eBPF_TCPFlow

sonda delle chiamate di sistema TCP accept() e TCP connect(), basata su eBPF

Salvatore Costantino

s.costantino5@studenti.unipi.it

Progetto gestione di reti 2017/2018

Sommario

PANORAMICA

Il programma sviluppato consente di organizzare in flussi, le informazioni riguardanti le chiamate di sistema connect() e accept() (**indirizzo sorgente**, **indirizzo destinazione** ecc.) relative al protocollo TCP.

Per tracciare le chiamate di sistema sopra citate, è stato necessario iniettare del codice **eBPF** all'interno del kernel.

Quindi, piuttosto che creare flussi di rete concentrandosi sulla cattura di pacchetti, vengono generati flussi intercettando le chiamate di sistema (accept() e connect()) effettuate dai vari processi a livello applicativo. Rispetto alla generazione di flussi basata su pacchetti di rete (come in **NetFlow**) possiamo individuare vantaggi e svantaggi dell'approccio usato dall'applicativo realizzato:

PROS

- **Basso overhead** nella gestione e creazione di flussi, poiché in generale il numero di pacchetti è maggiore (anche di molto) rispetto al numero delle chiamate di sistema in esame;
- **Tracciamento processi**. È possibile infatti sapere quale processo ha richiesto, o ha accettato, una qualche connessione con altri host remoti.

CONS

- **Impossibilità di estrarre informazioni dalla sequenza di pacchetti** scambiati tra due host, poiché viene analizzata solamente (e parzialmente) la fase del **three way handshake**;
- È possibile sapere **solo** con quale host remoto, ha stabilito una connessione un certo processo.

Nell'ambito della sicurezza informatica, questo **tool** può essere impiegato per individuare processi malevoli: un **elevato numero di connessioni accettate o richieste** in una certa unità di tempo, è **sintomo** di probabile **attività malevola** (ovviamente non è detto che lo sia sicuramente, si pensi per esempio ad applicazioni di file sharing basate sul paradigma p2p).

FLUSSI

Un generico flusso è univocamente determinato dalla seguente chiave: **PID**, **nome processo**, **versione IP**, **indirizzo IP sorgente**, **indirizzo IP destinatario**, **numero di porta** (locale se flusso relativo a connect(), remota se flusso relativo ad accept()). Il valore, associato ad una certa chiave, è invece composto da due attributi: contatore di chiamate appartenenti a quel flusso (**counter**), e **MTBS** (**mean time between sys-call**);

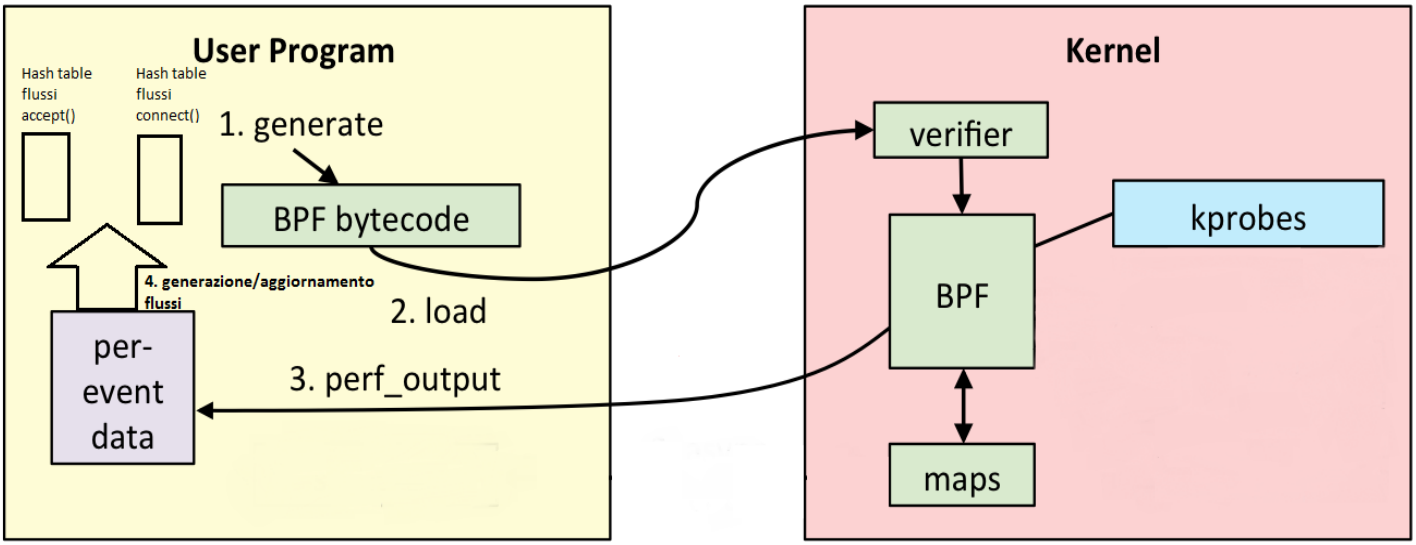
Il **MTBS** misura quanto tempo in media intercorre tra due "comunicazioni" consecutive appartenenti ad un certo flusso; tale attributo è stato introdotto per poter analizzare in modo più critico l'intero flusso: il solo valore del contatore non dice molto sulla natura del flusso, specialmente per analisi di sicurezza; per esempio, un numero elevato di connessioni vicinissime (temporalmente) tra loro sono sintomo di una possibile attività malevola; con il solo contatore non sarebbe stato possibile effettuare questo tipo di analisi.

In estrema sintesi, il *MTBS* indica quanto sono **insistenti** le connessioni tra un processo ed un certo host remoto.

La **granularità** dei flussi è stata mantenuta volutamente **fine** per consentire ad un eventuale **collector** di eseguire quante più analisi possibili (magari a granularità maggiori); per esempio, un flusso generato dal tool non mostra immediatamente in **quante** connessioni è stato coinvolto un certo processo, ma con una opportuna trasformazione è possibile dedurre in modo semplice tale informazione.

ARCHITETTURA

Viene illustrata schematicamente l'architettura del tool realizzato.



Come detto in precedenza, per tracciare le chiamate di sistema accept() e connect() è stato necessario iniettare del codice eBPF all'interno del kernel. eBPF, dal punto di vista del linguaggio, non è altro che codice C scritto a basso livello.

Quindi, una volta scritto del codice eBPF, questo viene prima compilato in **bytecode** grazie al compilatore **bcc**, e successivamente viene iniettato nel kernel (se la validazione va a buon fine). Il codice scritto viene quindi agganciato alle funzioni del kernel che si vogliono tracciare (se si vogliono tracciare eventi del kernel). Il tool in questione collega delle funzioni (chiamate **kprobes**) alle routine del kernel che realizzano l'accept() e la connect() (**tcp_v4_connect/tcp_v6_connect** per la connect(), **inet_csk_accept** per l'accept()).

Le funzioni agganciate agli eventi del kernel possono essere chiamate sia al momento dell'invocazione delle sys-call (se **kprobe**), sia quando queste ritornano (se **kretprob**). La decisione sull'uso di una **kprobe**, di una **kretprobe** o di **entrambe** dipende dalla system call da monitorare. Per esempio, non avrebbe molto senso installare una kprobe per l'**inet_csk_accept()**, poiché le informazioni che ci servono, vengono fornite quando la chiamata di sistema ritorna.

Una volta configurato (con opportune **espressioni di filtraggio**, definibili dall'utente) ed inizializzato il codice BPF, vengono aperti (dallo spazio utente) dei canali di comunicazione (buffer circolari) tra kernel ed user-space; attraverso queste **pipe** le sonde installate possono comunicare con il processo a livello

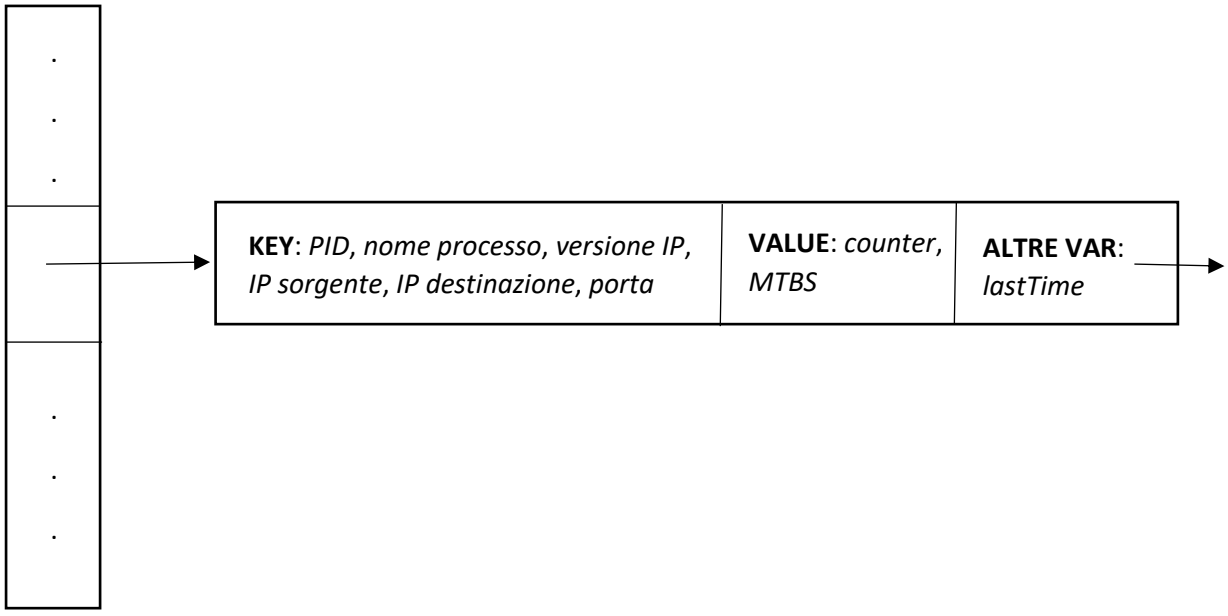
utente, trasmettendo informazioni riguardanti le chiamate tracciate. Nella fattispecie vengono comunicate tutte e sole le informazioni per creare e aggiornare i flussi a livello utente. Quindi, quando arrivano dati dal kernel, viene invocato un'apposita **callback** (in base all'evento che li ha generati) che inserisce in una tabella hash di flussi le informazioni ricevute. Sono state definite due tabelle hash, una per i flussi relativi alla TCP accept() e l'altra per i flussi relativi alla TCP connect().

Quando l'applicazione riceve un segnale di tipo **SIGINT**, i flussi vengono stampati a schermo oppure scritti in un file, (in base alle direttive dell'utente) ed il processo termina.

STRUTTURE DATI

Le due tabelle hash presenti nel sistema sono implementate da due oggetti della classe **SimpleFlowTable** (che estende la classe base **AbstractFlowTable**). Le collisioni vengono gestite con liste di trabocco. Ogni elemento delle liste di trabocco è un oggetto della classe **FlowEntry** (rappresenta un flusso).

Viene data una rappresentazione schematica della struttura dati:



La variabile di lavoro *lastTime* interviene nel calcolo del MTBS; essa infatti mantiene il timestamp dell'ultima sys-call appartenente al flusso.

GUIDA ALL'USO

Per i prerequisiti leggere il file **README.txt**, presente nella directory del tool.

Per lanciare lo script occorre recarsi nella directory del tool e digitare da terminale il seguente comando: **sudo ./eBPF_TCPFlow.py** (controllare permessi di esecuzione).

È possibile eseguire lo script con le seguenti **opzioni**:

- *-p* <PID>: consente di tracciare solo il processo con PID <PID>

- *-P* <port1, port_2, …, port_n>: consente di tracciare solo le connessioni che riguardano le porte **port_i** i=1…n.
- *-R* <port_1, port2>: consente di tracciare solo le connessioni che riguardano le porte nell'**intervallo** [port_1, port2], port_1<=port_2
- *-f* <filepath>: consente di scrivere l'output su file <filepath>

In Sintesi, per lanciare il programma digitare:

sudo ./eBPF_TCPFlow [-p <pid>] [-P <port_1,…,port_n>] [-R <port_1,port_2>] [-f <filepath>]

Per terminare l'applicazione è necessario inviare al processo un segnale **SIGINT** (CTRL_C da terminale); prima della chiusura l'output viene stampato a schermo oppure viene scritto su file se esplicitamente indicato dall'utente (tramite opzione -f).

L'**output** ha la seguente struttura:

```
working... Press CTRL-C to print results
^C
PID    COMM             IP SADDR                                    DADDR                                    RPORT  COUNT MTBS(s)
4429   transmission-gt  4  131.114.119.103                          170.82.140.79                            13497  1     -
4429   transmission-gt  4  131.114.119.103                          109.115.25.251                           51413  1     -
4429   transmission-gt  4  131.114.119.103                          2.110.31.100                             51413  2     1.535329
4429   transmission-gt  4  131.114.119.103                          84.117.120.175                           51413  1     -
4438   transmission-gt  4  131.114.119.103                          91.189.95.21                             6969   2     3.064291
4429   transmission-gt  4  131.114.119.103                          24.96.70.68                              51413  1     -
4429   transmission-gt  4  131.114.119.103                          2.62.161.154                             54242  1     -
4429   transmission-gt  4  131.114.119.103                          41.82.10.44                              51413  1     -

PID    COMM             IP SADDR                                    DADDR                                    LPORT  COUNT MTBS(s)
4429   transmission-gt  4  95.91.212.71                             131.114.119.103                          51413  1     -
4429   transmission-gt  4  91.189.95.21                             131.114.119.103                          51413  1     -
4429   transmission-gt  4  51.174.196.170                           131.114.119.103                          51413  1     -
4429   transmission-gt  4  81.136.82.180                            131.114.119.103                          51413  1     -
```

Ogni riga rappresenta un flusso. Nell'esempio, gli **ultimi** 4 flussi sono relativi alla TCP accept(), i restanti sono relativi alla TCP connect().

CONCLUSIONI

Il codice relativo alla parte **eBPF** è stato scritto facendo riferimento ad alcuni tool già esistenti, presenti in rete:

- **tcpconnect:** *https://github.com/iovisor/bcc/blob/master/tools/tcpconnect.py*
- **tcpaccept:** *https://github.com/iovisor/bcc/blob/master/tools/tcpaccept.py*

Il codice è stato compreso ed è stato modificato opportunamente per consentire un'agevole gestione dei flussi.

Il tool funziona correttamente su **Ubuntu 18.04 LTS**.