Relazione progetto di
Gestione di reti

“Slowloris”

di Bucchianeri Elena, 463889

INDICE

1. Introduzione:

Slowloris è un software che permette ad una singola macchina, con capacità molto limitate, di occupare risorse di un server. L'attacco al server, di tipo DOS, consiste nell'aprire numerose connessioni verso un server e cercare di mantenerle aperte il più possibile inviando richieste parziali. Il server, sovraccaricato da queste connessioni, non potrà soddisfare le ulteriori richieste degli altri utenti.

Il software sviluppato consente di rilevare un attacco di tipo Slowloris analizzando offline i flussi registrati. In sostanza il programma controlla il numero di connessioni aperte per ogni minuto; se rileva che vi sono più di 300 connessioni in almeno uno dei minuti analizzati, stampa a video l'indirizzo ip delle macchina che hanno richiesto più connessioni e la porta del server attaccata. Il programma viene avviato passando come argomenti i file contenenti i flussi. I file passati al programma si suppongono ben formattati ed ordinati nel tempo, inoltre i flussi in essi contenuti devono essere V5. Il software può essere diviso in due fasi principali: la prima fase, che individua tutte le connessioni aperte col server e la seconda fase, che si occupa di analizzare le connessioni per individuare un possibile attacco.

2. PRIMA FASE: creazione delle connessioni.

Il programma, per prima cosa, si occupa di ricostruire le connessioni aperte col server memorizzandole in un ArrayList chiamato "*connessioni*". Ogni connessione è descritta da due flussi monodirezionali: uno, chiamato A o di lato A, i cui pacchetti, in generale, ci si aspetta vadano dal client al server, e l'altro, chiamato B o di lato B, i cui pacchetti transitano in direzione opposta. Similmente l'oggetto Connessione, che modella una connessione, ha due variabili chiamate A e B di tipo Flusso; la variabile A, cioè il flusso di lato A, viene inizializzata in fase di creazione dell'oggetto, in quanto il flusso iniziale è solitamente quello con la richiesta di apertura di una connessione. La variabile di lato B, invece, viene inizializzata tramite il metodo *add()* della classe Connessione, invocato appena si incontra il primo flusso in direzione opposta.

Quindi la prima fase consiste nel costruire, per ogni riga del file, ad eccezione della prima contenente il template, un oggetto Flusso inizializzandone i campi più importanti con le informazioni prelevate dal file.
Alla prima riga valida del primo file si crea direttamente anche un oggetto Connessione, a partire dall'oggetto Flusso appena creato, e lo si inserisce direttamente nell'ArrayList. Prima dell'inserimento si analizzano i flag TCP e si inizializzano i vari campi del nuovo oggetto Connessione: le variabili "*syninviato*" e "*fininviato*" vengono poste ad 1 in caso si individui rispettivamente un SYN e un FIN tra i flag; la variabile "*tempoinizio*" viene inizializzata col valore del campo FIRST_SWITCHED, "*tempofine*" con il valore del campo LAST_SWITCHED incrementato di 1, e le variabili "*totpacchetti*" e "*totbytes*" con il numero di pacchetti ed il numero di bytes del flusso.
Per le successive righe del primo file e per tutte le linee dei seguenti file, prima di creare una nuova Connessione a partire dal Flusso appena individuato, si deve verificare che non esista già una Connessione nell'Array, creata precedentemente, a cui il flusso appartiene.

- In caso negativo si crea un nuovo oggetto Connessione: si analizza i flag TCP, le variabili "*syninviato*" e "*fininviato*" vengono poste ad 1 in caso rispettivamente si individui un SYN e un FIN tra i flag; la variabile "*tempoinizio*" viene inizializzata col valore del campo FIRST_SWITCHED, "*tempofine*" con il valore del campo LAST_SWITCHED incrementato di 1, e le variabili "*totpacchetti*" "*totbytes*" vengono inizializzate con il numero di pacchetti ed il numero di bytes del flusso.
- In caso positivo si deve cercare nell'arrayList la connessione a cui il flusso appartiene tramite i 4 campi che individuano univocamente una connessione: IPSORG, IPDEST,

PORTASORG, PORTADEST. Il flusso appena creato sarà o un flusso di lato A o un flusso di lato B.
- Se è un flusso di lato A è necessario semplicemente analizzare i flag TCP e aggiornare alcune variabili: se vi è un FIN la variabile "*fininviato*" diventa true, se c'è un RST la variabile "*chiusa*" diventa true, "*tempofine*" prende il valore di LAST_SWITCHED del nuovo flusso incrementandolo di 1, a "*totpacchetti*" e "*totbytes*" vengono aggiunti i pacchetti e bytes del nuovo flusso.
- Se è un flusso di lato B significa che la ricerca della connessione nell'ArrayList ha avuto successo invertendo i 4 campi tra sorgente e destinatario. In questo caso può essere la prima volta che si incontra un flusso in tale direzione.
    - Se è la prima volta che si individua un flusso in direzione B si deve inizializzare la variabile B della Connessione a cui il flusso appartiene, invocando il metodo *add()* con parametro l'oggetto Flusso appena costruito. In seguito si aggiornano vari campi e si analizza i flag TCP: se si individua un SIN + ACK la variabile "*aperta*" diventa true, se si individua un FIN + ACK o un RST la variabile "*chiusa*" diventa true, "*tempofine*" assume il valore del campo LAST_SWITCHED, "*totpacchetti*" e "*totbytes*" vengono incrementati con i pacchetti ed i bytes di questo flusso.
    - Se non è la prima volta che incontriamo un flusso in direzione opposta, è necessario analizzare i flag TCP, e similmente a caso sopra, si inizializzano o modificano le variabili della Connessione a cui il flusso appartiene.

3. SECONDA FASE:

A questo punto il programma ha tutte le connessioni all'interno dell'arrayList "*connessioni"* e considera ed analizza solo quelle aperte, cioè di cui ha individuato un flusso contenente un SYN e successivamente un SYN+ACK.
Per procedere nell'analisi il programma deve individuare due dati, uno, l'intervallo di tempo, in minuti, impiegato dal probe per la creazione dei file passati come parametro e l'altro, un valore SysUpTime che coincida con l'inizio dell'intervallo di tempo; quindi si assegna alla variabile "*iniziointervallo"* il valore di "*tempoinizio"* (corrispondente al FIRST_SWITCHED del flusso di apertura) più basso tra tutti quelli presenti nelle connessioni che risultato aperte e alla variabile "*fineintervallo*" si assegna invece il "*tempofine*" (valore corrispondente all'ultimo LAST_SWITCHED relativo alla connessione) massimo tra quelli delle connessioni aperte.
Successivamente si calcola l'intervallo in minuti sottraendo "*iniziointervallo*" a "*fineintervallo*" e dividendo per 60 (si incrementano i minuti trovati di 1, per arrotondare il risultato).
Da qui in poi, conoscendo i minuti, inzia l'analisi vera e propria. Si invoca la funzione "*individuaFlussi()*" che prende come parametri l'ArrayList di connessioni, i minuti di durata dell'analisi dei flussi e il tempo di inizio intervallo e restituisce un array di interi dove ogni intero indica il numero di connessioni avute nel minuto indicato dal valore dell'indice.
Se in almeno un minuto si hanno più di 300 connessioni, allora vi è la possibilità di un attacco e si procede con la seconda parte dell'analisi, altrimenti, se le connessioni sono poche, il programma termina indicando l'assenza di un possibile attacco. Questa seconda parte di analisi consiste nel chiamare la funzione "*individuaIPePorta()*" che ha il compito di costruire due array bidimensionali uno di stringhe, per gli IP, e uno di interi, per le porte.
L'array bidimensionale di stringhe, chiamato "*indip*" ha tante righe quanti sono i minuti dell'intervallo e, per ogni minuto, gli array contengono l'elenco di tutti gli indirizzi ip, responsabili dell'apertura di una connessione che in tal minuto è ancora aperta. Similmente l'array bidimensionale di interi, chiamato, "*porteserver*" ha tante righe quanti i minuti dell'intervallo e, per ogni minuto, gli array contengono l'elenco di tutte le porte utilizzate dal server in una connessione.
A partire da questi due arraybidimensionali, per ogni minuto, si costruiscono due oggetti di tipo **IndirizziPresenti** e **PortePresenti**, che permettono di contare, quali sono gli IP e le porte più frequenti. Se, infatti, IndirizziPresenti e PorteFrequenti hanno un indirizzo o una porta con un

contatore > 300 viene stampato a video due informazioni, l'indirizzo e la porta coinvolti, considerando il primo dato responsabile, nel minuto, della maggior parte delle connessioni ed il secondo come la porta che riceve la maggior parte delle connessioni

4. Classi definite:

La classe **Applicazione** è la classe principale che contiene il main. Ha vari metodi statici:

- *checkzero()*: metodo utilizzato per controllare che i flag non siano tutti a 0.
- *individuaflussi()*: metodo che calcola quanti sono i flussi aperti per minuto. Restituisce un array di interi, dove gli indici rappresentano il minuto ed il contenuto il numero di connessioni aperte.
- *individuaIPePorta()*: metodo che permette di raggruppare per minuto gli IP sorgente e le porte del server utilizzate. Inizializza due array bidimensionali, uno di stringhe per gli IP, uno di interi per le porte.
- *calcolotcpflags()*: metodo per velocizzare l'analisi dei flagTCP: a partite dal campo TCP_FLAGS di in un flusso, la funzione crea e restituisce un array di 6 posizioni dove ogni indice corrisponde ad un flag, quindi in 0 abbiamo URG, 1 ACK, 2 PSH, 3 RST, 4 SYN, 5 FIN.

La classe **Flusso** memorizza come variabili d'istanza le informazioni più importanti tra quelle che troviamo in ogni riga dei file: IPV4_SRC_ADDR, IPV4_DST_ADDR, L4_SCR_PORT, L4_DST_PORT, cioè i 4 campi che permettono di identificare se due righe del file appartengono ad una solita connessione, e IN_PKTS, IN_BYTES, FIRST_SWITCHED, LAST_SWITCHED.

La classe **Connessione** memorizza 2 flussi (oggetti Flusso) A e B, che rappresentano i flussi delle 2 direzioni di comunicazione (A, variabile di tipo Flusso, indica il flusso di apertura di una connessione, cioè contiene il SYN, ed è, infatti, inizializzato in fase di creazione della Connessione). Poichè una connessione può essere aperta e/o chiusa la classe ha 4 variabili d'istanza: "*syninviato*" e "*fininviato*", per memorizzare l'avvenuta richiesta di apertura o chiusura di una connessione (quindi rispettivamente la presenza di SYN e FIN a 1 nei flag TCP analizzati), "*aperta*" e "*chiusa*". La variabile "*aperta*" è posta a true al momento di un SYN+ACK in risposta ad un SYN, cioè "*syninviato*" deve essere già true. La variabile "*chiusa*" diventa true o all'arrivo di un singolo RESET, o per un FIN+ACK in risposta ad un FIN e quindi "*fininviato*" deve essere già true. Questa classe inoltre memorizza il numero totale di pacchetti e bytes scambiati in due variabili "*totpacchetti*" e "*totbytes*" e il tempo di inizio e fine della connessione rispettivamente in "*tempoinizio*" e "*tempofine*".
La classe ha due metodi principali:

- *add()*: metodo che permette di aggiungere all'oggetto connessione il flusso di direzione opposta appena esso viene individuato la prima volta.
- *isIn()*: metodo che permette di verificare se un flusso appartiene o meno alla connessione. Restituisce tre valori: 0 in caso il flusso non appartenga alla connessione, 1 in caso appartenza alla connessione e sia da considerarsi sul lato A, verso il server, e 2 in caso i flusso appartenga alla connessione ma in direzione opposta, ovvero sul lato B, dal server al client.

La classe **IndirizzoIP** contiene come variabile d'istanza una stringa, chiamata "*ip*", che indica l'indirizzo IP associato, e un intero, rappresentante un contatore, chiamato "*counter*", inizializzato ad 1 in fase di costruzione, che può essere incrementato chiamando il metodo *incrementaContatore()*.

La classe **IndirizziPresenti** ha una variabile di istanza "*indirizzi*" di tipo ArrayList di **IndirizzoIP**. I metodi principali della classe sono:

- *aggiungi()*: metodo che prende come parametro una stringa, contente un ip, e permette di

aggiungere un nuovo indirizzo IP. Prima dell'aggiunta si verifica che non esista già un oggetto IndirizzoIP contenente tale ip come stringa, se esiste viene incrementato il contatore di questo oggetto, altrimenti si aggiunge un nuovo oggetto IndirizzoIP con ip specificato da parametro.
- *indMaxCounter()*: metodo che restituisce l'oggetto IndirizzoIP che ha come valore del contatore l'intero più alto tra tutti gli IndirizziIp dell'array.

La classe **Porta** ha come variabili d'istanza due interi, uno per la porta, chiamato "*p*", e uno per il contatore, chiamato "*counter*", inizializzato ad 1 in fase di costruzione e che può essere incrementato chiamando il metodo *incrementaContatore()*.

La classe **PortePresenti** ha una variabile d'istanza di tipo ArrayList di Porta.
I metodi più importanti sono:
- *aggiungi()*: metodo che prende come parametro un intero, indicante una porta, e permette di aggiungere un nuova porta all'array. Prima dell'aggiunta si verifica che non esista già un oggetto Porta contenente tale porta, se esiste viene incrementato il contatore di questo oggetto, altrimenti si aggiunge un nuovo oggetto Porta.
- *indMaxCounter()*: metodo che restituisce la Porta che ha come valore del contatore l'intero più alto.

Istruzioni per compilare ed eseguire:

Il Main è contenuto nella classe Applicazione e, dopo aver compilato in un jar, il programma può essere invocato con vari file di test:

11.flows: flussi slowloris (intervallo circa 7 min)
506part1.flows e *506part2.flows*: flussi slowloris (intervallo circa 10 min)
806part1.flows e *806part2.flows*: flussi slowloris (intervallo di circa 20 min)

normale.flows: flussi normali (intervallo circa 7 min)
normalep2p1.flows, normalep2p2.flows, normalep2p3.flows: flussi normali, traffico p2p (intervallo circa 20 min.)

I file di test attesi dal programma contengono flussi monodirezionali, anche spezzati, in quanto il programma ricostruisce le connessioni, e come prima riga devono contenere il template.

Il programma viene lanciato, ad esempio, con:
java -jar slowloris.jar 506part1.flows 506part2.flows