Università degli studi di
Milano-Bicocca

Cyber Security
approfondimento

Anomaly detection on user-agent strings

Autore:
Marco Savino - 793516 - m.savino5@campus.unimib.it

Abstract

Gli *hacker* si nascondono in *terabyte* di traffico HTTP legittimo di un'organizzazione, all'interno del quale diventa sempre più difficile rilevare il traffico malevolo. Gli esperti di sicurezza informatica sono tuttavia a conoscenza del fatto che gli *hacker* potrebbero utilizzare *user agent* insoliti ed estranei all'organizzazione, per comunicare con il server di comando, ricevere aggiornamenti, inviare informazioni rubate o scaricare comandi da eseguire.
Questo approfondimento ha l'obiettivo di comprendere il campo dell'*user agent* e mostra alcuni metodi utilizzati per identificare quelli usati con finalità maligne.

1 User-Agent String

Quando un *browser Web* richiede una pagina ad un *server Web*, invia una stringa contenente informazioni relative alla piattaforma, al sistema operativo e sui software installati sul computer richiedente. Il *server Web* può quindi utilizzare queste informazioni per personalizzare meglio il contenuto della pagina per quel particolare *browser*, come per esempio inviare una versione della pagina che è meglio strutturata per gli *smartphone*.
Questa stringa,chiamata *User-Agent string*, ha molte forme ma solitamente è strutturata come il seguente esempio:

Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1866.23 Safari/537.36

Esistono molti prodotti *Web server diversi*, che utilizzano una varietà di architetture e linguaggi di programmazione. E tutti devono, a un certo punto, prendere la stringa ricevuta dalla richiesta o dal *browser* e analizzarla, al fine di poter personalizzare la pagina *Web* come richiesto.
È esattamente l'analisi della *User-Agent string* che viene presa di mira dagli *hacker*, in quanto il momento in cui un sistema remoto sta elaborando un input controllato da un utente è fonte di vulnerabilità.

2 Metodologia di rilevamento delle anomalie

Le tecniche basate su regole sono altamente empiriche e possono solo catturare eventi dannosi identici a quelli accaduti in passato: per le *User-Agent String* è impossibile elencare tutto ciò che può andare storto. Gli eventi dannosi sono molto rari rispetto al volume del traffico in un'organizzazione e raccogliere una quantità sufficiente di dati etichettati non è realistico, considerando anche la variabilità delle *User-Agent String*.
La metodologia di rilevamento delle anomalie delle *User-Agent String* proposta da *Eirini Spyropoulou, Jordan Noble e Chris Anagnostopoulos* consiste nel rappresentare queste stringhe direttamente nello spazio definito dalle loro distanze. In particolare sono state utilizzate le seguenti misure di distanze basate sulle stringe:

- ***Levenshtein***: misura la differenza tra due sequenze. La distanza di *Levenshtein* tra due parole è il numero minimo di modifiche a carattere singolo (inserimenti, eliminazioni o sostituzioni) necessarie per cambiare una parola nell'altra. Questa distanza può anche essere definita "*distanza di modifica*".

$$Lev_{A,B}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{A,B}(i-1,j) + 1 \\ lev_{A,B}(i,j-1) + 1 \\ lev_{A,B}(i-1,j-1) + 1_{A_i \neq B_j} \end{cases} & \text{otherwise.} \end{cases}$$

  dove $lev_{A,B}(i,j)$ è la distanza tra i primi i caratteri di A ed i primi j caratteri di B.

- ***Jaccard***: È una misura di somiglianza per i due set di dati, con un intervallo dallo 0% al 100%. Maggiore è la percentuale, più simili sono le due popolazioni.

$$Jacc_{A,B} = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Sulla base di queste due misure gli autori propongono le seguenti due metriche:

- Distanza tra *tuple valore-token*:

$$d((t_1, v_1), (t_2, v_2)) = \begin{cases} 0, \text{ if } t_1 = t_2 \text{ and } v_1 = v_2 \\ 1, \text{ if } t_1 \neq t_2 \\ \delta(v_1, v_2), \text{ otherwise.} \end{cases}$$

- Distanza tra *User-Based String*:

$$version\_dist_{A,B} = \frac{\sum_{t_i \in A} \sum_{t_j \in B} d((t_i, v_i), (t_j, v_j))}{|A \cap B|}$$

Tutti i *token* della stringa analizzata sono stati codificati con 0 per i numeri, 1 per le lettere e 2 per i valori non alfanumerici.
Successivamente viene calcolata l'entropia della distribuzione di probabilità di tutte le transizioni per ogni *token*, impostando una soglia dell'entropia in base alla distribuzione cumulativa dei valori ottenuti per tutti i *token nel corpus*.
In questo modo le stringhe dall'aspetto casuale avranno una maggiore entropia.

3 Esperimento e risultati

Per condurre l'esperimento sono stati utilizzati due diversi algoritmi:

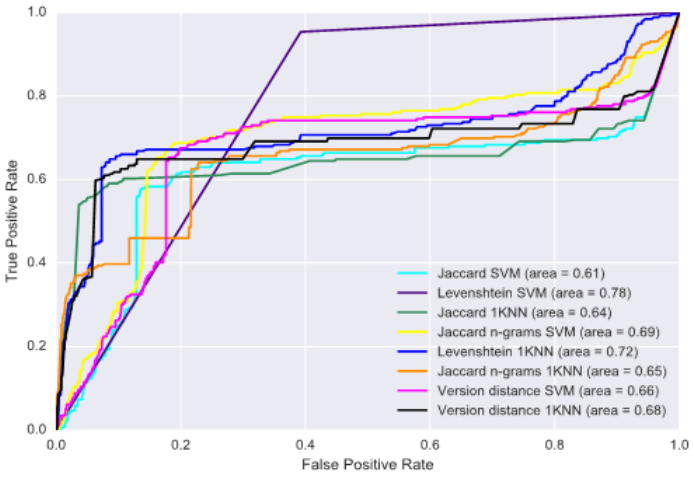
1. ***One-class SVM***: algoritmo senza supervisione che apprende una funzione decisionale per il rilevamento di anomalie. I dati vengono classificati come simili o diversi rispetto ai dati di *train*

2. ***1-KNN***: il punteggio è la distanza dal primo *nearest neighbor*

L'addestramento dei modelli di rilevamento delle anomalie è stato effettuato su dati provenienti da due diverse fonti:

	Size of raw data	Distinct user agent strings
Bluecoat proxy data	7,943,657	2,760
Honeypot data	994,693	4,048

Per stimare l'abilità dei modelli di apprendimento automatico è stata utilizzata la convalida incrociata (*k-fold cross validation*) con $k = 10$.
Per la valutazione dei risultati vengono utilizzati i punteggi dell'AUC (*Area Under the ROC Curve*) che restituisce la probabilità che il modello classifichi un esempio positivo casuale più altamente di un esempio negativo casuale. Più un punteggio si avvicina ad 1, più le previsioni sono corrette.



E' possibile notare che il modello che restituisce la migliore previsione per l'identificazione di *User-Agent String* è è il *Levenshtein SVM*, mentre il modello che restituisce la minor accuratezza è il *Jaccard SVM*.