Indice:

1. Introduzione
   1. Definizione del problema
   2. Struttura della tesi
2. Traffico di rete
   1. Pacchetti di dati
   2. Zeek
3. Process discovery
   1. Trace di eventi (fornire
   2. Conversione di pacchetti in Trace
   3. Reti di Petri
   4. Fitness di un trace
4. Anomaly detection
   1. Definizione del problema
   2. Isolation Forest
5. Validazione empirica
   1. CICID 2017
   2. Risultati dell’estrazione dei trace
   3. Etichettatura dei trace
   4. Metriche di valutazione e setting sperimentale
   5. Risultati
6. Conclusioni

Appendice A - Diagramma delle classi

1. Zeek
2. Creazione dei trace dai pacchetti
3. Anomaly detection

Appendice B -Manuale Utente

1. Download e installazione dei software necessari
2. Download del Dataset
3. Generazione di un file xes
4. Avvio sperimentazione

Bibliografia

1. Introduzione
   1. Definizione del problema

La sperimentazione nasce con l’obiettivo di **rilevare connessioni anomale**, cioè dove si è effettuato un attacco a livello di sicurezza, in un traffico di rete, mediante l’uso di analisi delle connessioni tramite algoritmi di *process mining*, coadiuvato da algoritmi di *Anomaly detection*.

Inizialmente è necessario ottenere i pacchetti e memorizzarli all’interno di *log* di connessioni; quindi, le cui righe del *log* corrispondono ad una connessione (quadrupla indirizzo IP e porta dell’*host* mittente e indirizzo e porta dell’*host* destinatario), e ogni connessione è composta da pacchetti inviati e ricevuti dai due *host*. Le connessioni possono essere catturate tramite un software *sniffer* di pacchetti come *Wireshark*, che una volta terminata la cattura dei pacchetti, questi vengono memorizzati in dei file *.pcap*. Questi file vengono poi analizzati da software specifici che permettono di convertirli file *log*. Il software utilizzato in questa sperimentazione è *Zeek*, un software a linea di comando che dato in input un file *.pcap*, genera dei file *log*, ognuno dei quali contiene informazioni specifiche per i diversi protocolli adoperati da ogni connessione, e un file *conn.log* che contiene tutte le connessioni catturate nel file *.pcap* e le informazioni comuni a tutte le connessioni.

A questo punto è stato creato un applicativo che converte i file *log* di *Zeek*, in file *xes*, che è una specifica di file *xml*, contenente una serie di *trace*, i quali raggruppano una serie di *event*. I file xes sono dei contenitori chiave per il *process mining*, poiché i processi da analizzare corrispondono ai trace e le attività per ogni processo o trace sono gli eventi, che disposti in sequenza descrivono una sequenza di attività che verrà successivamente analizzata per estrarre conoscenza.

I *trace,* in questo caso, sono identificati dalla quadrupla indirizzo IP sorgente, porta sorgente, indirizzo IP destinazione, porta di destinazione, mentre, gli eventi, sono tutte le connessioni, o righe del *log*, che sono accomunate dalla quadrupla del trace.

La sezione di ***process mining***consisterà di due fasi: generazione delle *PetriNet* e calcolo dei livelli di *conformance*.

La **prima fase** ha come obiettivo quello di modellare le *PetriNet* generando dei file *xes* con *trace* ed *event* usando le connessioni normali. Successivamente si calcolano i livelli di *conformance* di ulteriori connessioni normali.

La **seconda fase** ha l’obiettivo di calcolare i livelli di *conformance* di connessioni di tipo normale e anomalo con le *PetriNet* modellate durante la prima fase.

La sezione di ***Anomaly detection***consiste anch’essa di due fasi: la fase di apprendimento del modello di *anomaly detector* e quella di predizione delle anomalie.

Durante la **fase di apprendimento** si utilizzano i livelli di *conformance* calcolati con connessioni normali per allenare il modello di una *Isolation forest.*

Durante la **fase di predizione** si predicono le connessioni anomale con i livelli di conformance calcolati nella seconda fase del *process mining*.

* 1. Struttura della tesi

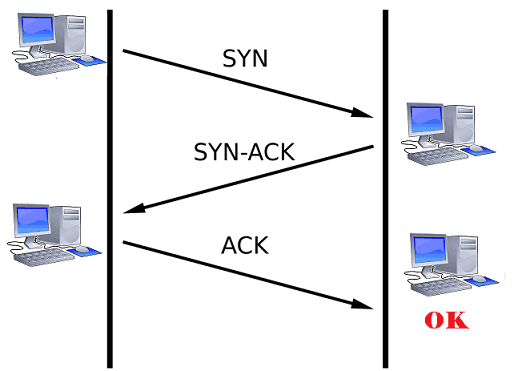
1. Traffico di rete

La sperimentazione si concentra sulla rilevazione di traffico di rete anomalo, nel senso che rileva degli attacchi di rete all’interno di una rete di calcolatori.

I calcolatori per poter comunicare tra di loro, devono essere collegati tra di loro tramite un mezzo fisico, di solito un cavo o l’etere. Dopo essere stati collegati tra loro, direttamente o tramite degli intermediari, possono iniziare a comunicare, e questo è possibile perché sono programmati tramite uno stesso protocollo di rete, chiamato *Internet Protocol*(o IP), che permette di inviare e ricevere messaggi in due protocolli di rete principali: TCP e UDP (rispettivamente “*Transmission Control Protocol*” e *“User Datagram Protocol”*.

Il protocollo UDP non è orientato alla connessione, permette, cioè, di inviare e ricevere pacchetti di dati in maniera molto veloce ed efficiente, ma senza sicurezza che questi arrivino effettivamente a destinazione. Questo protocollo viene adoperato, solitamente, quando non è necessario che tutti i pacchetti arrivino a destinazione e che vengano inviati e ricevuti più velocemente possibile, quindi in applicazioni di *live streaming* o di chiamate di tipo VOIP, o in protocolli come il DNS, dove anche se un pacchetto dovesse essere perso, può sempre essere inviato nuovamente.

Il protocollo **TCP** permette di inviare e ricevere pacchetti con sicurezza che arrivino a destinazione o che, in caso di perdita di pacchetto per un qualsivoglia motivo indipendente dagli interlocutori (interferenza, traffico elevato, …), il pacchetto verrà ritrasmesso. Questo è possibile perché è un protocollo orientato alla connessione, che consiste in tre fasi principali (come rappresentato schematicamente nella seguente figura):

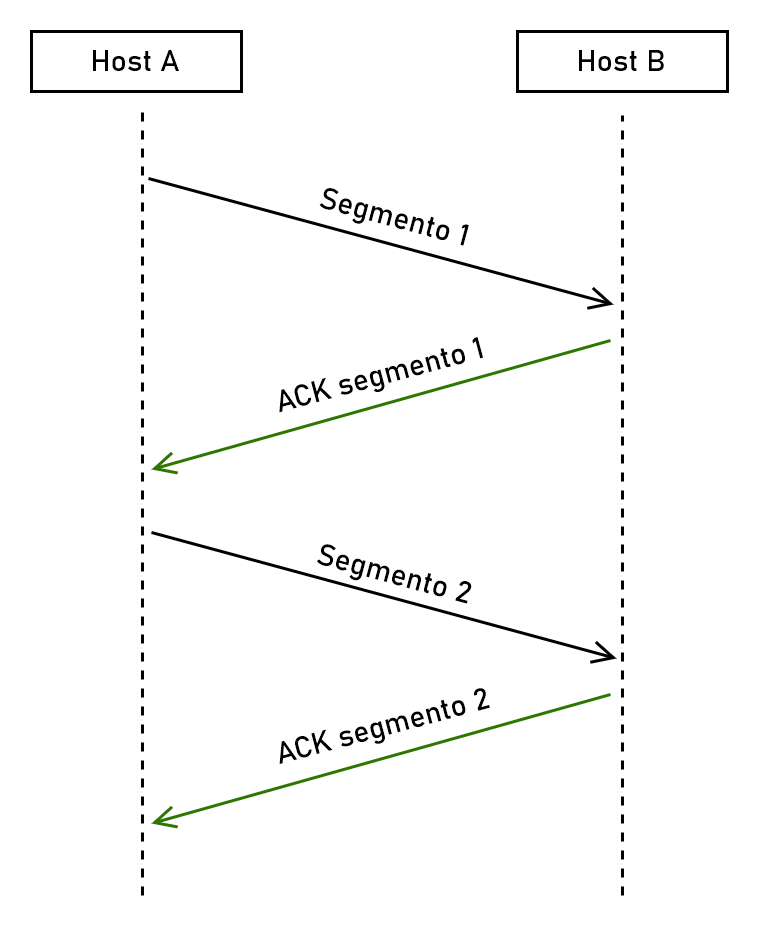


https://www.aiutocomputerhelp.it/wp-content/uploads/Tcp\_normal.png

1. Avvio della connessione: il *client* richiede al *server* di avviare una connessione tra i due, quindi di voler comunicare. Questo avviene tramite la “stretta di mano a tre vie” (*three-way handshake*), che consiste di tre pacchetti scambiati con dei particolari bit impostati a *true* nella testa del pacchetto:
   1. Il *client* invia un pacchetto col bit *SYN* a *true*;
   2. Il *server* risponde inviando un pacchetto con i bin *SYN+ACK* a *true;*
   3. Il *client* invia un pacchetto col bit *ACK* a *true*;

A questo punto, se tutto è andato a buon fine, la connessione viene avviata con successo e due *host* iniziano a scambiarsi messaggi.

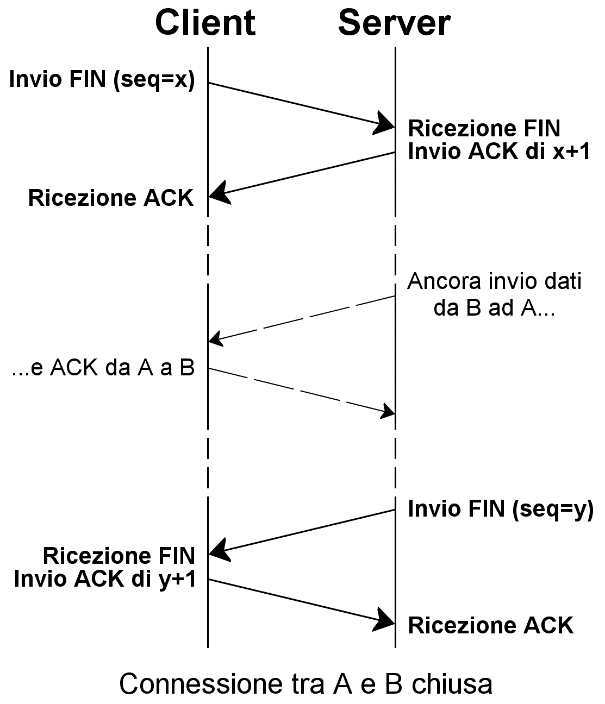
1. Scambio di messaggi: a questo punto ogni messaggio che viene scambiato tra i due *host* deve seguire il seguente schema (rappresentato schematicamente nella seguente figura):



https://fibra.click/images/tcp/sequence.png

* 1. Un *host* invia un messaggio all’altro *host*, il pacchetto ha con sé il numero della sequenza, che corrisponde al numero di bytes inviati dall’inizio della connessione (sommato ad un numero generato casualmente all’apertura della connessione, questo per evitare che dei pacchetti che sono stati persi precedentemente con stessa quadrupla identificativa e numero di sequenza simile, possano contrastare con i pacchetti inviati in questa connessione)
  2. L’*host* ricevente, se il messaggio arriva a destinazione, analizza il pacchetto e invia un pacchetto con il bit *ACK* impostato a *true* e il numero di sequenza pari al numero di sequenza del pacchetto ricevuto sommato alla dimensione in bytes del pacchetto ricevuto. In questo modo è possibile per il mittente sapere se almeno il numero di bytes ricevuti dal destinatario corrispondono a quelli inviati dal mittente. Nel caso in cui il messaggio non arrivi a destinazione, il mittente attenderà un certo periodo di tempo, se non arriva alcuna risposta dal destinatario, questo invierà nuovamente il messaggio.

1. Terminazione della connessione: in questa fase, uno dei due *host* richiede di terminare la connessione, avvengono queste tre fasi (rappresentato schematicamente nella seguente figura):



https://upload.wikimedia.org/wikipedia/commons/b/bb/TCP-Chiusura-a-4-vie.png

1. Uno dei due *host* invia un pacchetto col bit *FIN* impostato a *true*, l’altro *host* riceve il pacchetto e risponde con un pacchetto con pacchetto con bit *ACK* a *true*.
2. Se l’*host* ricevente deve ancora inviare dati, continua ad inviarli come avviene nella fase di scambio di messaggi.
3. Quando l’*host* ha concluso la trasmissione di dati, invia un pacchetto con bit *FIN* a *true*, il ricevente risponde con bit *ACK* a *true* e la connessione viene terminata.
   1. Pacchetti di dati
   2. *Zeek*