, essa imposta sempre errno: questo valore può essere passato a iptc_strerror() per generare un messaggio di errore.

4.3 Comprendere il NAT

Benvenuti al Network Address Translation presente nel kernel. Si noti che l'infrastruttura offerta è stata progettata più con l'obiettivo della completezza piuttosto che della bruta efficienza, interventi futuri potranno incrementare sensibilmente le prestazioni. Al momento sono contento che funzioni.

NAT è suddiviso in connection tracking (il quale non manipola affatto i pacchetti), e il codice di NAT stesso. Il connection tracking è stato progettato per essere utilizzato come modulo di iptables, ed effettua sottili distinzioni riguardanti lo stato, che generalmente il NAT non considera proprio.

4.3.1 Connection Tracking

Il connection tracking (tracciamento delle connessioni) si aggancia agli hook di alta priorità NF_IP_LOCAL_OUT e NF_IP_PRE_ROUTING, in ordine, per vedere i pacchetti prima che entrino nel sistema.

Il campo nfct della skb è un puntatore ad uno degli array infos[], presenti all'interno della struct ip_conntrack. Quindi si può ricavare lo stato della skb in base a quale elemento di questo array esso sta puntando: questo puntatore codifica sia la struttura state sia la relazione di skb con questo stato.

Il modo migliore per estratte il campo 'nfct' consiste nel chiamare 'ip_conntrack_get()', la quale restituisce NULL se non è impostato, oppure il puntatore alla connessione, inoltre compila ctinfo che descrive la relazione del pacchetto con questa connessione. Questo tipo enumerato può assumere diversi valori:

IP_CT_ESTABLISHED

Il pacchetto è parte di una connessione stabilita, nella direzione originale.

IP_CT_RELATED

Il pacchetto è correlato ad una connessione, ed è passato nella direzione originale.

IP_CT_NEW

Il pacchetto sta cercando di creare una nuova connessione (ovviamente, è nella direzione originale).

IP_CT_ESTABLISHED + IP_CT_IS_REPLY

Il pacchetto è parte di una connessione stabilita, nella direzione di risposta.

IP_CT_RELATED + IP_CT_IS_REPLY

Il pacchetto è correlato ad una connessione, ed è passato nella direzione di risposta.

Quindi un pacchetto in risposta può essere identificato effettuando un test di tipo >= IP_CT_IS_REPLY.

4.4 Estendere il Connection tracking/NAT

Questi framework sono stati progettati per essere adattati a qualsiasi tipo di protocollo e tipo differente di mapping. Alcuni di questi tipi di mapping potrebbero essere piuttosto specifici, per esempio mapping load-balancing/fail-over.

Internamente, il connection tracking, prima di cercare connessioni o regole che siano soddisfatte, converte un pacchetto in una tupla, che consiste nella parte interessante del pacchetto. Questa tupla ha una parte manipolabile e una parte non manipolabile; chiamate src e dst, almeno come appaiono nel mondo del SNAT durante l'ispezione del primo pacchetto (nel caso del mondo del Destination NAT corrisponderebbero ad un pacchetto di risposta). La tupla per ogni pacchetto dello stesso stream, nella stessa direzione, è sempre uguale.

Ad esempio, una tupla di un pacchetto TCP contiene la parte manipolabile: indirizzo IP sorgente e porta sorgente, la parte non manipolabile: indirizzo IP destinazione e porta destinazione. Tuttavia non è necessario che la parte manipolabile e la parte non manipolabile siano dello stesso tipo; ad esempio, una tupla di un pacchetto ICMP contiene la parte manipolabile: indirizzo IP sorgente e l'id ICMP, e la parte non manipolabile: indirizzo IP destinazione, tipo e codice ICMP.

Ogni tupla ha un inverso, il quale corrisponde alla tupla relativa ai pacchetti dello stream che arrivano in risposta. Ad esempio, l'inverso di un pacchetto ICMP ping, icmp id 12345, da 192.168.1.1 a 1.2.3.4, è un pacchetto ping-reply, icmp id 12345, da 1.2.3.4 a 192.168.1.1.

Queste tuple, rappresentate dalla 'struct ip_conntrack_tuple', sono ampiamente utilizzate. Di fatto, assieme con l'hook da cui il pacchetto arriva (il quale ha effetto sul tipo di manipolazione), e il dispositivo coinvolto, questa corrisponde all'informazione completa del pacchetto.

La maggior parte delle tuple sono contenute entro la 'struct ip_conntrack_tuple_hash', che aggiunge una entry alla lista doppiamente linkata, e un puntatore alla connessione a cui la tupla appartiene.

Una connessione è rappresentata dalla 'struct ip_conntrack' la quale ha due campi 'struct ip_conntrack_tuple_hash': uno riguardante la direzione del pacchetto originale (tuplehash[IP_CT_DIR_ORIGINAL]), e uno riguardante la direzione dei pacchetti in risposta (tuplehash[IP_CT_DIR_REPLY]).

Comunque, la prima cosa che il NAT fa è di verificare se il codice del connection tracking è riuscito ad estrarre una tupla e a trovare una connessione esistente, controllando il campo nfct della skbuff; ciò permette di conoscere se è un tentativo di nuova connessione, o in caso contrario, quale direzione ha; nell'ultimo caso inoltre sono poi effettuate le manipolazioni stabilite precedentemente per questa connessione.

Se corrisponde invece all'inizio di una nuova connessione, si cercherà una regola per questa tupla, utilizzando il meccanismo standard di attraversamento di iptables. Se una regola viene soddisfatta, è utilizzata per inizializzare le manipolazioni, sia per quella direzione sia per la risposta; il codice del connection tracking ci farà notare che la risposta, come aspettato, è stata cambiata. Quindi sarà manipolata come sopra.

Se non c'é nessuna regola, viene creato un collegamento 'null': questo di solito non mappa il pacchetto, ma esiste per assicurare che non si mappi un altro stream sopra uno esistente. Qualche volta, il collegamento null non può essere creato, in quanto si è già mappato sopra uno stream, in questo caso la manipolazione per protocollo potrebbe provare a rimapparla, anche se è nominalmente un collegamento 'null'.

4.4.1 Target NAT standard

I target NAT sono simili ai target estensione di iptables, eccetto per il fatto che sono utilizzati solo con la tabella 'nat'. Sia i target SNAT che DNAT prendono una 'struct ip_nat_multi_range' come dato extra; ciò serve per specificare l'intervallo di indirizzi che è consentito utilizzare per un mapping. Un elemento di intervallo 'struct ip_nat_range' consiste in un indirizzo IP minimo e massimo inclusi, e in un valore massimo e minimo inclusi specifici del protocollo (es. porte TCP). C'è inoltre spazio per i flag, i quali specificano se l'indirizzo IP può essere mappato (qualche volta si desidera mappare solo la parte specifica del protocollo di una tupla, non l'IP), e un altro per indicare che la parte specifica del protocollo dell'intervallo è valida.

Un intervallo multiplo consiste in un array di elementi 'struct ip_nat_range'; ciò significa che un intervallo potrebbe essere 1.1.1.1-1.1.1.2 porte 50-55 E 1.1.1.3 porta 80. Ogni elemento dell'intervallo viene aggiunto all'intervallo (una unione, per chi ama la teoria degli insiemi).

4.4.2 Nuovi protocolli

All'interno del kernel Implementare un nuovo protocollo prima di tutto significa decidere quale parte di una tupla deve essere manipolabile e quale no. Qualsiasi cosa nella tupla deve avere la proprietà di identificare univocamente lo stream. La parte manipolabile della tupla è poi la parte su cui si può effettuare il NAT: per il caso TCP questa è la porta sorgente, per ICMP è l'ID; insomma qualcosa utilizzabile come identificatore dello stream. La parte non manipolabile consiste invece nella parte restante del pacchetto, che identifica univocamente lo stream, ma con cui non si può giocare (es. porta destinazione TCP, tipo ICMP).

Una volta prese queste decisioni, si può scrivere un'estensione al codice del connection-tracking nella directory, e proseguire popolando la struttura 'ip_conntrack_protocol' che è necessario poi passare alla funzione 'ip_conntrack_register_protocol()'.

I campi della 'struct ip_conntrack_protocol' sono:

list

Da impostare a '{ NULL, NULL }'; utilizzata per unirsi alla lista.

proto

Il numero del protocollo; vedere '/etc/protocols'.

name

Nome del protocollo. Questo è il nome che l'utente vedrà; in genere è meglio se corrisponde ad uno dei nomi canonici presenti in '/etc/protocols'.

pkt_to_tuple

Funzione che, dato il pacchetto, riempie le parti specifiche della tupla riguardanti il protocollo. Il puntatore 'datah' punta all'inizio dell'intestazione (giusto dopo l'intestazione IP), mentre datalen è la lunghezza del pacchetto. Se il pacchetto non è abbastanza lungo per contenere le informazioni dell'intestazione, restituisce 0; datalen sarà comunque sempre di almeno 8 byte (imposto dal framework).

invert_tuple

Questa funzione è utilizzata semplicemente per cambiare la parte specifica del protocollo della tupla in modo tale che appaia come quella di un pacchetto di risposta.

print_tuple

Questa funzione è utilizzata per stampare la parte specifica del protocollo di una tupla; di solito è usata la funzione sprintf() con il buffer fornito. Restituisce il numero di caratteri utilizzati. Questa è utilizzata per stampare gli stati per la /proc.

print_conntrack

Questa funzione è utilizzata per stampare la parte privata della struttura conntrack, se presente, usata inoltre anche per stampare gli stati in /proc.

packet

Questa funzione è chiamata quando un pacchetto è visto quale parte di una connessione stabilita. Si ottiene un puntatore alla struttura conntrack, l'intestazione IP, la lunghezza, e la ctinfo. Ritorna un verdetto per il pacchetto (normalmente NF_ACCEPT), oppure -1 se il pacchetto non è parte valida di una connessione. Si può cancellare la connessione dall'interno di questa funzione se desiderato, ma è d'obbligo utilizzare la seguente forma per evitare concorrenze (vedere ip_conntrack_proto_icmp.c):

```
if (del_timer(&ct->timeout))
        ct->timeout.function((unsigned long)ct);
```

new

Questa funzione è chiamata quando un pacchetto crea una connessione per la prima volta; non c'è un argomento ctinfo, dato che il primo pacchetto è ctinfo IP_CT_NEW per definizione. Restituisce 0 se fallisce nella creazione della connessione, o un immediato timeout di connessione.

Una volta scritto e testato ciò è possibile tracciare il proprio nuovo protocollo, ora è tempo di istruire NAT su come interpretarlo. Ciò significa realizzare un nuovo modulo; un'estensione al codice NAT e di andare a popolare la struttura 'ip_nat_protocol' che sarà necessario passare a 'ip_nat_protocol_register()'.

list

Da impostare a '{ NULL, NULL }'; utilizzata per unirsi alla lista.

name

Nome del protocollo. Questo è il nome che l'utente conoscerà; in genere, per l'auto-caricamento nello userspace, è meglio se corrisponde ad uno dei nomi canonici di '/etc/protocols', come vedremo più avanti.

protonum

Numero del protocollo; vedere '/etc/protocols'.

manip_pkt

Questa è l'altra metà della funzione pkt_to_tuple del connection tracking: si può pensare ad essa come a tuple_to_pkt. Ci sono comunque alcune differenze: si ottiene un puntatore all'inizio dell'intestazione IP e la lunghezza totale del pacchetto. Ciò perché alcuni protocolli (UDP, TCP) necessitano di conoscere l'intestazione IP. Si fornirà il campo ip_nat_tuple_manip della tupla (ossia, il campo src), piuttosto che l'intera tupla, e il tipo di manipolazione che si sta per effettuare.

in_range

Questa funzione è utilizzata per indicare se la parte manipolabile della tupla fornita appartiene all'intervallo dato. Questa funzione è un po' complicata: si sta per fornire il tipo di manipolazione che è stata applicata alla tupla, la quale ci dice come interpretare l'intervallo (ci stiamo rivolgendo all'intervallo sorgente o a quello destinazione?).

Questa funzione è utilizzata per controllare se un mapping esistente ci colloca nell'intervallo corretto, e inoltre per controllare se non è necessaria una manipolazione.

unique_tuple

Questa funzione è il nocciolo del NAT: data una tupla e un intervallo, si sta per alterare la parte della tupla relativa al protocollo per sistemarla nell'intervallo, e renderla unica. Se non si riesce a trovare una tupla non utilizzata nell'intervallo, deve restituire 0. Si ottiene inoltre un puntatore alla struttura conntrack, richiesta dalla ip_nat_used_tuple().

L'approccio comune è di iterare semplicemente la parte della tupla relativa al protocollo attraverso l'intervallo, utilizzando 'ip_nat_used_tuple()' fino a quando una non restituisce false.

Si noti che il caso mapping nullo è già stato controllato: o è esterno all'intervallo dato o è già occupato.

Se IP_NAT_RANGE_PROTO_SPECIFIED non è impostato, ciò significa che l'utente sta effettuando il NAT, non il NAPT: sta facendo qualcosa di ragionevole con l'intervallo. Se il mapping non è desiderabile (per esempio, entro TCP, un mapping sulla destinazione non dovrebbe modificare la porta TCP a meno che non sia ordinato) deve restituire 0.

print

Dato un buffer di caratteri, una tupla match e una maschera, mostra per esteso le parti relative al protocollo e ritorna la lunghezza del buffer utilizzato.

print_range

Dato un buffer di caratteri e un intervallo, stampa per esteso la parte relativa al protocollo dell'intervallo e restituisce la lunghezza del protocollo utilizzato. Questa non sarà chiamata se il flag IP_NAT_RANGE_PROTO_SPECIFIED non è stato impostato per l'intervallo.

4.4.3 Nuovi target NAT

Questa è la parte davvero interessante. Si possono scrivere nuovi target NAT che provvedano un nuovo tipo di mapping. Due extra target sono forniti nel pacchetto di default: MASQUERADE e REDIRECT. Questi sono abbastanza semplici per illustrare il potenziale e la capacità di realizzare un nuovo target NAT.

Queste sono realizzate come qualsiasi altro target di iptables, internamente essi estraggono la connessione e chiamano 'ip_nat_setup_info()'.

4.4.4 Protocol helper

I protocol helper per il connection tracking permettono al codice del connection tracking di comprendere i protocolli che usano connessioni multiple (es. FTP). Inoltre segnano le connessioni 'figlie' come correlate alla connessione iniziale, solitamente leggendo il relativo indirizzo del flusso di dati.

I protocol helper per il NAT fanno due cose: innanzitutto permettono al NAT di manipolare il flusso di dati per cambiarne l'indirizzo, ed in secondo luogo di eseguire il NAT sulle connessioni correlate all'originale.

4.4.5 Moduli helper per il connection tracking

Descrizione Il compito del modulo per il connection tracking è quello di stabilire quali pacchetti appartengono ad una connessione già stabilita. Per farlo il modulo esegue i seguenti passi:

- Dice a netfilter quali pacchetti interessano al modulo (la maggior parte degli helper opera su una particolare porta).
- Registra una funzione con netfilter. Questa funzione è chiamata per ogni pacchetto che corrisponde ai criteri precedentemente espressi.
- Una funzione 'ip_conntrack_expect_related()' che può essere chiamata per dire a netfilter di aspettarsi connessioni correlate.

Se ci sono azioni aggiuntive che devono essere fatte la prima volta che un pacchetto, appartenente ad una connessione prevista, arriva, il modulo può registrare una funzione callback che sarà chiamata in quel momento.

Strutture e funzioni disponibili La funzione init del proprio modulo deve chiamare 'ip_conntrack_helper_register()' con un puntatore a 'struct ip_conntrack_helper'. Questa struttura ha i seguenti campi:

list

Questa è la testa di una lista linkata. Netfilter gestisce questa lista internamente. Da inizializzare con '{ NULL, NULL }'.

name

Questo è un puntatore ad una stringa costante che specifica il nome del protocollo. (ftp, irc, ...)

flags

Un gruppo con una o più delle seguenti flag:

- IP_CT_HELPER_F_REUSE_EXPECTRiusa le expectation se il limite (vedi 'max_expected') è raggiunto.

me

Un puntatore al modulo dell'helper. Da inizializzare con la macro 'THIS_MODULE'.

max_expected

Massimo numero di expectation non confermate (in sospeso).

timeout

Timeout (in secondi) per ogni expectation non confermata. Una expectation è cancellata 'timeout' secondi dopo che l'expectation identificata con la funzione 'ip_conntrack_expect_related()'.

tuple

Questa è una 'struct ip_conntrack_tuple' che specifica a quali pacchetti il modulo per il conntrack helper è interessato.

mask

Maschera riferita a 'struct ip_conntrack_tuple'. Questa maschera specifica quali bit di `tuple` sono validi.

help

La funzione che netfilter dovrebbe chiamare per ogni pacchetto che verifica tuple+mask

Esempio schematico di un conntrack helper module

```
#define FOO_PORT        111

static int foo_expectfn(struct ip_conntrack *new)
{
        /* funzione chiamata quando il primo pacchetto di una connessione
           prevista arriva */

        return 0;
}

static int foo_help(const struct iphdr *iph, size_t len,
                struct ip_conntrack *ct,
                enum ip_conntrack_info ctinfo)
{
        /* analizza i dati passati con questa connessione e
           decide come saranno i pacchetti correlati */

        /* aggiorna i dati privati della connessione master
           (session state, ...) */
        ct->help.ct_foo_info = ...

        if (there_will_be_new_packets_related_to_this_connection)
        {
                struct ip_conntrack_expect exp;
```

```
                memset(&exp, 0, sizeof(exp));
                exp.t = tuple_specifying_related_packets;
                exp.mask = mask_for_above_tuple;
                exp.expectfn = foo_expectfn;
                exp.seq = tcp_sequence_number_of_expectation_cause;

                /* dati privati della connessione slave */
                exp.help.exp_foo_info = ...

                ip_conntrack_expect_related(ct, &exp);
        }
        return NF_ACCEPT;
}

static struct ip_conntrack_helper foo;

static int __init init(void)
{
        memset(&foo, 0, sizeof(struct ip_conntrack_helper);

        foo.name = "foo";
        foo.flags = IP_CT_HELPER_F_REUSE_EXPECT;
        foo.me = THIS_MODULE;
        foo.max_expected = 1;   /* una expectation alla volta */
        foo.timeout = 0;        /* le expectation non terminano */

        /* siamo interessati a tutti i pacchetti TCP con porta di destinazione 111 */
        foo.tuple.dst.protonum = IPPROTO_TCP;
        foo.tuple.dst.u.tcp.port = htons(FOO_PORT);
        foo.mask.dst.protonum = 0xFFFF;
        foo.mask.dst.u.tcp.port = 0xFFFF;
        foo.help = foo_help;

        return ip_conntrack_helper_register(&foo);
}

static void __exit fini(void)
{
        ip_conntrack_helper_unregister(&foo);
}
```

4.4.6 Moduli helper per il NAT

Descrizione I moduli helper per il NAT, gestiscono il NAT per alcune specifiche applicazioni. Di solito questo include la manipolazione al volo dei dati: si pensi al comando PORT dell'FTP, dove il client comunica al server a quale IP/porta connettersi. Per questo un modulo helper per l'FTP deve sostituire l'IP/porta dopo che il comando PORT è stato eseguito in una connessione FTP.

Se stiamo trattando con il protocollo TCP, la faccenda si complica leggermente. La ragione sta nella possibile variazione delle dimensioni del pacchetto (esempio FTP: la lunghezza della stringa che rappresenta la tupla IP/porta dopo che il comando PORT l'ha modificata). Se cambiamo le dimensioni del pacchetto, abbiamo un syn/ack di differenza tra la parte sinistra e destra del NAT. (ad esempio se estendiamo un pacchetto di 4 ottetti, bisogna aggiungere questo offset al numero di sequenza TCP di ogni pacchetto che seguirà).

E' anche richiesta la gestione del NAT di tutti i pacchetti correlati. Prendiamo come esempio ancora FTP, dove tutti i pacchetti in ingresso di una connessione DATA devono essere NATtati verso l'IP/porta specificati dal client con il comando PORT nella connessione di controllo, piuttosto che passare attraverso la normale tabella lookup.

- funzione callback per i pacchetti che producono una connessione correlata (foo_help)
- funzione callback per tutti i pacchetti correlati (foo_nat_expected)

Strutture e funzioni disponibili La funzione 'init()' del modulo helper chiama 'ip_nat_helper_register()' con un puntatore ad una 'struct ip_nat_helper'. Questa struttura ha i seguenti elementi:

list

Ancora un altro header di una lista usata internamente da netfilter. Da inizializzare con { NULL, NULL }.

name

Un puntatore ad una stringa costante con il nome del protocollo

flags

Un gruppo di zero, una o più delle seguenti flag:

- IP_NAT_HELPER_F_ALWAYSChiama il NAT helper per ogni pacchetto, non solo per i pacchetti che il conntrack ha riconosciuto come causa di expectation.
- IP_NAT_HELPER_F_STANDALONEDice al nucleo del NAT che questo protocollo non ha un conntrack helper, ma solo un NAT helper.

me

Un puntatore al modulo dell'helper. Da inizializzare usando la macro 'THIS_MODULE'.

tuple

una 'struct ip_conntrack_tuple' che descrive a quali pacchetti in NAT helper è interessato.

mask

maschera una 'struct ip_conntrack_tuple', che dice a netfilter quali bit di `tuple` sono validi.

help

La funzione che è chiamata per ogni pacchetto che verifica tuple+mask.

expect

La funzione che è chiamata per ogni primo pacchetto di una connessione prevista.

E' molto simile alla scrittura di un connection tracking helper.

Esempio di un modulo helper per il NAT

```
#define FOO_PORT        111

static int foo_nat_expected(struct sk_buff **pksb,
                        unsigned int hooknum,
                        struct ip_conntrack *ct,
                        struct ip_nat_info *info)
```

```
/* chiamata ogni volta che arriva il primo pacchetto di una connessione correlata.
   parametri:   pksb    packet buffer
                hooknum HOOK di provenienza (POST_ROUTING, PRE_ROUTING)
                ct      informazioni su questa connessione (correlata)
                info    &ct->nat.info
   valore di ritorno: verdetto (NF_ACCEPT, ...) */
{
        /* Cambio dei valori ip/porta del pacchetto con i valori
           letti da master->tuplehash, per mapparlo allo stesso modo,
           viene chiamata ip_nat_setup_info e restituito NF_ACCEPT. */

}

static int foo_help(struct ip_conntrack *ct,
                    struct ip_conntrack_expect *exp,
                    struct ip_nat_info *info,
                    enum ip_conntrack_info ctinfo,
                    unsigned int hooknum,
                    struct sk_buff  **pksb)
/* chiamata per ogni pacchetto riconosciuto dal conntrack come expectation-cause
   parametri:   ct      struct ip_conntrack della connessione principale
                exp     struct ip_conntrack_expect dell'expectation
                        determinata dal conntrack helper per questo protocollo
                info    (STATO: related, new, established, ... )
                hooknum HOOK di provenienza (POST_ROUTING, PRE_ROUTING)
                pksb    packet buffer
*/
{

        /* estrazione di informazioni sui futuri pacchetti correlati
           (è possibile condividere informazioni con la funzione foo_help
           del connection tracking).
           Scambio di indirizzo/porta con i valori per il masquerading,
           inserimento della tupla dei pacchetti correlati */
}

static struct ip_nat_helper hlpr;

static int __init(void)
{
        int ret;

        memset(&hlpr, 0, sizeof(struct ip_nat_helper));
        hlpr.list = { NULL, NULL };
        hlpr.tuple.dst.protonum = IPPROTO_TCP;
        hlpr.tuple.dst.u.tcp.port = htons(FOO_PORT);
        hlpr.mask.dst.protonum = 0xFFFF;
        hlpr.mask.dst.u.tcp.port = 0xFFFF;
        hlpr.help = foo_help;
        hlpr.expect = foo_nat_expect;

        ret = ip_nat_helper_register(hlpr);

        return ret;
}
```

```
static void __exit(void)
{
        ip_nat_helper_unregister(&hlpr);
}
```

4.5 Comprendere Netfilter

Netfilter è piacevolmente semplice, ed è stato descritto in modo abbastanza esauriente nei capitoli precedenti. Tuttavia, qualche volta è necessario andare oltre a ciò che l'infrastruttura NAT o ip_tables offrono, oppure si potrebbe desiderare di sostituirle interamente.

Un importante problema per netfilter (beh, in futuro) è il caching. Ogni skb ha un campo 'nfcache': una bitmask che indica quali campi dell'intestazione sono stati esaminati e se i pacchetti sono stati alterati o no. L'idea è che ciascun hook fuori di netfilter effettui un OR con i bit rilevanti, in questo modo si potrà successivamente realizzare un sistema cache sufficientemente intelligente da comprendere quando i pacchetti non necessitano di essere passati attraverso netfilter.

I bit più importanti sono NFC_ALTERED, che specifica che il pacchetto è stato alterato (questo è già utilizzato per l'hook IPv4 NF_IP_LOCAL_OUT, per re-instradare i pacchetti alterati), e NFC_UNKNOWN, che indica che il caching non dovrebbe essere effettuato in quanto sono state esaminate alcune proprietà non esprimibili. Se incerti, semplicemente si imposti il flag NFC_UNKNOWN nel campo nfcache della skb all'interno del proprio hook.

4.6 Realizzare nuovi moduli Netfilter

4.6.1 Introduzione agli hook di Netfilter

Per ricevere/manipolare i pacchetti nel kernel, si può semplicemente scrivere un modulo che registri un hook netfilter. Questa è sostanzialmente un'espressione di interesse per alcuni determinati punti; gli attuali punti sono specifici per protocollo, e sono definiti nelle intestazioni di netfilter specifiche per i protocolli, ad esempio netfilter_ipv4.h.

Per registrare e rimuovere le registrazioni di hook di netfilter, si utilizzeranno le funzioni 'nf_register_hook' e 'nf_unregister_hook'. Ciascuna di queste richiede un puntatore ad una 'struct nf_hook_ops' che si dovrà popolare come segue:

list

Utilizzata per unirsi alla lista linkata: impostare a '{ NULL, NULL }'

hook

Funzione invocata quando un pacchetto colpisce questo hook. La funzione deve restituire NF_ACCEPT, NF_DROP oppure NF_QUEUE. Nel caso NF_ACCEPT, sarà chiamato il successivo hook agganciato a questo punto. Nel caso NF_DROP, il pacchetto sarà scartato. Nel caso NF_QUEUE, sarà accodato. Si riceverà inoltre un puntatore ad un puntatore skb, perciò si può sostituire completamente la skb, se desiderato.

flush

Al momento non utilizzata: progettata per far passare i pacchetti giunti quando la cache viene svuotata. Forse non sarà mai implementata: impostare a NULL.

pf

La famiglia del protocollo, es. nel caso IPv4, 'PF_INET'.

hooknum

Numero dell'hook a cui si è interessati, es. 'NF_IP_LOCAL_OUT'.

4.6.2 Processare i pacchetti accodati

Questa interfaccia è al momento utilizzata da ip_queue; ci si può registrare per gestire, per un dato protocollo, i pacchetti accodati. Ha una semantica simile a quella delle registrazioni di un hook, eccetto il fatto che è possibile bloccare il trattamento del pacchetto, inoltre si vedranno solo i pacchetti per i quali un hook ha risposto con un 'NF_QUEUE'.

Le due funzioni utilizzate per registrare l'interesse ai pacchetti accodati sono 'nf_register_queue_handler()' e 'nf_unregister_queue_handler()'. La funzione che si registrerà sarà chiamata con il puntatore 'void *' che poi si passerà alla 'nf_register_queue_handler()'.

Se nessun altro è registrato per gestire un protocollo, restituire NF_QUEUE è equivalente a restituire NF_DROP.

Una volta registrato l'interesse ai pacchetti accodati, essi cominciano ad essere accodati. Si può fare qualsiasi cosa con essi, ma è obbligatorio chiamare 'nf_reinject()' una volta terminato (non si effettui semplicemente un kfree_skb()). Quando si effettua il reinject di skb, si passi la skb, la 'struct nf_info' gestore della queue fornita e un verdetto: con NF_DROP i pacchetti vengono scartati, NF_ACCEPT fa sì che continuino ad iterare attraverso gli hook, NF_QUEUE che siano nuovamente accodati, e NF_REPEAT che l'hook che ha accodato i pacchetti sia nuovamente consultato (si evitino i loop infiniti).

Si può guardare all'interno della 'struct nf_info' per ottenere informazioni ausiliarie sul pacchetto, quali ad esempio interfacce e hook.

4.6.3 Ricevere comandi dallo Userspace

E' cosa comune che componenti di netfilter vogliano interagire con lo userspace. Il metodo affinché ciò avvenga richiede il meccanismo setsockopt. Nota che ogni protocollo deve essere modificato per poter chiamare nf_setsockopt() per i numeri setsockopt che non comprende (e nf_getsockopt() per i numeri getsockopt), finora solo IPv4, IPv6 e DECnet sono stati modificati.

Utilizzando una tecnica ora familiare, si registrerà una 'struct nf_sockopt_ops' utilizzando nf_register_sockopt(). I campi di questa struttura sono i seguenti:

list

Utilizzata per unirsi alla lista. Impostare a '{ NULL, NULL }'.

pf

La famiglia del protocollo che si gestisce, es. PF_INET.

set_optmin

e

set_optmax

Questi specificano l'intervallo (esclusivo) di numeri setsockopt gestiti. Quindi utilizzare 0 e 0 significa non avere numeri setsockopt.

set

Questa è la funzione chiamata quando l'utente richiama una delle setsockopts. Si dovrebbe controllare che esse abbiano capacità NET_ADMIN entro questa funzione.

get_optmin
e

get_optmax
Questi specificano l'intervallo (esclusivo) dei numeri setsockopt gestiti. Quindi utilizzare 0 e 0 significa non avere numeri setsockopt.

get
Questa è la funzione chiamata quando l'utente richiama una delle getsockopts. Si dovrebbe controllare che esse abbiano capacità NET_ADMIN entro questa funzione.

Gli ultimi due campi sono utilizzati internamente.

4.7 Gestione del pacchetto nello userspace

Utilizzando la libreria libipq e il modulo 'ip_queue', quasi tutto ciò che può essere fatto nel kernel può ora essere effettuato nello userspace. Ciò significa che, con qualche penalità nella velocità, si può sviluppare il proprio codice interamente nello userspace. A meno che non si stia provando a filtrare bande larghe, si dovrebbe trovare questo approccio superiore al manipolamento del pacchetto nel kernel.

Nei primi giorni di vita di netfilter ho constatato ciò portando una versione embrionale di iptables nello userspace. Netfilter apre le porte a tutte le persone che vogliono scrivere per conto proprio moduli efficienti di manipolazione della rete, e in qualsiasi linguaggio desiderato.

5 Portare moduli di filtraggio dei pacchetti da 2.0 e 2.2

Si dia un'occhiata al file ip_fw_compat.c per un semplice esempio che dovrebbe rendere i porting piuttosto semplici.

6 Gli hook di Netfilter per realizzare un tunnel

Gli autori dei driver per il tunneling (o incapsulamento) dovrebbero seguire due semplici regole per il kernel 2.4 (come fanno i driver nel kernel, ad esempio net/ipv4/ipip.c):

- Rilasciare skb->nfct se si sta per rendere il pacchetto irriconoscibile (es. decapsulare/incapsulare). Non c'è bisogno di fare questo se lo si sta mettendo in un *nuovo* skb.

  Altrimenti il NAT userà le vecchie informazioni del connection tracking per modificare il pacchetto, con pericolose conseguenze.

- Bisogna essere sicuri che i pacchetti incapsulati passino attraverso l'hook LOCAL_OUT, e quelli decapsulati attraverso PRE_ROUTING (la maggior parte dei tunnel usano ip_rcv(), che fa proprio questo).

  Altrimenti l'utente non sarà in grado di filtrare come si aspetta da un tunnel.

Il modo tradizionale per fare la prima cosa consiste nell'inserire le seguenti linee di codice prima di wrappare o unwrappare il pacchetto:

```
        /* Dice al framework netfilter che questo pacchetto non è lo
           stesso di prima!*/
#ifdef CONFIG_NETFILTER
        nf_conntrack_put(skb->nfct);
        skb->nfct = NULL;
#ifdef CONFIG_NETFILTER_DEBUG
        skb->nf_debug = 0;
#endif
#endif
```

Solitamente, tutto quello che bisogna fare per il secondo accorgimento, è trovare dove il pacchetto appena incapsulato va dentro ip_send(), e sostiruirlo con qualcosa tipo:

```
        /* Invia il "nuovo" pacchetto dall'host locale */
        NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev, ip_send);
```

Seguendo queste regole la persona che imposta imposta le regole per il packet filtering sulla tunnel-box vedrà qualcosa del genere per un pacchetto sottoposto a tunneling:

1. FORWARD hook: pacchetto normale (da eth0 -> tunl0)
2. LOCAL_OUT hook: pacchetto incapsulato (verso eth1).

E per il pacchetto di risposta:

1. LOCAL_IN hook: pacchetto di risposta incapsulato (da eth1)
2. FORWARD hook: pacchetto di risposta (da eth1 -> eth0).

7 La suite per i test

Nel CVS è presente una suite per i test: più test la suite gestisce, e maggiore sarà la certezza che dei cambiamenti al codice non abbiano silenziosamente corrotto qualcosa. Test banali sono importanti quanto quelli più ingegnosi: sono i test banali che semplificano i test complessi (ci si assicuri che le basi funzionino correttamente prima di eseguire i test complessi).

I test sono semplici: sono giusto degli script shell presenti nella sotto-directory testsuite/ che si suppone abbiano successo. Gli script sono eseguiti in ordine alfabetico, quindi '01test' sarà eseguito prima di '02test'. Correntemente ci sono 5 directory di test:

00netfilter/
: test generali riguardanti il framework netfilter

01iptables/
: test riguardanti iptables

02conntrack/
: test riguardanti il connection tracking

03NAT/
: test riguardanti il NAT

04ipchains-compat/

test riguardanti la compatibilità ipchains/ipfwadm

All'interno della directory testsuite/ è presente uno script 'test.sh'. Esso configura due semplici interfacce (tap0 e tap1), abilita il forwarding, e rimuove tutti i moduli di netfilter. Quindi esegue da ciascuna directory ogni script test.sh fino a quando uno fallisce. Questo script ha due argomenti opzionali: '-v' che specifica di visualizzare ogni test processato e un nome opzionale di test: se è fornito, lo script salterà tutti i test fino a trovare quello specificato.

7.1 Realizzare un test

Si crei un nuovo file in una directory appropriata: si provi a numerare il proprio test così sarà eseguito al momento opportuno. Ad esempio, allo scopo di effettuare il test del tracciamento delle risposte ICMP (02conntrack/02reply.sh) è necessario innanzitutto controllare che i pacchetti ICMP uscenti siano tracciati correttamente (02conntrack/01simple.sh).

Solitamente è meglio creare più file di piccole dimensioni, ciascuno dei quali si occupi di una sola area, ciò aiuta le persone che eseguono la testsuite ad isolare immediatamente i problemi.

Se qualcosa non funziona durante il test, semplicemente si effettui un 'exit 1', il quale causa un fallimento; se riguarda qualcosa che si aspettava fallisse, si potrebbe stampare un messaggio unico. I propri test dovrebbero concludersi con 'exit 0' se tutto è stato eseguito correttamente. E' necessario controllare che **tutti** i comandi siano stati eseguiti con successo, utilizzando 'set -e' all'inizio dello script oppure appendendo '|| exit 1' alla fine di ciascun comando.

Le funzioni di aiuto 'load_module' e 'remove_module' possono essere utilizzate per caricare i moduli: con la testsuite non si dovrebbe mai contare sull'auto-caricamento a meno che non sia proprio quello che si desidera specificatamente verificare.

7.2 Variabili e ambiente

Si hanno due interfacce in gioco: tap0 e tap1. I loro indirizzi sono rispettivamente nelle variabili `$TAP0` e `$TAP1`. Entrambe hanno netmask 255.255.255.0; le loro reti sono rispettivamente in $TAP0NET e $TAP1NET.

E' presente un file temporaneo vuoto in $TMPFILE. Esso è cancellato al termine del proprio test.

Lo script sarà eseguito dalla directory testsuite/, se presente. Quindi si può accedere ai tool (quali iptables) utilizzando un path che cominci con '../userspace'.

Lo script può visualizzare maggiori informazioni se $VERBOSE è impostata (si intende che l'utente specifichi '-v' dalla linea comandi).

7.3 Tool utili

Ci sono parecchi tool utili nella sotto-directory tools: ciascuno esce ritornando uno stato non zero se ha riscontrato un problema.

7.3.1 gen_ip

Si possono generare pacchetti IP utilizzando 'gen_ip', il quale emette un pacchetto IP verso lo standard input. Si possono alimentare di pacchetti tap0 e tap1 inviando lo standard output verso /dev/tap0 e /dev/tap1 (questi sono creati subito dopo la prima esecuzione della testsuite, se non già esistenti).

gen_ip è un programma semplice che è al momento piuttosto pignolo riguardo l'ordine degli argomenti. Prima di tutto richiede gli argomenti generali opzionali:

FRAG=offset,length

Genera il pacchetto, quindi lo converte in un frammento utilizzando i parametri offset e lenght forniti.

MF

Imposta il bit 'More Fragments'.

MAC=xx:xx:xx:xx:xx:xx

Imposta l'indirizzo sorgente MAC.

TOS=tos

Imposta il campo TOS del pacchetto (da 0 a 255).

Seguono gli argomenti obbligatori:

source ip

Indirizzo IP sorgente del pacchetto.

dest ip

Indirizzo IP destinazione del pacchetto.

length

Lunghezza totale del pacchetto, intestazioni incluse.

protocol

Numero del protocollo del pacchetto, es. 17 = UDP.

Poi gli argomenti dipendono dal protocollo: nel caso UDP (17), essi consistono nei numeri di porta sorgente e destinazione. Nel caso ICMP (1), essi consistono nel tipo e nel codice del messaggio ICMP: se il tipo è 0 oppure 8 (ping-reply o ping) allora sono richiesti altri due argomenti (i campi ID e sequence). Nel caso TCP sono richiesti la porta sorgente, la porta destinazione e i flag (SYN, SYN/ACK, ACK, RST oppure FIN). Ci sono tre argomenti opzionali: OPT= seguito da una lista di opzioni separate da virgole, SYN= seguito da un numero di sequenza e ACK seguito anch'esso da un numero di sequenza. Infine, l'argomento opzionale DATA specifica che il carico del pacchetto TCP è da riempire con il contenuto dello standard input.

7.3.2 rcv_ip

Si possono vedere i pacchetti IP utilizzando 'rcv_ip', il quale visualizza la linea comandi il più possibile corrispondente con i valori originali dati a gen_ip (i frammenti sono l'eccezione).

Ciò è estremamente utile per l'analisi dei pacchetti. Richiede due argomenti obbligatori:

wait time

Il tempo massimo di attesa, espresso in secondi, per un pacchetto proveniente dallo standard input.

iterations

Numero di pacchetti da ricevere.

C'è inoltre un argomento opzionale DATA che causa la visualizzazione, dopo l'intestazione del pacchetto, del contenuto di un pacchetto TCP sullo standard output.

La modalità di utilizzo di 'rcv_ip' in uno script shell è la seguente:

```
# Imposta il controllo, in questo modo si può utilizzare & negli script shell
set -m

# Attendi due secondi per un pacchetto proveniente da tap0
../tools/rcv_ip 2 1 < /dev/tap0 > $TMPFILE &

# Assicurati che rcv_ip sia in funzione
sleep 1

# Invia un ping
../tools/gen_ip $TAP1NET.2 $TAP0NET.2 100 1 8 0 55 57 > /dev/tap1 || exit 1

# Attendi rcv_ip,
if wait %../tools/rcv_ip; then :
else
    echo rcv_ip failed:
    cat $TMPFILE
    exit 1
fi
```

7.3.3 gen_err

Questo programma prende un pacchetto (come generato da gen_ip, ad esempio) dallo standard input e lo rigira in un errore ICMP.

Richiede tre argomenti: un indirizzo IP sorgente, un tipo e un codice. L'IP di destinazione sarà impostato utilizzando l'indirizzo IP sorgente del pacchetto dato allo standard input.

7.3.4 local_ip

Questo prende un pacchetto dallo standard input e lo immette nel sistema da un raw socket. Ciò consente di dare l'apparenza di un pacchetto generato localmente (come separato dal pacchetto fornito ad uno dei dispositivi ethertap, sembra quindi un pacchetto generato in remoto).

7.4 Consigli vari

Tutti i tool assumono di poter fare qualsiasi cosa in una lettura o scrittura: ciò è vero per i dispositivi ethertap, ma potrebbe non essere vero se si sta facendo qualcosa di complicato con le pipe.

dd può essere utilizzato per tagliare i pacchetti: dd ha un'opzione obs (output block size) che può essere usata per produrre in output il pacchetto in una singola scrittura.

Si effettui prima di tutto il test per successo: ad esempio per verificare se i pacchetti sono bloccati con successo, prima si testi se i pacchetti passano normalmente **poi** che alcuni siano bloccati. In caso contrario un problema non correlato potrebbe fermare i pacchetti ...

Si cerchi di scrivere test corretti, non del tipo ‘provare in modo casuale e vedere cosa accade’. Se un test corretto fallisce, ciò rappresenta un’ottima cosa da sapere. Se invece un test casuale fallisce non è di grande aiuto.

Se un test fallisce senza ritornare un messaggio, si può aggiungere un ‘-x’ alla prima riga dello script (es. ‘#! /bin/sh -x’) per vedere quali comandi sono stati eseguiti.

Se un test fallisce di tanto in tanto, si controllino eventuali interferenze casuali nel traffico di rete (si provi a disabilitare tutte le proprie interfacce esterne). Stando nella stessa rete di Andrew Tridgell, ad esempio, tendo ad essere assillato dai broadcast di Windows.

8 Motivazione

Come sviluppatore di ipchains ho realizzato (in uno di quei momenti di flash-abbaglianti-mentre-attendi-di-entrare in un ristorante cinese a Sidney) che il filtraggio dei pacchetti era effettuato nel posto sbagliato. Non riesco a trovarla ora, ma ricordo una lettera inviata ad Alan Cox, che gentilmente rispondeva ‘perché prima di tutto non termini quello che stai facendo, probabilmente è la cosa giusta’. In parole povere, pragmatismo doveva prevalere su La Cosa Giusta.

Dopo aver terminato ipchains, che inizialmente doveva essere una modifica minore della parte del kernel riguardante ipfwadm, diventata poi una consistente riscrittura, e aver scritto l’HOWTO, mi sono reso conto di quanta confusione esistesse nella vasta comunità di Linux a riguardo delle questioni quali filtraggio dei pacchetti, mascheramento, port forwarding e così via.

Questa è la soddisfazione di fornire il proprio supporto: ottieni una stretta percezione su cosa gli utenti cercano di fare, e con che cosa si trovano a lottare. Il free software per lo più è ricompensato quando è nelle mani della maggior parte degli utenti (questo è il punto, giusto?), e ciò consente poi di poterlo rendere migliore. L’architettura, non la documentazione, è la chiave per risolvere i problemi.

Quindi avevo esperienza, per quanto riguardava il codice di ipchains, e una buona idea su cosa le persone volevano fare. Esistevano solo due problemi.

Primo, non volevo tornare indietro sulla sicurezza. Essere un consulente sulla sicurezza è un tiro alla fune costante e morale tra la coscienza e il portafogli. Ad un livello di principio si vende la percezione della sicurezza, la quale è in discordia con l’attuale sicurezza. Forse lavorare nel campo militare, dove si comprende la sicurezza, potrebbe essere differente.

Il secondo problema è che i nuovi utenti non sono l’unica preoccupazione; un numero crescente di compagnie e ISP utilizzano queste funzionalità. C’era quindi la necessità di un input fidato proveniente da queste classi di utenti se si desiderava poi scalare verso gli utenti casalinghi.

Questi problemi sono stati risolti quando mi sono imbattuto in David Bonn, di fama WatchGuard, allo Usenix nel Luglio 1998. Stavano cercando un coder del kernel Linux; alla fine concordarono di indirizzarmi per un mese ai loro uffici di Seattle per vedere se si poteva elaborare un accordo in cui loro si sarebbero impegnati a sponsorizzare il mio nuovo codice e il mio sforzo per il supporto. La cifra concordata fu maggiore di quanto aspettato, perciò non ottenni un taglio dello stipendio. Ciò significa che non ho più da pensare a consulenze esterne per un po’.

L’esposizione alla WatchGuard mi portava all’esposizione a quei grandi clienti di cui avevo bisogno, e l’indipendenza da loro mi permetteva di supportare tutti gli utenti (es. concorrenti della WatchGuard) in modo eguale.

Avrei potuto quindi sviluppare netfilter con comodità, portare ipchains al di sopra, ed essere soddisfatto. Sfortunatamente, il codice di masquerading sarebbe comunque rimasto nel kernel: rendere il masquerading indipendente dal filtraggio è uno dei punti più importanti nel momento in cui si sposta il filtro dei pacchetti, ma per fare ciò è necessario portare anche il masquerading al di sopra del framework netfilter.

La mia esperienza con la funzionalità 'interface-address' di ipfwadm (rimossa con ipchains) mi aveva insegnato che non c'era alcuna speranza di togliere il codice del masquerading e di attendere che qualcuno, che ne avesse bisogno, realizzasse un porting al di sopra di netfilter al posto mio.

Perciò avevo bisogno di avere almeno tante funzionalità quante il codice corrente; preferibilmente qualcuna in più, per incoraggiare utenti di nicchia ad adottarlo. Ciò significava rimpiazzare il proxy trasparente (volentieri!), masquerading e port forwarding. In altre parole, un completo strato NAT.

Anche se avevo deciso di portare lo strato esistente del masquerading, invece di scrivere un sistema NAT generico, il codice del masquerading ormai mostrava già i segni dell'età, e mancanza di manutenzione. Non c'era un manutentore del masquerading e si vedeva. Sembra che gli utenti più seri non utilizzino affatto il masquerading, e inoltre non ci sono molti utenti casalinghi disponibili alla manutenzione. Persone ottime come Juan Ciarlante avevano apportato correzioni, ma ormai si era arrivati ad uno stadio (essendo stato esteso più e più volte) che una riscrittura era davvero necessaria.

Prego notare che non ero la persona adatta ad effettuare una riscrittura del NAT: non utilizzavo più il masquerading, e non avevo studiato il codice esistente a suo tempo. Forse è questa la ragione per cui mi ha impegnato più a lungo di quanto previsto. Il risultato è comunque abbastanza buono, secondo la mia opinione, e assicuro che ho imparato davvero molto. Non dubito comunque che una seconda versione sarà migliore, una volta constatato come le persone la utilizzano.

9 Ringraziamenti

Ringrazio tutti coloro che sono stati di aiuto, specialmente Harald Welte per aver scritto il paragrafo sui protocol helper.