my-pcap

Andrea Cardaci

SGR — 2009

1 Caratteristiche

my-pcap cattura da una o tutte le interfacce disponibili fermandosi se interrotto oppure se è stato raggiunto il numero massimo di pacchetti specificati. Attualmente vengono analizzati solo i pacchetti TCP e UDP su IPv4 (il riconoscimento viene delegato a un filtro BPF). Le opzioni di cattura comprendono inoltre la modalità promiscua e il filtraggio in base alla direzione dei pacchetti.

1.1 Sniffer di pacchetti

La cosa più semplice che è possibile fare è il dump dei pacchetti intercettati mostrando alcune informazioni fondamentali come la sorgente e la destinazione.

1.2 Analizzatore di flussi

my-pcap può memorizzare i pacchetti catturati compattandoli in flussi monodirezionali. Un flusso contiene un certo numero di informazioni relative ai pacchetti che lo compongono, divise in:

1. chiave:
    - IP sorgente;
    - porta sorgente;
    - IP destinazione;
    - porta destinazione.
2. data:
    - numero di pacchetti;
    - dimensione in byte del traffico;
    - timestamp del primo pacchetto ricevuto;
    - timestamp dell'ultimo pacchetto ricevuto.

Quando un flusso è attivo/inattivo da troppo tempo viene considerato scaduto, estratto da my-pcap e restituito, ad esempio attraverso una stampa.

1.3 Statistiche

Durante la cattura my-pcap conta il numero di pacchetti che ha ricevuto facendo distinzione tra TCP e UDP. Questi dati vengono inseriti in un database round-robin (RRDtool) che ne tiene traccia per un certo periodo; è possibile disegnare un grafico dell'andamento: numero di pacchetti al secondo.

1.4 Interfaccia HTTP

Opzionalmente my-pcap può attivare un server HTTP (utilizzando libmicrohttpd) attraverso il quale mostrare in una pagina HTML gli ultimi flussi estratti e il suddetto grafico.

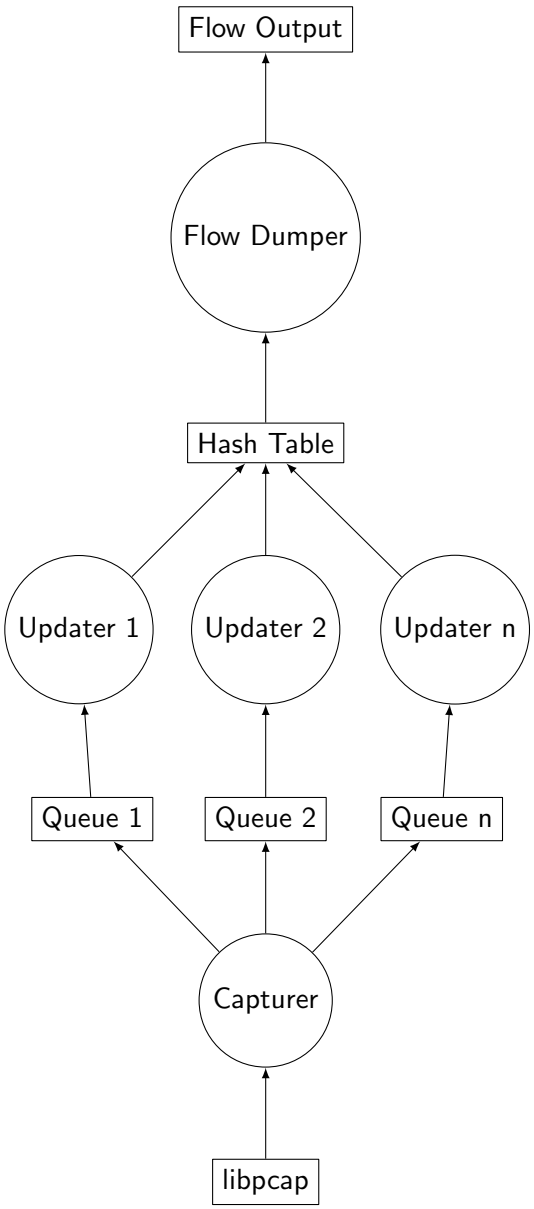

Figura 1: *Rappresentazione della struttura interna di `my-pcap`: i cerchi indicano i thread, mentre i rettangoli le risorse; le freccie possono essere interpretate come: "una risorsa è letta da un thread" e "un thread scrive su una risorsa".*

2 Struttura

In figura 1 è mostrata un'approssimazione della struttura interna di `my-pcap`; di seguito verrà illustrato il funzionamento dei vari thread.

2.1 Capturer

La cattura viene effettuata da un thread tramite la funzione `pcap_loop`. Il limite di un singolo thread è per evitare attese inutili su un mutex, che in caso contrario sarebbe necessario[1] a prevenire race conditions.

Il compito del thread `Capturer` è tuttavia limitato (oltre ovviamente a chiedere il prossimo pacchetto a `libpcap`) a calcolare l'hash del pacchetto (come mostrato in figura 2) e in base a questa inserirlo (non tutto il pacchetto ma solo le informazioni rilevanti, vedere § 1.2) nella coda appropriata.

Le code hanno lo scopo di dividere i pacchetti tra i vari thread `Updater` (uno per coda) al fine di garantire l'assenza di interferenze al momento dell'aggiornamento della tabella hash. In altre parole un

[1] a meno di meccanismi come `PF_RING`

Figura 2: *La hash è un intero (senza segno) a 32 bit formato dallo XOR di queste tre righe.*

thread dovrà poter accedere solamente ai propri slot senza mai interferire con quelli dei suoi colleghi. Questo è garantito dal modo in cui vengono smistati i pacchetti nelle code:

$$indice\_della\_coda = hash \ \% \ numero\_di\_threads$$

Queste code sono dei buffer circolari che vengono usati senza meccanismi di sincronizzazione, almeno dalla parte del `Capturer` che si limita a segnalare la presenza di pacchetti; i thread `Updater` aspettano quindi di essere segnalati per iniziare il loro ciclo.

Se il `Capturer` dopo aver scelto una coda, trova che questa è piena, scarta il pacchetto; per il motivo sopra descritto non può delegarlo a un'altra coda. Per evitare allocazioni di memoria non necessarie gli elementi delle code hanno un flag di uso.

Tutto questo può funzionare anche senza meccanismi di sincronizzazione perché su ogni coda agiscono esattamente due thread: il primo inserisce (e pone il flag a 'usato') solo se un elemento non è usato; il secondo rimuove (e pone il flag a 'non usato') solo se un elemento è usato. È importante notare che il flag viene modificato solo dopo aver effettuato l'azione.

2.2 Updater

Dopo aver estratto un pacchetto dalla propria coda, un thread `Updater` lo inserisce nella tabella hash; la posizione gli viene passata dal `Capturer` che, come già detto, ha utilizzato per scegliere la coda.

La tabella hash è implementata con liste di trabocco dove ogni elemento ha, tra le altre cose, un flag di uso; questo approccio è stato usato per limitare le allocazioni/deallocazioni dato che i flussi devono essere memorizzati solo per un tempo limitato, dopo il quale altri prenderanno il loro posto. Un altro motivo di questa scelta riguarda l'interferenza con il thread `Flow Dumper`: in questo modo è possibile fare a meno di meccanismi di sincronizzazione, seppur con qualche problema (vedere § 3).

L'esatta procedura di aggiornamento della tabella hash all'arrivo di un pacchetto (una volta individuata la lista di trabocco) è la seguente:

1. la lista viene scorsa fino a che non si trova l'elemento da aggiornare o fino alla fine della lista; nel mentre viene salvato l'eventuale primo elemento non utilizzato;

2. se l'elemento viene trovato allora viene controllato il suo flag di uso: se è attivo viene aggiornato, altrimenti resettato; se invece viene raggiunta la fine della lista (quindi il pacchetto richiede un nuovo flusso) le opzioni sono due: se nella ricerca è stato trovato un elemento non usato allora viene utilizzato, altrimenti un nuovo nodo viene allocato alla fine della lista.

2.3 Flow Dumper

Questo thread scorre ripetutamente la tabella hash in cerca di flussi che sono scaduti, quando ne trova uno lo stampa (o lo passa al server `HTTP` se è attivo) e commuta il flag in modo che il nodo possa essere riutilizzato in futuro.

2.4 Altri thread

Oltre a questi c'è un altro thread che si occupa della gestione dei segnali. Per aggiornare il database round-robin viene registrato un timer (`setitimer`) che a intervalli regolari invia un segnale (`SIGALRM`) al gestore, il quale tramite il comando `RRDtool` inserisce i valori nel database.

3 Problemi

Nel ridurre a zero le operazioni di sincronizzazione (lock, wait, etc.) sono occorsi alcuni problemi che tuttavia non compromettono la stabilità del programma:

1. non deallocare mai i nodi inutilizzati della tabella hash può portare a lungo andare a un rallentamento globale causato dall'aumento della lunghezza media delle liste di trabocco, tuttavia problemi di memoria non dovrebbero verificarsi perché in caso di impossibilità di allocare un nodo, il pacchetto viene semplicemente scartato; e comunque dato il concetto di flusso, nuovi nodi si libereranno in continuazione.
2. un thread `Updater` prima di aggiornare un elemento controlla lo stato del flag di uso, se tra queste due operazioni il `Flow Dumper` trova che il medesimo elemento è scaduto procede con il dump; il problema è che l'`Updater` lo aggiornerà comunque prolungando la durata del flusso, questo può portare nel migliore dei casi a due dump (entrambi corretti) riguardanti lo stesso flusso, mentre nel peggiore le informazioni che il `Flow Dumper` leggerà saranno state in parte aggiornate dall'`Updater` con lo scarto di un pacchetto.

4 Utilizzo

Eseguito senza parametri (per le opzioni vedere figura 3), `my-pcap` non mostra alcun output, ma tenta comunque di effettuare la cattura.

`my-pcap` può catturare da una qualsiasi delle interfacce di rete, enumerabili tramite l'opzione `l`. Una volta scelte le opzioni cattura (`ipnd`) occorre scegliere una o più opzioni di output (`rofw`, vedere § 1).

Per poter utilizzare il database round-robin è necessario prima crearne uno con l'opzione `c`, è importante notare che per effettuare la cattura è necessario essere super-utente, mentre il database dovrebbe essere creato da un utente normale dato che una volta iniziata la cattura i privilegi di `root` vengono abbandonati, creando dei problemi se il programma è stato eseguito tramite il comando `sudo` (vedere § 5).

L'interfaccia `HTTP` (`w`) per avere una qualche utilità deve essere usata insieme a una o entrambe le opzioni che abilitano la gestione dei flussi e delle statistiche.

5 Note

Seguono alcune considerazioni sul progetto:

- La scelta di non deallocare mai i nodi inutilizzati porta inevitabilmente, come già detto, a un calo delle prestazioni; al momento non è certo se convenga evitare allocazioni/deallocazioni di memoria oppure mantenere le liste di trabocco il più corte possibile. Possibili soluzioni includono:
  - l'uso di un mutex che garantisca la mutua esclusione tra un `Updater` e il `Flow Dumper`;
  - accorgimenti nel trattare gli elementi della lista al fine di non causare interferenza tra i thread, anche mediante l'eliminazione ritardata dei nodi.
- Per una questione di semplicità `my-pcap` supporta solamente `IPv4`, anche se il passaggio alla versione successiva non dovrebbe causare troppi problemi.
- La scelta di memorizzare i flussi in modo monodirezionale è dovuta al fatto che in questo modo la gestione dei flussi è più semplice.
- Le statistiche dividono i pacchetti solamente in `TCP` e `UDP`, sarebbe più interessante andare avanti e monitorare l'utilizzo dei servizi (`HTTP`, `DNS`, `FTP`, etc.); per far questo basta analizzare il payload del livello applicazione cercando di riconoscere un pattern e/o in base alle porte utilizzate.
- `my-pcap` fa uso di una quantità di parametri definiti nel codice tramite `#define` che ne regolano profondamente il funzionamento; sono inizializzati a valori che non sono necessariamente la scelta migliore ma piuttosto la scelta più comoda per testare il programma.

```
usage:
    my-pcap -h
    my-pcap -c <rrd>
    my-pcap -g <rrd>
    my-pcap [-i <int>] [-p] [-n <max>] [-d i | o]
            [-r <rrd>] [-o] [-f] [-w <port>]

 -h        : print these infos.
 -l        : list compatible devices.
 -c <out>  : create a rrd database in <out>.
 -g <rrd>  : create a sample graph of <rrd>.
 -i <int>  : read packets from <int>; 'any' by default.
 -p        : promiscuous mode.
 -n <max>  : read <max> packets and exit.
 -d i|o    : capture direction, input(I) or output(O);
             input and output by default.
 -r <out>  : fill a rrd database in <out>.
 -o        : dump some info on stdout.
 -f        : dump some net flow info on stdout or http.
 -w <port> : enable http server on port <port>.

Analyze TCP and UDP over IPv4 packages only.
Use SIGTERM or SIGINT to break the process.
```

Figura 3: *Risultato del comando 'my-pcap -h'.*

- La funzione hash (figura 2) fa il suo lavoro ma è probabile che con una funzione differente si ottengano risultati migliori.
- Come già accennato per catturare da `libpcap` occorre essere super-utente, quindi le possibilità sono due:

  1. essere `root` e quindi ogni file prodotto ha i medesimi privilegi;
  2. eseguire `my-pcap` tramite il comando `sudo`, in questo caso il programma se ne accorge (controllando la variabile d'ambiente `SUDO_USER`) e dopo aver richiesto il descrittore per la cattura ed eventualmente attivato l'interfaccia `HTTP` ripristina l'utente originale.

  Questa può forse non essere la scelta migliore in un contesto realistico, dove la cattura e l'analisi dei pacchetti spetterebbe ad un utente fittizio ad hoc, ma è senz'altro comoda in queste circostanze.
- L'utilità pratica dell'interfaccia `HTTP` per come è adesso è decisamente scarsa, serve solo a mostrare un altro modo, oltre la stampa su `stdout`, di esportare i dati ottenuti durante la cattura.
- Sempre a riguardo dell'interfaccia `HTTP`, sarebbe utile stabilire su quale indirizzo far partire il servizio, se non altro per evitare di catturare il traffico di chi accede all'interfaccia; questo è impossibile con la versione di `libmicrohttpd` usata, purtroppo quella più recente dà attualmente qualche problema.