Port scans detector with prolog

Progetto per il corso di intelligenza artificiale anno 2010/2011

Andrea Imparato - Lorenzo Tessari

Indice

1 Introduzione

In questo progetto è stato realizzato un *port scanners detector* che utilizza **Prolog** per il riconoscimento di un port scanning.

Il port scanning è una tecnica che viene utilizzata per l'identificazione delle porte aperte in un host remoto. Come si sa, infatti, quando si apre un servizio questo si mette in ascolto delle connessioni in entrata su una determinata porta, assegnata dal sistema operativo.

Alcuni di questi servizi molto spesso possono essere vulnerabili e quindi sfruttabili dall'esterno per ottenere accesso non autorizzato. Il *port scanning* viene utilizzato dunque per il riconoscimento di quali servizi sono attivi ed è il primo passo prima di un eventuale attacco.

Esistono vari software sul mercato che eseguono l'analisi degli host di una rete per il riconoscimento di intrusioni non autorizzate e vengono chiamati *Intrusion detection systems*. Questi software possono essere molto evoluti e sfruttare tecniche di *intelligenza artificiale* o *apprendimento automatico* per le loro attività che possono essere analisi del traffico di rete, dei log oppure del carico cpu o I/O del sistema. All'interno di questi software è presente un modulo per il riconoscimento di scansioni di porte.

In questo progetto si è voluto realizzare uno di questi possibili moduli utilizzando una base di conoscenza realizzata in *Prolog*.

2 Consuntivo

Per la realizzazione del progetto sono state impiegate all'incirca 100 ore complessive distribuite in questo modo:

- 50 ore per lo studio della progettazione e lo studio delle tecnologie da utilizzare nel progetto
- 40 ore per lo sviluppo
- 10 ore per la stesura di questa relazione

3 Use case

Sono stati progettati 2 casi d'uso principali:

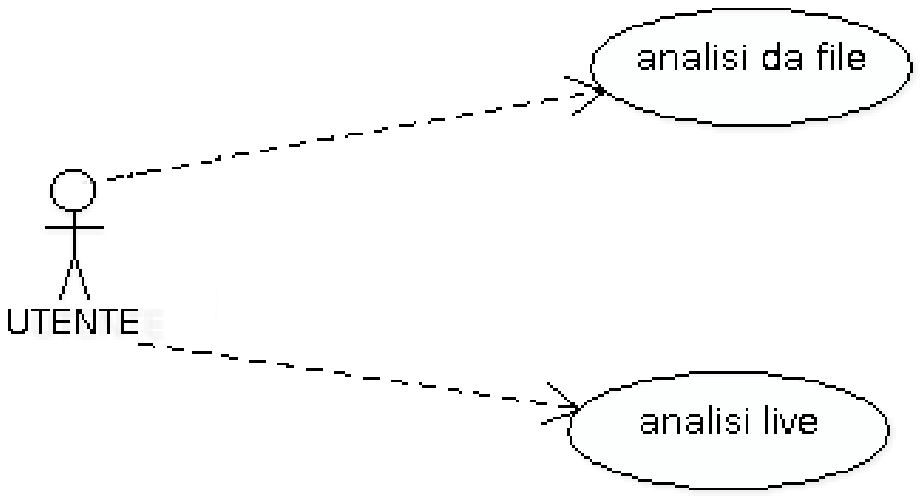

Figura 1: Casi d'uso

- analisi **offline** da file: l'utente può far analizzare un file di traffico precedentemente catturato. In gergo il traffico viene sniffato, vengono catturati i pacchetti e salvati su file. I programmi maggiormente utilizzati per la cattura sono **tcpdump** e **Wireshark**. Il primo esegue da riga di comando e il secondo è una sua interfaccia grafica. Dopo aver catturato il traffico, l'utente esegue l'analisi del file e il sistema riporta la presenza o meno di un port scanning da parte di un host remoto e il suo indirizzi ip.
- analisi **online** del traffico di rete: l'utente esegue il software che effettua lo sniffing live del traffico e in modo continuativo la sua analisi. Se viene rilevato uno scanning il sistema esce e riporta la presenza dello scannning e dell'host remoto che l'ha eseguito e il suo indirizzo ip.

4 Strumenti

Nel progetto sono stati utilizzati principalmente 2 tool di supporto:

- **nmap**: il port scanner open source più utilizzato. Ha moltissime opzioni ed implementa praticamente ogni tecnica di port scanning conosciuta.
- **wireshark**: uno *sniffer* per l'analisi del traffico. Questo tool l'abbiamo utilizzato per effettuare il *reverse engineering* di come nmap effettua il port scanning e per salvare il traffico per l'analisi del detector.

5 Scenari

In questa sezione verranno illustrati i vari possibili scenari in cui l'applicazione vedrà il suo utilizzo. In una prima fase l'applicazione verrà installata su uno degli host della propria rete, verrà messa dunque in modalità di sniffing oppure gli verrà caricato un file di traffico salvato precendentemente. Dopo di ciò i casi in cui ci si potrà trovare sono fondamentalmente 2:

- l'host sul quale è installata l'applicazione non ha servizi attivi, dunque tutte le sue porte sono chiuse.
- l'host ha attivi alcuni servizi.

L'applicazione dunque monitorerà l'host ed informerà l'eventuale amministratore di una scansione di porte.
Ora dobbiamo vedere quali sono i possibili casi per questa scansione:

- un host remoto effettua una scansione generale su tutte le porte dell'host.
- un host remoto scansiona solo un piccolo range di porte, queste possono essere proprio le stesse porte attive sull'host, diciamo n *porte* oppure le porte dell'host ed alcune altre, dunque $n + k$.

In tutti e due i casi il nostro detector deve avvisare della scansione in corso.

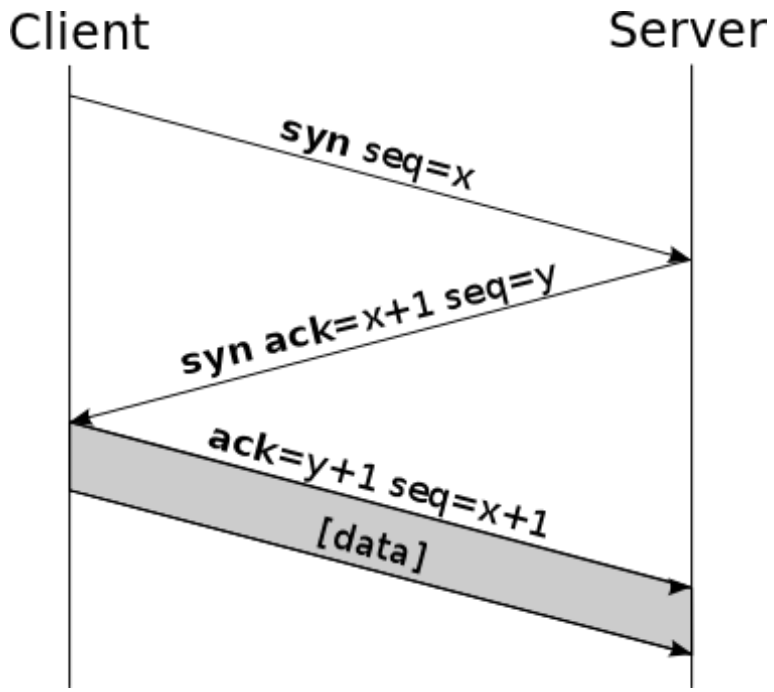

Figura 2: Protocollo Tcp

6 Protocollo Tcp e port scanning

Nella figura è rappresentata una *connessione tcp client-server*. Come si sa, una connessione tcp è formata dal cosiddetto *Three-way handshake* che può essere riassunto così:

- il client invia al server un pacchetto con **flag SYN** impostato ad 1 (attivo) e il campo *sequence number* inizializzato ad un numero casuale *x*.
- il server risponde con un pacchetto con **flag SYN** e **flag ACK** impostati ad 1, il campo **sequence number** impostato ad un valore casuale y e il campo *Acknowledgment number* uguale a *x + 1*
- il client risponde al messaggio del server con un pacchetto con **flag ACK** attivo, il campo *Acknowledgment number* impostato a y + 1 e il campo **sequence number** uguale a *x + 1*. Ora il client può iniziare l'invio dei dati al server.
- se la porta del server sulla quale vuole comunicare il client è chiusa, dopo il primo passaggio il server risponde con un pacchetto con **flag**

RST (reset) attivo e con campo *sequence number* e Acknowledgment number come nel secondo passaggio prima. Con il flag rst la connessione tcp viene interrotta e in questo modo il client riconosce che la porta sul server è chiusa e che quindi non può comunicare.

Uno port scanning non è altro che il tentativo di aprire porte su di un server e riportare quali di queste sono aperte e quali sono chiuse. Il nostro ports scan detector riesce a riconoscere 2 tipi di port scanning:

- **TCP connect scan**: il client completa tutto il *Three-way handshake* su ognuna delle porte sottoposte a scanning. E' la tecnica più rumorosa perchè genera molto traffico ma è anche quella più affidabile.
- **SYN scan**: il client non completa tutte le fasi del protocollo TCP ma si ferma dopo che il server ha risposto con il primo pacchetto con **flag SYN** attivo. Se la porta non fosse aperta infatti il server avrebbe risposto con un pacchetto con flag **RST** attivo. Dunque è corretto ritenere la porta sia aperta.

7 Progettazione

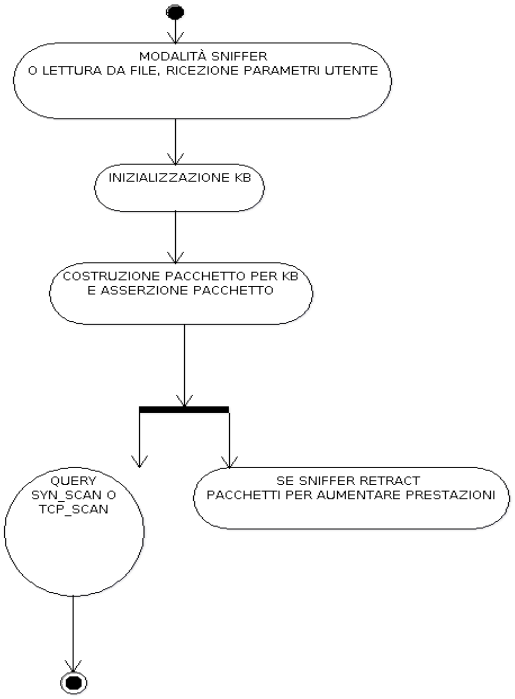

Figura 3: Diagramma delle attivita

La figura rappresenta il *diagramma delle attività* dell'intero software. Come si può vedere all'inizio vengono richiesti i parametri di inizializzazione da parte dell'utente: questi sono del tipo:

kb.pl n_connections file.pcap sniffer seconds_retract_timer: il primo parametro è la base di conoscenza scritta in prolog (nelle successive sezioni la descriveremo approfonditamente), il secondo parametro è il numero di connessioni tcp che devono essere attive affinchè venga riconosciuto un port scanning, il terzo parametro invece può rappresentare un file di traffico sniffato oppure essere la stringa sniffer: nel primo caso il software effettua l'analisi del file ed inferisce lo scan o meno, nel secondo caso si avvia in modalità demone ed effettua l'analisi *online*. Infine il quarto parametro rappresenta il numero di secondi da aspettare affinchè venga avviato il Thread che effettua il retract dei fatti della base di conoscenza.

Questo è il diagramma delle classi del progetto:

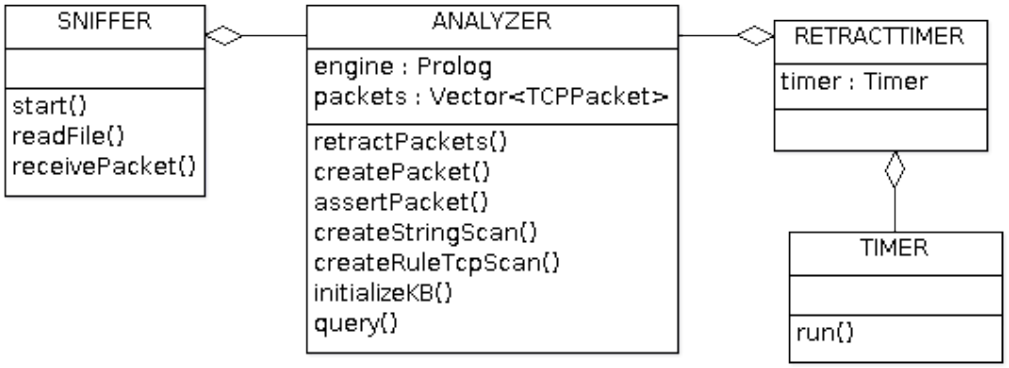

Figura 4: Diagramma delle classi

Descrizione dettagliata delle classi e dei loro metodi:

SNIFFER: classe che effettua lo sniffing del traffico

- start(): avvia lo sniffer se in modalità live
- readFile(): legge da file un dump di traffico
- receivePacket(): ogni volta che lo sniffer riceve un pacchetto viene invocato questo metodo. Asserisce il pacchetto nella base di conoscenza ed effettua la query per il riconoscimento del port scanning.

ANALYZER: classe che si occupa dell'analisi dei pacchetti

- createPacket(): prende un pacchetto in formato raw (jpcap) e costruisce un termine prolog per la base di conoscenza
- assertPacket(): asserisce il pacchetto nella base di conoscenza
- createRuleTcpScan(): crea le regole a partire dal paramentro **n_connections** che fornisce l'utente. Ad esempio con il parametro uguale a 4 il programma genera la seguente regola:

```
tcp_scan(X,Y):- connessione_tcp(X,Y,A1,A2)
,A1\=A2, connessione_tcp(X,Y,A3,A4)
,A3\=A4, connessione_tcp(X,Y,A5,A6),
A5\=A6, A2\=A4, A2\=A6, A4\=A6.
```

  Il suo significato verrà illustrato meglio nella sezione 9.

- createStringScan(): metodo ausiliario per createRuleTcpScan
- initializeKB(): inizializza la base di conoscenza con il file prolog ed asserisce la regola parametrica dell'utente
- query(): esegue la query per il port scanning
- retractPackets(): questo metodo esegue il retract ogni tot secondi (scelti dall'utente) di tutta la base di conoscenza. E' stato necessario implementare tale metodo per questioni di performance: una base di conoscenza che cresce in modo infinito porta a prestazioni notevolmente scarse!

RETRACTIMER: si occupa di avviare il thread che esegue il retract della base di conoscenza

8 Tecnologie

Il progetto è stato realizzato in *Java*. Le librerie che sono state utilizzate sono:

- jpcap: porting in java della libreria pcap per l'analisi del traffico di rete.
- tuprolog: libreria java per utilizzare prolog all'interno di codice java. Tra le molte disponibili è stata utilizzata questa per le migliori prestazioni rispetto alle altre.

9 Inferenza

In questa sezione viene illustrato il cuore dell'applicazione e cioè le regole che definiscono il motore di inferenza prolog.

```
/* pacchetti:

   CLIENT: syn, seq = x
   SERVER: syn, seq = y,ack= x + 1,
   CLIENT: seq = x + 1,ack = y + 1
   se porta chiusa
   CLIENT: rst,seq = x + 1,ack = y + 1

*/

/* regola per connessione tcp */
connessione_tcp(SOURCE,DESTINATION,SD,DP):-
pacchetto(SD,DP,syn,SOURCE,DESTINATION,X,0)% syn, seq = x
,pacchetto(DP,SD,syn,DESTINATION,SOURCE,Y,Z)%seq = y,ack= x + 1,
,pacchetto(SD,DP,SOURCE,DESTINATION,Z,W)%seq = x + 1,ack = y + 1
,Z is X+1,W is Y+1.




/* regola per connessione connessione syn */
connessione_syn(SOURCE,DESTINATION,SD,DP):-
pacchetto(SD,DP,syn,SOURCE,DESTINATION,X,0)% syn, seq = x
,pacchetto(DP,SD,syn,DESTINATION,SOURCE,Y,Z)%seq = y,ack= x + 1,
,Z is X+1.


/* regola per riconoscere se la porta e' chiusa */
porta_chiusa(SOURCE,DESTINATION,SD,DP):-
pacchetto(SD,DP,syn,SOURCE,DESTINATION,X,0)% syn, seq = x
,pacchetto(DP,SP,rst,DESTINATION,SOURCE,0,Z)%seq = x + 1,ack = y + 1
,Z is X+1.
```

Il predicato base all'interno della KB è **pacchetto**. Esso rappresenta un singolo pacchetto nel network ed è formato da 6 termini:

- SP: source port, rappresenta la porta sorgente del pacchetto

- DP: destination port, rappresenta la porta destinazione del pacchetto
- syn o rst: rappresentano i flag SYN o RST impostati a 1 del pacchetto. Possono anche non essere presenti.
- DESTINATION: ip di destinazione
- SOURCE: ip sorgente
- le ultime 2 variabili rappresentano rispettivamente l' acknowledgment number e il sequence number

Nella base di conoscenza sono presenti anche altre 3 regole: **connessione_tcp**, **connessione_syn** e **porta_chiusa**: la prima rappresenta l'inferenza di una connessione tcp tra 2 host su una porta sorgente ed una di destinazione, la seconda una connessione syn, cioè che si ferma dopo lo scambio dei primi 2 messaggi tra gli host e la terza la richiesta di un host di una connessione su una porta e la relativa risposta dell'host di destinazione di porta chiusa.

Come avevamo precedentemente illustrato la regola principale della base di conoscenza è quella per il riconoscimento di uno scanning. Le regole per inferirlo sono 2: una per il riconoscimento di un port scanning tcp e uno per il port scanning syn. Le regole vengono generate come detto precedentemente utilizzando l'input dell' utente **n_connections_open_port** e **n_connections_closed_port** che rappresentano il numero di connessioni tcp che devono essere attive per inferire un port scanning e il numero di porte chiuse che devono essere interrogate. Le regole ad esempio con richiesta di 4 connessioni aperte e 3 porte chiuse sono generate così:

```
tcp_scan(X,Y):- connessione_tcp(X,Y,A1,A2),A1\=A2,
connessione_tcp(X,Y,A3,A4),
A3\=A4,connessione_tcp(X,Y,A5,A6)
,A5\=A6,connessione_tcp(X,Y,A7,A8),A7\=A8,
A2\=A4,A2\=A6,A2\=A8,A4\=A6.A4\=A8,A6\=A8.

syn_scan(X,Y):- connessione_syn(X,Y,A1,A2),A1\=A2,
connessione_syn(X,Y,A3,A4),A3\=A4
,connessione_syn(X,Y,A5,A6),A5\=A6,
connessione_syn(X,Y,A7,A8),A7\=A8,A2\=A4,A2\=A6,
A2\=A8,A4\=A6.A4\=A8,A6\=A8.
```

```
tcp_scan(X,Y):- porta_chiusa(X,Y,A1,A2),
A1\=A2, porta_chiusa(X,Y,A3,A4),
A3\=A4, porta_chiusa(X,Y,A5,A6),
A5\=A6, A2\=A4, A2\=A6, A4\=A6.
```

Come si può vedere per attivare le regole sono necessarie rispettivamente n connessioni tcp o connessioni syn e n connessioni di porte chiuse. Sono da notare i vincoli di differenza tra le porte di destinazioni delle varie connessioni.

10 Compilazione ed esecuzione del progetto

Per compilare il progetto si utilizza il tool java **ant**. Un esempio di parametri necessari alla compilazione su piattaforma linux:

sudo ant compile jar

Questo comando creerà un file *project.jar* all'interno della directory del progetto.

Per avviare il progetto invece dare:

java -jar project.jar opzioni

11 Risultati

Il progetto è stato testato in diverse condizioni di utilizzo:

- scan delle porte aperte e scan di porte aperte/chiuse
- nessuna generazione di traffico dell'host sottoposto a scan
- host sottoposto a scan genera molto traffico
- l'host viene sottoposto a tcp scan
- l'host viene sottoposto a syn scan

In quasi tutti i casi il software si è comportato correttamente riportando lo scan (se i parametri dell'utente erano sufficienti per riconoscerlo ovviamente). L'unico problema è nello syn scan. In questo particolare scan infatti per essere più nascosto possibile infatti il tool di scanning spesso genera pochissimo traffico e riesce ad eludere il nostro port scanners detector.

12 Sviluppi futuri

Nel futuro il progetto potrebbe essere reso un Intrusion Detection System vero e proprio con l'aggiunta di regole all'interno della sua base di conoscenza e la loro relativa integrazione all'interno del codice Java. Infine potrebbero essere usati anche approcci di apprendimento automatico per far sì che il progetto possa migliorare con l'esperienza e dunque limitare al massimo i falsi positivi/negativi. Ad esempio come dati di apprendimento si possono utilizzare i dati del Dipartimento della difesa americano presente a questo indirizzo http://goo.gl/km3RX.