UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Laurea Triennale in Informatica

Correlazione di allarmi di rete e rilevamento anomalie su serie temporali

Relatore:

Prof: Luca Deri

Candidato:

Francesco Lorenzoni

ANNO ACCADEMICO 2021/2022

Indice

1 Introduzione **7**

1.1 Motivazione e Obiettivi 8

1.2 Struttura della tesi 10

2 Stato dell'arte **11**

2.1 Funzione similarità 11

2.2 Correlazione tramite grafi 13

2.3 Tassonomia tecniche 13

2.4 Periodicità allarmi 14

3 Considerazioni su metodo e contesto **17**

3.1 Procedimento proposto 17

3.1.1 Periodicità 18

3.1.2 Criteri domain specific 18

3.1.3 Allarmi di livello superiore 19

3.1.4 Workflow 20

3.2 Ambiente di test 22

3.2.1 Osservazione ambiente 23

4 Allarmi di livello superiore **25**

4.1 Allarmi di livello superiore 25

4.1.1 Flussi di allarmi 26

4.1.1.1 Paradigma client-server 27

4.1.1.2 Binary Application Transfer 27
4.1.1.3 Periodicità 27
4.1.2 Domain Generation Algorithm 31
4.1.2.1 Falsi Positivi 32
4.1.3 TLS Self-signed Certificate 33
4.1.4 Probing 34
4.1.5 Binary Application Transfer 35
4.1.6 Accesso Remoto 36
4.1.7 Blacklisted host peers 36
4.1.8 Host che generano allarmi eterogenei 36

5 Rating **39**
5.1 Rating come misura 39
5.1.1 Calcolo rating 40
5.2 Rilevamento Outlier 41
5.2.1 Outlier nella singola finestra temporale 43

6 Implementazione **45**
6.1 Considerazioni preliminari 45
6.1.1 Formato allarmi 45
6.2 Struttura codice 47
6.3 Gestione iniziale allarmi 48
6.4 Flussi di allarmi 49
6.4.1 Entropia e paradigma CLI-SRV 50
6.4.2 Periodicità 50
6.4.3 Binary Application Transfer 51
6.5 Singoli Allarmi 52
6.5.1 Binary Application Transfer 53
6.5.2 Suspicious DGA Domain 54
6.5.3 TLS Certificate Self-signed 57

6.5.4 Probing 58
6.5.5 Remote Access 58
6.5.6 Blacklisted Peers 59
6.5.7 Host che generano allarmi eterogenei 59
6.6 Output allarmi 60
6.7 Rating e Outlier 63
6.7.1 Rating 63
6.7.2 Rilevamento Outlier 64
6.7.3 Rappresentazione Outlier 66

7 Validazione 69

7.1 Triage e Ambiente test 70
7.2 Anomalia 1 71
7.3 Anomalia 2 75
7.4 Anomalia 3 77
7.5 Anomalia 4 79
7.6 Anomalia 5 80
7.7 Anomalia 6 81
7.8 Anomalia 7 82
7.9 Anomalia 8 83
7.10 Anomalia 9 84
7.11 Anomalia 10 85
7.12 Analisi su Singola Finestra 86
7.13 Tabella Riassuntiva 87

8 Conclusioni 89

8.1 Requisiti 89
8.2 Obiettivo 90
8.3 Miglioramenti futuri 91
Ringraziamenti 93

A Appendice **95**

A.1 Codice sorgente 95

A.2 Richiami teorici 95

A.2.1 Entropia 95

A.2.2 Deviazione Standard 97

A.2.3 Media e outlier 98

A.2.4 Strutture dati 99

A.3 Flussi 100

A.4 Snippet codice 100

Capitolo 1

Introduzione

Il tema della sicurezza nelle reti informatiche ha assunto ormai da alcuni anni un peso importante.
Attacchi informatici possono portare a importanti disservizi e danni economici o a gravi violazioni di privacy, con ripercussioni talvolta anche significative sulla vita delle persone.

Esistono vari strumenti volti alla gestione di questo aspetto, fra cui gli IDS: *Intrusion Detection System*. Gli IDS sono prodotti software e/o hardware di monitoraggio atti a identificare e riportare, attraverso l'esportazione di allarmi, attività malevole, come accessi esterni indesiderati e vari attacchi, fra cui Denial of Service, SQL Injection, basati su Exploit, o provenienti da macchine interne infette da malware. Possono essere suddivisi in varie categorie, in base alle analisi che svolgono: le principali distinzioni riguardano il metodo di rilevamento delle intrusioni e il traffico esaminato. Il sistema di rilevamento può essere signature-based, anomaly-based o ibrido.
Signature-based significa confrontare il traffico, anche sui singoli pacchetti al livello 3 modello ISO/OSI, con dei pattern (*firme*) in un database di intrusioni note; i sistemi anomaly-based invece sono tipicamente basati sulla costruzione di un modello attraverso il Machine Learning che definisca l'attività normale della rete e su questa

individui anomalie indicanti possibili intrusioni.
Gli IDS possono distinguersi anche in Network-based IDS (*NIDS*) e Host-based IDS (*HIDS*), in base al traffico esaminato: i NIDS monitorano un'intera rete, mentre gli HIDS si limitano ad un singolo nodo. Noi ci concentreremo sugli allarmi esportati da ntopng, software open-source di monitoraggio che presenta delle feature associabili a un NIDS.

Gli IDS si distinguono dagli IPS (*Intrusion Prevention System*) per limitarsi alla sola segnalazione delle possibili intrusioni, che verrà esaminata da un addetto che agirà di conseguenza se necessario, mentre gli IPS possono attivamente bloccare del traffico in seguito al rilevamento di un problema. Si possono trovare sul web articoli di facile lettura che illustrano le caratteristiche degli IDS con potenzialità e limiti.

1.1 Motivazione e Obiettivi

Il volume massivo di allarmi generati in reti topologicamente complesse rende difficile il controllo degli stessi, essendo quest'ultimo spesso svolto manualmente dagli amministratori di rete. Gli addetti tipicamente tendono ad utilizzare più di un sistema di rilevamento intrusioni (*IDS*) per poter sopperire alle mancanze degli uni con la presenza degli altri, portando la quantità di allarmi da gestire a crescere ulteriormente, e ad una probabile ridondanza fra questi.

Il lavoro dell'addetto alla sicurezza della rete è esaminare allarmi e avvisi forniti dai software, discernere a prima vista i falsi positivi dai sospetti problemi, e poi verificare se questi ultimi indichino dei rischi per la sicurezza o dei comportamenti propri del normale traffico della rete (come protocolli obsoleti o l'invio di report e log), oppure se siano anch'essi falsi positivi, causati da uno sbaglio del software di rilevamento.

È un lavoro oneroso, ed è presente il rischio concreto che l'addetto impieghi tempo ad esaminare allarmi sintomo di situazioni non pericolose, causate dal normale traffico, e invece gli sfugga un vero rischio. Proporre all'amministratore dei problemi almeno apparentemente più critici rispetto ad altri significherebbe permettergli di dedicare del tempo prima a situazioni con priorità maggiore e poi al resto. È difficile fornire una classificazione sicura degli allarmi come gravi o innocui, ma si possono fare dei tentativi per avvicinarvisi e fornirne una qualitativa.
Molti allarmi sono causati dal normale traffico della rete e non rappresentano consistenti pericolosità; dunque mettendo in correlazione alcuni allarmi e definendone un andamento analizzandoli nel tempo, le anomalie individuabili su tale andamento potrebbero essere indice di un problema più critico.

L'**obiettivo** della tesi è definire un modello che descriva il comportamento dei nodi di una rete in termini di allarmi esportati da un software di analisi del traffico, mettendo questi ultimi in correlazione, evincendone le criticità e analizzandoli nel tempo, al fine di individuare e notificare all'amministratore di rete asset quasi, se non del tutto, critici e meritevoli di attenzione; infine, implementare il modello e verificarne funzionamento e validità.
Non possiamo ambire ad avere precisione e certezza assolute, in quanto le situazioni difficilmente hanno una classificazione binaria, e inoltre appare spesso cruciale per una valutazione corretta la validazione da parte dell'amministratore di rete, il quale deve poter avere l'ultima parola.

In altre parole, vogliamo sapere *quali* host hanno riportato un comportamento anomalo, *quando* lo hanno fatto, e in *cosa* consiste tale comportamento. Così facendo, auspichiamo, l'amministratore di rete potrà concentrarsi su una cerchia ristretta di situazioni da indagare, invece che su migliaia di allarmi.

1.2 Struttura della tesi

In questo primo capitolo (1) abbiamo introdotto il contesto e quali sono gli obiettivi. Nel secondo (2) sono descritti alcuni studi precedentemente svolti da terzi sulla correlazione di allarmi.

Nel terzo capitolo (3) viene descritto il lavoro svolto nei mesi di studio.

I successivi due capitoli (4,5) riguardano l'architettura della soluzione proposta. Nel capitolo 4 vengono discussi in dettaglio i criteri di criticità utilizzati per generare gli allarmi di livello superiore, senza riferimenti all'implementazione di questi, discussa invece nel capitolo 6.

Nel breve quinto capitolo (5) si illustra come viene calcolato il rating per gli host, come si costruiscono le serie temporali su questi, e come vengono generati i risultati e i grafici che li rappresentano.

Vi è un settimo capitolo (7) dedicato alla validazione dei risultati proposti, volta ad illustrare in alcuni casi situazioni critiche e in altri, invece, dei falsi positivi.

Infine vi è un ultimo capitolo (8) dedicato ai miglioramenti futuri e alle conclusioni, con riferimenti agli obiettivi inizialmente prefissati.

Capitolo 2

Stato dell'arte

Negli anni sono stati svolti diversi studi riguardanti l'*event correlation* nell'ambito delle reti, molti dei quali discutono come eseguire clustering di allarmi, al fine di ridurne il numero e facilitarne la gestione.
Studi come [40][6] si concentrano anche sull'aggregare allarmi esportati da IDS differenti, dopo averli normalizzati e ricodificati in un formato univoco, aspetto che esula dai nostri interessi.
Alcuni [30][41] propongono metodologie che non fanno importanti assunzioni sul tipo di allarme esaminato e cercano di aggregare allarmi svincolandosi da esso, definendo una *funzione di similarità*, che fornisca una stima di quanto due allarmi siano "aggregabili" (simili), e poi applicando algoritmi di learning che ottimizzino tale funzione. Ad esempio in [18] viene effettuato clustering di allarmi attraverso delle Self Organizing Maps (SOM), i cui pesi e le feature selezionate vengono ottimizzati sfruttando la Particle Swarm Optimization (PSO).

2.1 Funzione similarità

Ci sono alcuni studi volti a gestire specifici scenari, come [17] svolto da Lee et al., che per individuare attacchi DDoS aggregano allarmi tramite una funzione *distanza* (speculare alla *similarità*[38]) basata sulla distanza Euclidea, calcolata su alcune

feature degli allarmi.
Sulla selezione delle feature Vasquèz et al [12] hanno svolto un'analisi su quali siano effettivamente utili per la rilevazione di allarmi. Visto che si verifica la criticità rispetto a una cerchia finita di criteri noti e definiti 3.1, il modulo sviluppato attua una selezione degli attribuiti limitandoli a quelli necessari per tali criteri.

La funzione distanza come criterio di aggregazione allarmi viene usata anche in [15] e [31], in cui utilizza una gerarchia dei valori possibili sugli attributi dove la differenza matematica $\Delta = a - b$ non è possibile, e un algoritmo genetico interattivo *IGA* (Interactive Genetic Algorithm) per costruire i cluster di allarmi, sfruttato anche in [3] e qui comparato con altri algoritmi genetici. Anche in [27] viene utilizzata la gerarchia come metrica per stabilire la similarità fra allarmi.

Sun et al [13] propongono una forma di learning supervisionato, che prevede una fase di scelta dei pesi degli attributi dispendiosa in termini di risorse, e poi la definizione di una funzione di similarità basata su tali attributi con relativi pesi, in base alla quale aggregare allarmi simili, considerando una finestra temporale scorrevole. I risultati ottenuti mostrano un alto livello di aggregazione, soprattutto per attacchi volumetrici come *DDoS goldeneye* e *slowloris*. Rispetto a quanto vogliamo fare noi, non ci interessa la sola aggregazione, e vorremmo poter scendere nel dettaglio del tipo di allarme, almeno in alcuni casi.

Strumenti simili

Ci sono anche studi che si allontanano da quanto svolto in questa tesi nonostante ne condividano alcuni strumenti, ad esempio Qin et al [21] che applicano il test di causalità di Granger[34][26] su serie temporali per stabilire una correlazione fra allarmi, o [5] in cui si controlla se vengono rispettate o meno delle regole pre-definite dall'utente.

2.2 Correlazione tramite grafi

È possibile anche usufruire dei grafi per correlare allarmi: in [14] vengono definite delle "tassonomie" degli attributi rappresentate da alberi, in cui la similarità fra due allarmi dipende dalla distanza in numero di nodi fra due foglie x e x'. In [16] si costruisce un grafo in cui ogni nodo è un allarme e il peso rappresenta la possibile relazione fra allarmi, dopodiché vengono utilizzati degli algoritmi di *Graph Embedding*[35], per rappresentare come vettori gli allarmi e le relazioni, e infine questi ultimi vengono aggregati con una variante di *K-Means* e *AHC* ([4]).

2.3 Tassonomia tecniche

Negli anni sono stati redatti degli articoli sintetizzanti le soluzioni proposte in letteratura riguardanti la correlazione di allarmi nelle reti. Salah et al [23] fornisce una tassonomia delle metodologie di alert correlation, citando e classificando alcune delle tecniche disponibili fino al 2013, anno di redazione del sondaggio. In [1] vengono elencate e brevemente commentate diverse strategie di alert aggregation.
Per noi sono di poco interesse metodi che richiedano:

- Machine Learning: vorremmo poter raggiungere gli obiettivi evitando che sia un algoritmo automatizzato ad inferire il modo in cui aggregare gli allarmi, ma piuttosto definendo (alcuni) principi di aggregazione e criticità, implementando un modello che ne sfrutti peculiarità e caratteristiche, e realizzando del codice che sia ottimizzato specificatamente per questi. Nell'ultimo mese è stato finanziato e avviato un costoso progetto [8] di 5 anni volto ad integrare innovativamente l'intelligenza artificiale nella cybersecurity.

- Learning supervisionato o regole expert-based: gli alert esportati da un NIDS come quello da noi utilizzato sono spesso strettamente dipendenti dal contesto in cui vengono generati, dunque degli esempi di training forniti da terzi

potrebbero risultare inefficaci; in aggiunta, affidare all'amministratore di rete una onerosa supervisione dell'apprendimento di un modulo, significherebbe ulteriore impegno per quest'ultimo, a cui invece vorremmo semplificare il lavoro. Inoltre anche delle regole specifiche[1] o esempi di training realizzati ad hoc potrebbero rivelarsi obsoleti in un secondo momento, dato che la topologia di rete e il corrispondente traffico sono un contesto mutabile nel tempo.

- Algoritmi di elevata complessità asintotica, che quando applicati a grandi quantità di dati si rivelano impegnativi in termini di risorse e possono richiedere del tempo prima di fornire un risultato.

Possono essere invece di interesse strategie matematiche e statistiche in fase di aggregazione e analisi.

2.4 Periodicità allarmi

Un importante, ma non unico, parametro nel determinare la criticità degli allarmi è la **periodicità** fra di essi, ovvero la regolarità nel tempo che intercorre fra l'esportazione di due allarmi consecutivi. In letteratura sono state proposte alcune soluzioni per individuarla e utilizzarla nella correlazione di allarmi.

Nonostante lo studio sia datato, Viinika, Debar et al [29] propongono un approccio semplice e piuttosto efficace, basato sull'analisi delle serie temporali. Aggregano alert generati da una stessa signature in un *alert flow* e per ciascuno individuano trend, periodicità, fattore random (rumore) e infine la rimanente parte è "anomalia", che se significativa genera un alert. Gli svantaggi sono la necessità di dover eseguire la stima di alcuni parametri, e soprattutto l'assunzione di sapere quale sia la periodicità degli eventi analizzati. Inoltre le statistiche da loro calcolate sono efficaci perché gli

[1] Ad esempio: $Host\ X$ esporta flussi $\neq SMTP \Rightarrow$ Allarme

allarmi esaminati sono relativi ad appena 5 signature e di tipologie particolarmente specifiche.
Lo studio evidenzia come limitarsi ad alcuni allarmi per sfruttare proprietà in essi insite possa essere efficace, ma non è applicabile al nostro scenario del tutto *"as-is"*, anche a causa della natura di `ntopng`, che non è strettamente signature based come (ad esempio) *Suricata* o *Snort*, usato nello studio; tuttavia l'idea della aggregazione in *flussi di allarmi* la riprenderemo anche noi, così come l'intuizione dell'usare il tipo di allarme per inferire informazioni aggiuntive.

Riguardo a ciò, [28] considera dei **flussi di allarmi** e per ciascuno di questi costruisce una serie temporale sull'intensità di allarmi. Dunque, determina tutti gli allarmi associati a una certa signature e tiene conto di quanti allarmi vengono generati ogni X minuti, ovvero l'*intensità*, e trova gli outlier[2] predicendo l'intensità attesa tramite l'*exponentially weighted moving average*[33].
È una strategia efficace solo in caso di una quantità di allarmi importante per ciascuna signature. Nel caso di studio esaminato in questa tesi si osserva un'intensità dei flussi raramente superiore a 3-4 allarmi ogni 60 minuti, che significa avere una popolazione di 2 o 3 osservazioni, quantità insufficiente per fare predizioni statistiche come quelle effettuate nello studio. È possibile far lievitare questo numero considerando allarmi di livello inferiore, che tuttavia sono di minore interesse, oppure considerando l'intensità di flussi ottenuti ignorando la tipologia degli allarmi (ciò che più si avvicina a una "signature" in `ntopng`), e considerando solo l'host che li ha generati.

In [24] si utilizza il *g-test* di Fisher (Fourier Transform) per distinguere eventi periodici (computer generated) ed eventi aperiodici (interazione umana), basandosi esclusivamente sui tempi di arrivo degli eventi. Tuttavia non si spinge molto oltre ciò. Il *g-test* può dunque essere valido per determinare la periodicità, ma preferiremo

[2] In statistica, data una serie di osservazioni si definisce *outlier* un valore anomalo che si discosta da esse[37]

l'approccio usato in [10], che è più semplice, basato sull'assunzione che la successione di intervalli di tempo fra un evento e l'altro in caso di flussi periodici avrà una bassa deviazione standard. È un sistema facilmente implementabile e a basso costo computazionale. È inoltre la stessa strategia utilizzata da `ntopng` per determinare l'eventuale periodicità dei *flussi*.

Capitolo 3

Considerazioni su metodo e contesto

3.1 Procedimento proposto

Spesso il traffico di rete ordinario può apparire sospetto ad un IDS ed è difficile distinguere i problemi effettivi dal cosiddetto "rumore".

Per tentare di attuare tale distinzione, innanzitutto definiremo alcuni **criteri** di criticità che useremo per calcolare un ***rating*** da associare a ciascun host che li soddisfa.
Ripetendo tale calcolo nel tempo, possiamo ottenere una traccia di quale sia il comportamento ordinario, in termini di sospetti problemi, di ciascun host, e individuare quando vi è un cambio significativo rispetto a questo, e da cosa è stato causato.

Riguardo ai sopracitati criteri di criticità, ci interessa individuare allarmi con caratteristiche comuni o comunque correlabili, al fine di generare degli ***allarmi di livello superiore*** (talvolta chiamati *meta-alert* in letteratura) come risultato di tale correlazione.

Tali criteri dovranno guidarci nel determinare quali allarmi mostrano **periodicità** e quali no, e in alcuni casi quali sono del tutto aperiodici; oltre a ciò ci interessa sfruttare le **proprietà specifiche** di alcune tipologie di allarmi per evidenziare quali di questi sono maggiormente critici rispetto agli altri.

3.1.1 Periodicità

Una successione di allarmi con attributi comuni (*flusso* di allarmi) mostra **periodicità** se vi è una regolarità nel tempo che intercorre fra l'esportazione di un allarme e il successivo. La periodicità fra allarmi è uno dei più rilevanti indici di comportamento potenzialmente sospetto: ad esempio può indicare telemetria o aggiornamenti indesiderati dovuti a configurazioni errate del software, oppure rivelare scanning[20], esfiltrazione dati o beaconing[19][22], come studiato in [2].
La periodicità ha le caratteristiche di essere una relazione relativa solo a successioni di allarmi, non a uno singolo, e di essere scollegata dal tipo di allarme in questione.

Specularmente anche allarmi **aperiodici**, che si verificano una sola volta, possono essere di altrettanto interesse, in quanto possono essere indice di un cambio nel comportamento di un host.

Questi aspetti verranno discussi più in dettaglio e con l'ausilio di esempi nella sezione 4.1.1.3.

3.1.2 Criteri domain specific

Periodicità a parte, scendendo nel dettaglio e nelle proprietà implicite di alcuni specifici tipi di allarmi, possiamo sfruttare queste ultime per scovare comportamenti maggiormente critici, con il limite di poterci concentrare solo su una finita cerchia di allarmi, ma più efficacemente.

Prendiamo come esempio il *port scanning* (o *probing*), tecnica utilizzata per scovare su una macchina servizi attivi e/o eventuali vulnerabilità, che consiste nell'invio di molteplici richieste a uno stesso host su porte diverse, generando molto traffico unidirezionale:

$$client \longrightarrow server$$

Si immagina che il traffico sia unidirezionale qualora il server non sia configurato per rispondere, o più semplicemente non vi siano servizi attivi sulle porte contattate.

Il probing non è identificabile partendo da un singolo flusso (A.3), ovvero una singola comunicazione identificata da ⟨`server_ip, server_port, client_ip, client_port , proto`⟩, ma conoscerne la natura ci permette di scoprire quando è in atto, cercando macchine server contattate unidirezionalmente su porte diverse, e dunque mettendo in correlazione allarmi distinti che condividono tipologia di traffico e host target.

Questo tipo di strategia è efficace per il probing, ma non applicabile ad attacchi di altro tipo, dunque è specifica per esso.

Il probing e altri criteri domain-specific sono discussi nelle sezioni 4.1.2 e successive.

3.1.3 Allarmi di livello superiore

Quindi per inferire degli *allarmi di livello superiore* dovremo:

Aggregare in flussi (A.3) allarmi con caratteristiche simili • Determinare flussi di allarmi periodici • Determinare flussi critici rispetto a criteri noti
Determinare allarmi aperiodici critici rispetto a criteri noti

Tabella 3.1: Allarmi di livello superiore

3.1.4 Workflow

Introduciamo il procedimento logico dietro al modello, implementato dal tool sviluppato: si considerano 24 finestre temporali di 60 minuti, per un periodo di osservazione complessivo di un giorno, e per ciascuna di queste vengono inferiti degli **allarmi di livello superiore**, basati su alcuni criteri di criticità da noi scelti schematizzati nella tabella 3.1, e sulla base di questi viene calcolato un **rating** per ogni host. In questo modo otterremo una serie temporale dei rating per ciascun host, e su questa potremo determinare degli outlier[37] (ovvero valori anomali che si discostano significativamente dalle altre osservazioni) usando metodi statistici, rappresentanti dei cambi di comportamento e potenzialmente dei problemi. Infine, potremo offrire all'amministratore di rete una vista che mostri quali host hanno deviato rispetto al loro comportamento "ordinario", quando lo hanno fatto, e cosa ha causato tale cambio.

Analisi su singola finestra Successivamente discuteremo anche della possibilità, che per ora lasciamo da parte, di confrontare i rating degli host in una singola finestra temporale, e trovare fra questi dei valori outlier.

Inferire *allarmi di livello superiore* in una finestra temporale di 60 minuti
Calcolare *rating* per ciascun host in base a essi
Costruire *serie temporale* dei rating di ciascun host ripetendo il calcolo 24 volte
Individuare gli outlier sulle serie temporali generate
Organizzarli in una vista che indichi host, finestra temporale di riferimento, e criteri di criticità rispettati

Tabella 3.2: Requisiti

Quanto espresso nella tabella 3.2 rappresenta gli obiettivi che hanno guidato la realizzazione del tool. Le analisi sui dati svolte durante lo sviluppo hanno guidato la scelta dei criteri e il tuning dei parametri.

Non abbiamo garanzia che gli outlier così individuati indichino con certezza un problema: un cambio di comportamento, seppur modellato su degli allarmi, non è necessariamente sintomo di una minaccia, ammesso che sia un effettivo cambio e non un falso positivo. Nel capitolo 7 dedicato alla validazione esamineremo dei risultati ottenuti dal tool, mostrando come alcuni di questi possano rappresentare effettive criticità e altri dei falsi positivi.

Per stabilire la dimensione delle finestre temporali da considerare sono stati svolti test con vari minutaggi, e 60 minuti è stata la scelta che sembrava portare risultati migliori: con finestre di dimensione inferiore, si rischiava di perdere periodicità[1], mentre scegliendo dimensioni maggiori, si potevano considerare periodici anche eventi sporadici, oppure perdere periodicità qualora si fosse verificata in un lasso di tempo ristretto (se un allarme viene esportato 6 volte a intervalli di 4 minuti e poi altre 7 volte su base irregolare, considerando finestre temporali molto ampie si perde l'informazione sulla periodicità mostrata inizialmente nelle prime 6 osservazioni).

Nonostante l'analisi, la discussione di questa e la soluzione proposta siano legate a `ntopng`, i concetti e le strategie applicate non lo sono e queste possono essere astratte.

Il modello è stato implementato in **Python**, che è un linguaggio che astrae molto dall'implementazione a basso livello, privilegiando leggibilità e semplicità, talvolta a scapito delle performance.

La scelta del linguaggio è stata dettata innanzitutto dalla sua leggibilità, auspicando di permettere facile astrazione di concetti e strategie, e di rendere più semplice

[1] in una finestra di 10 minuti è impossibile osservare eventi che avvengono ad intervalli di 6 minuti

applicare ed eventualmente implementare gli stessi in un contesto più specifico.

3.2 Ambiente di test

Studio, analisi e test sono stati svolti in una rete di un'azienda della grande distribuzione (GDO), il gruppo *Unicomm S.p.A*. La rete è protetta da Firewall, sistemi di prevenzione delle intrusioni (*IPS*) e altri IDS, dunque nei fatti è piuttosto pulita e sicura, ma vista la complessità mostra una discreta quantità di comportamenti potenzialmente sospetti. Per avere un'idea dell'entità del problema, `ntopng` esporta ogni 30 minuti fra 5 e 7 migliaia di allarmi di severità elevata.

Essendo la rete molto protetta, gran parte di essi è rumore, ma rimane una quantità ingestibile manualmente. Ambire a trovare *tutti* e *soli* i problemi "veri" è eccessivo, ma si può tentare quantomeno di evidenziare alcune delle possibili criticità, considerando informazioni non fornite esplicitamente dal software, o comunque non in un formato direttamente fruibile, che altrimenti l'utente responsabile dovrebbe controllare a mano.

Il motivo dietro al tanto rumore è che alcuni allarmi possono essere di scarso rilievo per una certa rete e gli applicativi eseguiti dalle macchine che vi appartengono. Prendiamo come esempio allarmi dei seguenti tipi, i primi tre riguardanti comunicazioni su protocolli obsoleti e l'ultimo sull'invio di credenziali in chiaro:

Samba Insecure Version
SSH Obsolete Client
TLS Old Protocol Version
Clear-Text Credentials

Tabella 3.3: Allarmi Basso Rilievo

Possono tutti, i primi tre in modo lampante, essere facilmente causati da macchi-

ne *volutamente* non aggiornate, a causa del periodo più o meno lungo di disservizio che si verifica in seguito a un aggiornamento, sia qualora questo dovesse implicare il rinnovo hardware di macchine datate che non supportano software più recente, sia qualora questo riguardasse l'aggiornamento dei soli applicativi software.
Spesso il periodo di disservizio e l'impegno richiesto per svolgere l'aggiornamento scoraggiano quest'ultimo e portano a posporlo finché non mostra un grosso vantaggio in termini di feature o sicurezza, o, in altri termini, diventa necessario.

Nel migliore dei casi questi allarmi sono poco rilevanti, in quanto l'amministratore di rete potrebbe reputare che la configurazione, seppur obsoleta, delle macchine vada bene e nel contesto di utilizzo sia sicura, oppure potrebbe decidere che vada aggiornata e adoperarsi per fare ciò; in ogni caso, tali allarmi sarebbero comunque di basso rilievo, in quanto di per sé non evidenziano una minaccia, bensì possibili vulnerabilità, al massimo utili nell'indagine di un problema più grave.

3.2.1 Osservazione ambiente

La costruzione del modello è stata accompagnata dall'osservazione del traffico sulla rete di test, coadiuvata dall'amministratore di rete che ha fornito delucidazioni sul ruolo di alcuni host e subnet, e suggerimenti basati sulla sua esperienza e su cosa avrebbe trovato utile da parte di un software di rilevamento.

Nei mesi di studio e sviluppo è stato dunque possibile osservare casi e situazioni in cui sarebbe risultata produttiva la correlazione di allarmi, riducendone ridondanza ed evidenziandone criticità; tali osservazioni sono state fonti di idee e intuizioni poi implementate e testate.
Tramite il contatto con l'amministratore è stato anche possibile determinare alcuni allarmi associabili a rumore, permettendo di filtrarli; in questo modo è stata eliminata dall'analisi una parte, seppur ridotta, della baseline della rete, ovvero del traffico standard, rendendo più efficace l'applicazione del modello.

Capitolo 4

Allarmi di livello superiore

In questo capitolo esamineremo quali sono le strategie dietro alla generazione degli *allarmi di livello superiore*. I dettagli implementativi verranno discussi in dettaglio nel capitolo 6.

Nell'appendice A sono elencati e descritti diversi strumenti matematici e informatici utilizzati nello sviluppo del tool e menzionati in questa tesi.

4.1 Allarmi di livello superiore

Nelle sottosezioni che seguono definiremo i criteri che, quando soddisfatti, porteranno all'esportazione di un meta-allarme di livello di superiore. Ricordando quanto elencato nella tabella 3.1 relativa a tali allarmi, definiremo con esattezza i flussi di allarmi ed esamineremo prima i criteri di criticità a essi relativi, e poi quelli svincolati da questi.

4.1.1 Flussi di allarmi

Definizione flussi

Per delineare una traccia del comportamento degli host può risultare utile mettere in correlazione allarmi che condividono uno stesso host e la stessa tipologia di allarme.
Alcune proprietà non possono essere inferite considerando i singoli allarmi come del tutto a loro stanti, quindi ci serviremo di questa prima forma di aggregazione.
La nozione di *flusso* viene dall'aggregazione dei pacchetti realizzata da NetFlow e IPFIX, volta a fornire un livello di astrazione superiore al singolo pacchetto, che permetta di avere una visione più chiara e leggibile di quali sono le comunicazioni attive in una rete [9].
Questo ci porta a considerare tre modi per costruire le **chiavi** K su cui raggruppare gli allarmi e quindi ottenere dei flussi:

$$\begin{aligned} SRV &: \langle srv, vlan_id, alert_id \rangle \\ CLI &: \langle cli, vlan_id, alert_id \rangle \\ SRVCLI &: \langle srv, cli, vlan_id, alert_id \rangle \end{aligned} \tag{4.1}$$

Dove srv e cli sono un indirizzo IP o un hostname.

Si può osservare che l'organizzazione in flussi riduca la quantità di elementi da esaminare, ma non è sufficiente per raggiungere gli obiettivi posti nel paragrafo 1.1; infatti di per sé non fornisce alcuna informazione in più all'analista di rete, se non il numero di allarmi aggregati in uno stesso flusso.

Esaminiamo le informazioni che possiamo derivare da un singolo flusso di allarmi, risultante dal raggruppamento di questi, usando una delle chiavi poc'anzi definite nella formula 4.1, limitandoci a considerare quelli che aggregano un numero di allarmi superiore a un valore soglia, così da escludere raggruppamenti irrisori.

4.1.1.1 Paradigma client-server

Secondo il paradigma *client-server* i server offrono servizi su determinate porte note, mentre i client comunicano con essi attraverso porte effimere, cambiandole per ogni nuova connessione. Ovviamente esistono eccezioni, come ad esempio un server FTP in *passive mode*, ma in generale un server che comunica su tante porte diverse e/o un client che invece ne usa poche, costituiscono un comportamento non necessariamente pericoloso, ma strano nella maggior parte dei casi. `ntopng` indica server e client per ogni allarme, ma questa informazione può essere validata o integrata da quelle di contesto che può fornire l'amministratore di rete; nell'ambiente di studio e analisi era noto che in determinate VLAN vi erano solo host server, e in altre solo host client.

Matematicamente, dato un flusso di allarmi, possiamo estrarre le serie di interi *srv_port* e *cli_port*, e su ciascuna di queste calcolare l'entropia, normalizzarla e verificare se supera (per i server) o è inferiore (per i client) un certo valore soglia; se sì, allora abbiamo individuato un comportamento strano.

4.1.1.2 Binary Application Transfer

Uno degli allarmi più importanti è *Binary Application Transfer*, che indica che è stato inviato un file binario eseguibile. Può capitare che in un flusso di allarmi di questo tipo il percorso del file trasferito sia sempre lo stesso, indicando ripetuti invii; può essere di maggiore interesse un file che viene inviato una singola volta, come vedremo più avanti, ma anche l'invio ripetuto di uno stesso file può costituire motivo di interesse.

4.1.1.3 Periodicità

Preso un flusso di allarmi, definiamo `tdiff` come l'intervallo di tempo che intercorre fra due allarmi consecutivi. Se gli allarmi sono generati a intervalli di tempo regolari, allora esiste un valore medio da cui `tdiff` non si discosta mai molto.

Possiamo verificare questa condizione in termini matematici utilizzando il coefficiente di variazione su `tdiff`:

$$CV = \frac{deviazione\ standard}{media} \tag{4.2}$$

Quando $CV < \ SOGLIA$ (e.g. 0.85) allora consideriamo il flusso di allarmi *periodico* con periodo `media(tdiff)`.
Maggiore sarà *SOGLIA*, maggiore sarà la tolleranza nel definire flussi come periodici o meno.

Per chiarire discutiamo un esempio di un flusso di soli 3 allarmi: in figura 4.1.2 vi è uno screenshot di `ntopng`. 3 allarmi sono pochi per dichiarare periodicità[1], ma l'esempio è solo a scopo "dimostrativo".

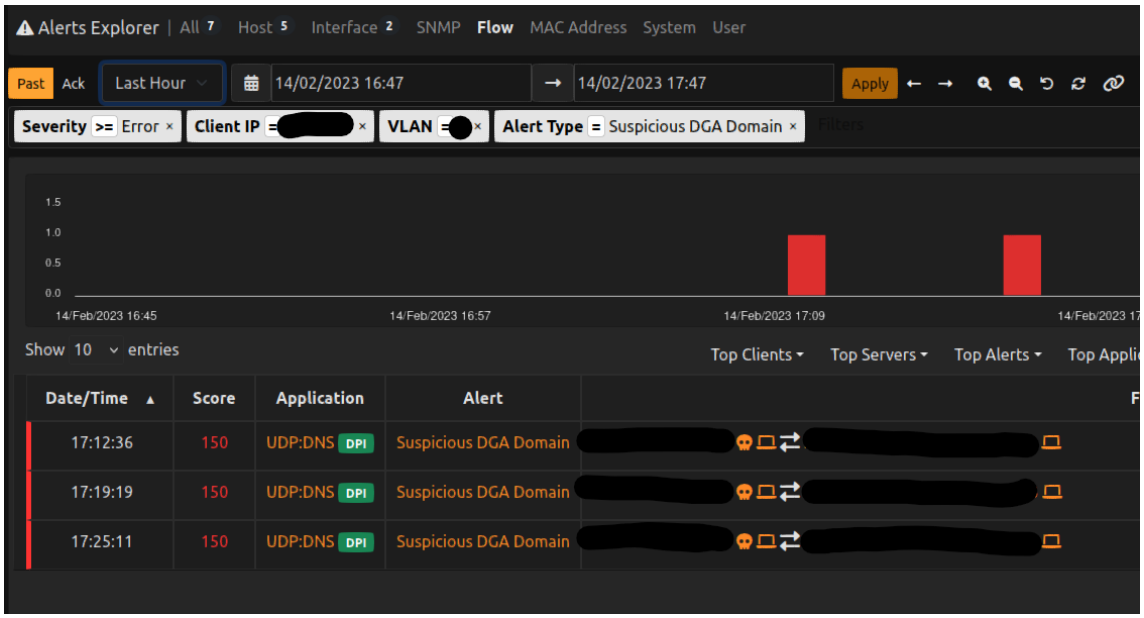


Figura 4.1: Flusso di allarmi periodico

Questi 3 allarmi condividono lo stesso client (oscurato) e la stessa tipologia di allarme (*Suspicious DGA Domain*). Sulla sinistra troviamo l'orario di esportazione

[1]Un flusso di 3 allarmi implica solo 2 osservazioni di `tdiff` su cui calcolare deviazione standard e media

di ciascun allarme, che useremo per calcolare i due `tdiff`, che ricordiamo essere l'intervallo di tempo che intercorre fra due allarmi consecutivi.

```
tdiff_0 = 17:19:19 - 17:12:36 = 00:06:43
tdiff_1 = 17:25:11 - 17:19:19 = 00:05:52
```

Calcoliamo la *media* e *deviazione standard* su `tdiff`:

`avg(tdiff) = 00:06:17 = 377 sec`

`stddev(tdiff) = 36.06`

A questo punto possiamo calcolare il *coefficiente di variazione* (CV):

`CV(tdiff) = 36.06 / 377 = 0.1`

Ipotizzando di fissare $SOGLIA = 0.85$, classificheremmo il flusso sopra mostrato come periodico, con periodo pari a `00:06:17`.

Osserviamo che per alcuni allarmi, ad esempio relativi a protocolli obsoleti[2], è tendenzialmente irrilevante che vi sia una periodicità, mentre per altri, come *Data Exfiltration* (indice di una possibile esfiltrazione dati), può essere critico.

Ci sono due casi limite di una certa rilevanza, che è opportuno evidenziare. Di uno di questi ne è esempio lo screenshot 4.2, in cui si osserva periodicità *"perfetta"*, gli allarmi arrivano ogni 9 minuti esatti, senza variare minimamente.

[2] e.g. *TLS Old Protocol Version*

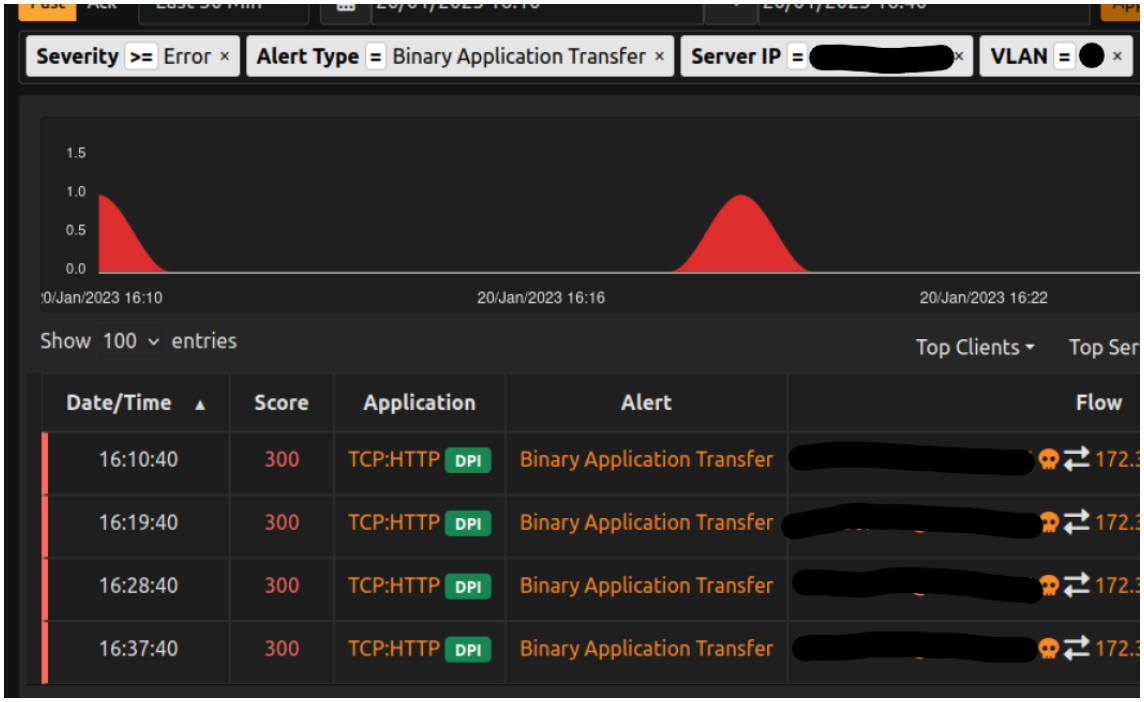
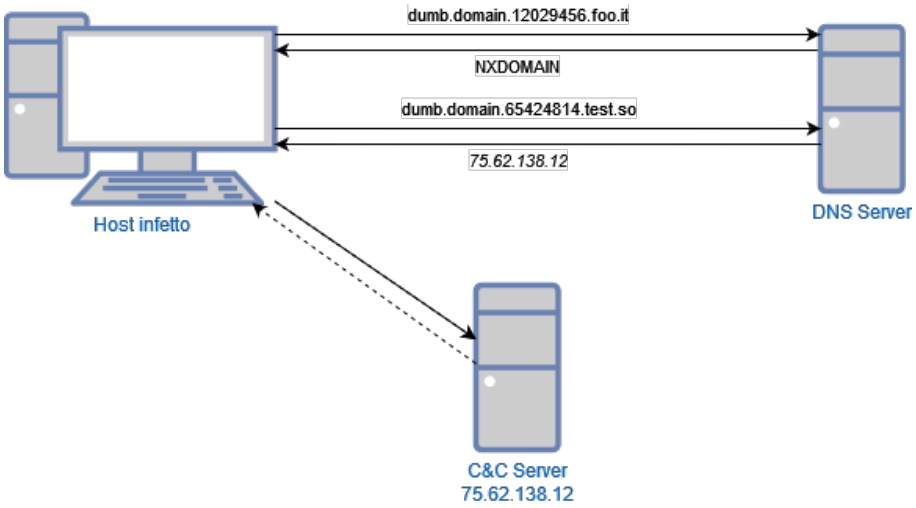


Figura 4.2: Flusso di allarmi con periodicità perfetta

Una situazione del genere quasi certamente non può essere una coincidenza.

Simultanei L'altro caso limite si verifica quando tdiff medio è prossimo a 0, indicando che degli allarmi appartenenti a uno stesso flusso sono arrivati (quasi) *simultaneamente*, situazione indubbiamente inusuale e meritevole di essere segnalata.

Periodicità simile Similmente, può capitare che degli allarmi appartenenti a flussi diversi arrivino con tdiff simili, anche se non esattamente uguali. Ovvero due flussi diversi potrebbero generare lo stesso tipo di allarme con periodicità simile. Dunque macchine distinte potrebbero essere affette da uno stesso problema. Ad esempio, se due macchine venissero infettate da uno stesso malware potrebbero generare allarmi di tipo *Data Exfiltration* ad intervalli di tempo simili.



4.1.2 Domain Generation Algorithm

Negli attacchi di tipo Command and Control (C&C o C2), una macchina infetta cerca di contattare un server remoto, di proprietà dell'attaccante, per ricevere istruzioni. Se tali server fossero pochi e noti, magari descritti in un elenco di stringhe, sarebbe facile individuarli e blacklistarli. Dunque gli attaccanti possono optare per inserire un algoritmo DGA (*Domain Generation Algorithm*) che indichi alla macchina infetta come generare dinamicamente un gran numero di nomi a dominio, di cui molti saranno non registrati, mentre alcuni saranno server C&C attivi che quando risolti verranno contattati per ricevere istruzioni.

Tipicamente domini così generati hanno un aspetto tipo:

`dumb.domain.12029456.foo.it`

oppure

`dumb.domain.12024814.test.so`

C'è una sezione del nome a dominio che rimane tale, mentre le altre vengono variate in modo dinamico.

Quando `ntopng` esporta un allarme di tipo *Suspicious DGA Domain*, possiamo controllare se una porzione, compresa fra due `"."`, del nome a dominio associato all'allarme generato è in comune con uno individuato precedentemente. In questo

modo andiamo a raggruppare gli host che generano domini sospetti in base ai domini stessi, e in particolare alle sezioni fra essi condivisi, che potrebbero indicare un'infezione di uno stesso malware.

Riprendendo l'esempio fatto sopra, assumiamo che vengano generati i seguenti allarmi, con `A,B,C` degli host:

⟨`A,B, dumb.domain.12029456.foo.it`⟩
⟨`A,C, dumb.domain.12024814.test.so`⟩

Tabella 4.1: Esempio Domini DGA

Allora otterremmo il seguente raggruppamento:

```
{"dumb.domain" : [("A","B"),("A","C")]}
```

Si noti che vengono considerate in comune solo porzioni intere del nome comprese fra `"."`, infatti `1202`, prefisso comune della porzione numerica dei due nomi a dominio, rimane esclusa dal raggruppamento.
Rimandiamo al capitolo A.1 per l'implementazione di come avviene l'individuazione delle porzioni di dominio comuni, che, per quanto apparentemente semplice, tende ad essere intricato e verboso.

4.1.2.1 Falsi Positivi

Nell'articolo [25] vengono evidenziate tre cause che possono portare a falsi positivi DGA, di cui la più rilevante è l'utilizzo di cache server da parte di **CDN** (*Content Delivery Network*), come Amazon, Akamai o Cloudflare, per ridurre i tempi di caricamento dei contenuti forniti dal servizio. Spesso vengono utilizzati dei DGA per generare i nomi a dominio di tali cache server, i quali spesso includono un prefisso variabile e un suffisso costante che include il nome del CDN (e.g. `"amazon.it"`).
Altre cause, fra cui le rimanenti due citate, possono essere:

- Il browser *Chrome* che prova a connettersi a tre siti generati casualmente all'avvio
- Codifica *Punycode* di caratteri appartenenti ad alfabeti non latini nei nomi a dominio, che può far apparire sospetti url in realtà regolari.

www.點看.co.uk
↓
www.*xn--c1yn36f*.co.uk

- Servizi di reputazione di siti web che potrebbero generare URL di richiesta o risposta sospetti, qualora vi codificassero dei contenuti.

La quantità di falsi positivi è notevole ed è necessario individuare i DGA maggiormente sospetti. Possiamo far ciò considerando esclusivamente gli allarmi *Suspicious DGA Domain* in cui il client è associato anche ad allarmi di tipo *Long-lived flow* e *Low-Goodput*. Infatti quando si è vittima di un C&C, tipicamente l'host client (ovvero il server C&C, considerando il punto di vista dell'attaccante) invia pochi dati e cerca di mantenere a lungo attiva la connessione.

In questo modo riusciamo a restringere abbastanza la quantità di allarmi da esaminare, limitandoli a quelli maggiormente critici.

Addizionalmente, l'amministratore di rete può whitelistare porzioni dei nomi di dominio, se sa che nella sua rete è normale del traffico verso determinati CDN.

4.1.3 TLS Self-signed Certificate

Per comunicazioni crittograficamente sicure lo standard è il protocollo TLS, che prevede l'esibizione di un certificato per fornire l'autenticazione. I certificati TLS generalmente vengono firmati da un'autorità terza, che ne attesta la validità e vi assegna una scadenza. È possibile firmare autonomamente un certificato, ma ciò viene generalmente percepito come una situazione anomala e potenzialmente rischiosa.

Banalmente anche i più comuni browser bloccano la navigazione verso siti che esibiscono un certificato self-signed.

Tuttavia in un contesto lavorativo alcune macchine interne alla rete potrebbero utilizzare il protocollo TLS al solo fine della cifratura della comunicazione. Dunque una scelta possibile dell'amministratore di rete è di non utilizzare dei certificati firmati da un'autorità terza, operazione non gratuita e che obbligherebbe ad aggiornarli periodicamente, bensì dei certificati generati e firmati autonomamente, che non abbiano scadenza alcuna. In questo modo la gestione degli stessi e degli applicativi che li usano va a semplificarsi molto, inoltre ha anche un costo inferiore, per non dire nullo.

Dunque possiamo limitarci a considerare critici gli allarmi *TLS Self-signed Certificate* solo quando riguardanti host **non** locali.

4.1.4 Probing

Il probing è un tipo di attacco che implica che un attaccante cerchi di contattare un host su svariate porte al fine di scoprire su quali di queste vengono offerti dei servizi, generando traffico (tipicamente) unidirezionale da client (attaccante) a server (vittima).

Possiamo dunque considerare gli allarmi di basso livello *Unidirectional Traffic* e individuare quali sono i server che presentano molteplici allarmi su porte diverse.

Per far ciò, ci torna utile l'entropia menzionata precedentemente. Dopo aver raggruppato gli allarmi in base all'host server, calcoliamo l'entropia sulle porte da esso utilizzate negli allarmi associati per determinare quanto queste ultime siano differenziate.

Dunque se un host è stato contattato unidirezionalmente almeno X volte e l'entropia sulle porte queste supera una certa soglia, lo classifichiamo come vittima di probing.

4.1.5 Binary Application Transfer

Come menzionato precedentemente, può capitare che venga inviato più di una volta uno stesso file eseguibile, su un medesimo flusso o su flussi distinti. È una situazione certamente di interesse e che richiede indagine, ma ci si può giustamente anche aspettare che un file malevolo transiti una sola volta nella rete. Stiamo dunque cercando dei trasferimenti di file del tutto aperiodici.

Ricordando che ad ogni allarme *Binary Application Transfer* (BAT) è associato il percorso del file trasferito, per individuare tali trasferimenti possiamo tenere traccia di tutti i *path* visti in una certa finestra temporale, ed esportare solo gli allarmi associati a path incrociati solo una volta.

Alla strategia di cui sopra, si possono abbinare delle opportune *whitelist* statiche e dinamiche.

Le eventuali *whitelist* possono riguardare i *path* in sé oppure il nome del server associato al trasferimento. Infatti, per un *BAT* può essere noto anche il nome dell'applicativo del server,

Ad esempio, nel contesto di test, molti allarmi *BAT* erano associati con il nome del server *"Microsoft-IIS/10.0"*, e nonostante gli host coinvolti, sia server che client, e i path dei trasferimenti variassero, tale nome rimaneva il medesimo. Tali trasferimenti erano causati da alcuni servizi della rete esaminata che si appoggiano a *Internet Information Services*, una soluzione per Web Server offerta da Microsoft. Dunque possiamo classificare questi allarmi come rumore, ed è sufficiente inserire il nome del server in una whitelist, piuttosto che i singoli host e/o i path, che sono piuttosto variabili e non costanti.

Raggruppando gli allarmi *BAT* in base al nome del server associato si può scoprire che molti di questi sono generati dal normale comportamento della rete, e che possono essere ignorati, o comunque considerati di basso rilievo.

Ad esempio ci sono sistemi di sicurezza, come *Symantec Endpoint Protection* che generano trasferimenti apparentemente sospetti, ma nei fatti di normale funzionamento, ma che lasciano traccia, rendendone semplice l'individuazione e il filtraggio.

4.1.6 Accesso Remoto

Una sessione di accesso remoto, avviata da applicativi di Desktop Sharing o che utilizzano protocolli come SSH o Telnet, fra un server locale e un client con IP pubblico, può essere traccia di una potenziale intrusione.
Tuttavia potrebbe anche rientrare nel traffico ordinario e una sessione potrebbe essere ripetuta più volte, dunque per evitare di generare troppo rumore, ciò che facciamo è generare un allarme la prima volta che una coppia di host (di cui uno locale e uno remoto) esportano un allarme di tipo *Remote Access*.

4.1.7 Blacklisted host peers

Host infetti o attaccati potrebbero comunicare con host appartenenti a blacklist. Non ci interessano gli host ad esse appartenenti, essendo questi già considerati malevoli, ma piuttosto gli host locali (o meno) che comunicano con essi.

4.1.8 Host che generano allarmi eterogenei

In una rete abbiamo detto essere presente del *rumore*, degli allarmi generati dal funzionamento ordinario delle macchine causati da possibili applicativi insicuri, software obsoleto, configurazioni particolari, eccetera.
Può essere ragionevole aspettarsi che gli host contribuiscano in modo più o meno omogeneo a tale rumore. In particolare, non necessariamente in termini di quantità, che può oscillare molto, ma di *tipologia* di allarmi generati. Appare essere un'assunzione forte, ma ha portato a risultati talvolta positivi.

Ad esempio, avere un server con installata una versione insicura di *Samba*[3], porterà ad un allarme *Samba Insecure Version* per ogni nuova connessione verso di esso. Immaginando che un tale server venga usato da più client, tutti genereranno almeno allarmi di quel tipo.

[3]Applicativo per gestire servizi di condivisione file e stampanti

Dunque, immaginando che tutti gli host generino a regime ordinario N tipi diversi di allarmi, possiamo determinare degli outlier sul numero di tipi allarmi generati.

$$A_h = \#\{\text{Tipologie allarmi generati dall'host } h\}$$

Ovvero, calcolato il valore medio $\bar{A}$ di allarmi generati per ogni host, estraiamo gli host h tali che $A_h > \bar{A}$.
Alternativamente possiamo sfruttare lo *scarto interquartile* come descritto in A.2.3

Capitolo 5

Rating

5.1 Rating come misura

Data la lista di allarmi superiori generati come descritto nei precedenti capitoli, calcoliamo il rating di ciascun host iterando su questa. Facciamo corrispondere ad ogni tipo di allarme superiore (4) una *feature*, a cui assegniamo un peso, in alcuni casi in parte variabile: mentre un trasferimento one-time di un file binario si può verificare o non verificare senza vie intermedie, per dei flussi periodici è invece possibile attribuire peso maggiore quando la periodicità è più forte, ovvero quando il coefficiente di variazione, definito nella formula 4.2, è più basso.
Ad ogni allarme di livello superiore, in base alla sua tipologia e al peso associato, corrisponderà un valore numerico, che incrementerà, attraverso una semplice somma, il rating degli host che l'allarme riguarda.

L'obiettivo del rating è di fornire una misura qualitativa, e, vista la natura non binaria delle situazioni trattate, è difficile definire dei pesi come giusti o sbagliati. Sono stati fissati dei pesi iniziali, assegnando valori maggiori a feature indicativamente più critiche, e a questi è seguita una fase di *tuning*, basata sul confronto fra risultati e dati grezzi. Lo scopo di tale procedimento è stato ricalibrare tali pesi in modo che i rating calcolati servissero il loro scopo, e non vi fossero feature eccessi-

vamente predominanti su altre e viceversa, rendendo la misura così ottenuta poco efficace.

Ricordiamo che il rating *non* dà una misura assoluta della criticità di un host, ma ci serve unicamente come strumento per avere una traccia del comportamento del suddetto, e poter su questo individuare dei cambi nel tempo.

5.1.1 Calcolo rating

Forniamo una traccia, con l'ausilio di un esempio, di come viene calcolato il rating. In base al tipo di valore associato a ciascun allarme di livello superiore, vengono estratti gli host (dunque tuple $\langle ip/name, vlan \rangle$) di interesse, e un *rate*, il cui valore dipende dalla feature in questione, ovvero dal tipo di allarme.
Ad esempio, considerando lo snippet di pseudocodice relativo a flussi di allarmi periodici (capitolo 4.1.1.3), viene assegnato un rate base pari a 20, più un "bonus" di al massimo 15, calcolato in base al rapporto fra il Coefficiente di Variazione (formula 4.2) e la soglia sotto la quale un flusso viene considerato periodico.

```
1 PERIODIC_rate = 20
2 PERIODIC_bonus = 15
3 def compute_periodic_rate (alert_flow_key) -> float:
4     percentage = 1 - (stats[alert_flow_key]["tdiff_CV"]/
    PERIODIC_CV_THRESHOLD)
5     bonus = PERIODIC_bonus * percentage
6     return PERIODIC_rate + bonus
```

20 e *15* sono valori qualitativi scelti in seguito alla fase di tuning menzionata nella precedente sezione 5.1.

Così si ottiene il *rate* per un singolo allarme di livello superiore relativo ad un dato flusso, che verrà sommato agli altri rate per l'host[1] ad esso associato. Il valore

[1]Oppure "*gli* host", in caso di un flusso SRVCLI (Cap. 4.1.1)

risultante da questa somma costituirà il rating per tale host.
Infine ribadiamo che essendo gli allarmi di livello superiore relativi a una finestra temporale, anche il corrispondente rating lo sarà, e le anomalie saranno individuate sulla successione di rating relativi a finestre consecutive.

Ricordando l'obiettivo iniziale 1.1, il modello deve descrivere la causa dell'anomalia, oltre a quando si è verificata: il totale dei rate servirà nella rilevazione di outlier, ma si terrà anche traccia di quanto ciascuna feature ha contribuito a tale totale, per conoscere le cause della eventuale anomalia.

5.2 Rilevamento Outlier

Quando si trattano serie temporali[2] è possibile trovare degli outlier[37] (ossia dei valori anomali) facendo una predizione basata sui precedenti valori, a cui viene assegnato peso che può essere fisso o esponenziale. Quando un'osservazione si discosta dalla predizione oltre un valore soglia, viene considerata un outlier.
I metodi di rilevamento degli outlier possono portare a risultati differenti: alcune osservazioni possono essere segnalate come anomale da un algoritmo e classificate come nella norma da altri. I risultati variano in base al metodo utilizzato, a soglie e parametri di tale metodo, e alla serie temporale su cui viene applicato.

Sono stati utilizzati diversi metodi di rilevamento degli outlier, di cui accenniamo il funzionamento, ma per i cui dettagli si rimanda alla bibliografia:

1. **Weighted Moving Average** (WMA)[36]: Per valutare l'i-esima osservazione, si considerano le n precedenti, e su queste si calcola una media pesata per determinare un range di valori in cui l'i-esima osservazione deve rientrare per non essere classificata come anomala.
2. **Exponential Smoothing**[33]: Anche in questo caso si utilizzano i valori antecedenti l'i-esimo per fare una predizione, ma i pesi vengono assegnati espo-

[2] Una serie di osservazioni ordinate rispetto al tempo [39]

nenzialmente. La formula per il Single Exponential Smoothing è la seguente:

$$s_0 = x_0 \qquad t = 0$$
$$s_t = \alpha x_t + (1-\alpha)s_{t-1} \quad t > 0$$

Dove x_t è l'osservazione t-esima, α è il fattore di smoothing, e s_t è il valore predetto per la t-esima osservazione. L'esponenzialità appare evidente sviluppando il termine s_{t-1} della seconda equazione.

(a) **Single Exponential Smoothing** (*SES*/*EWMA*)

(b) **Holt Linear** (Double Exponential Smoothing)

(c) **Holt Winters** (Triple Exponential Smoothing)

In linea generale si utilizzano *WMA* e *SES* quando una serie temporale non ha né *trend* né *stagionalità*, *Holt Linear* quando ha solo *trend*, e *Holt Winters* quando li ha entrambi.

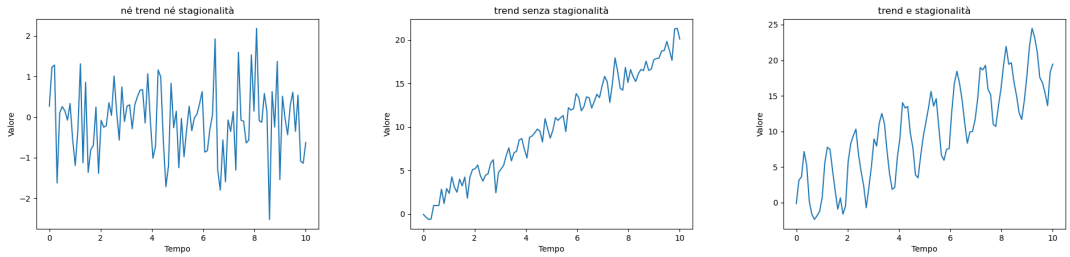


Figura 5.1: I tre tipi di serie temporale menzionati

Considerate delle finestre temporali consecutive, per ciascuna di queste finestre vengono inferiti gli allarmi di livello superiore e i relativi rating, ottenendo per ogni host una serie temporale in cui l'i-esimo elemento è il rating di tale host nella i-esima finestra.

Sono state apportate alcune accortezze nell'implementazione algoritmi affinché venissero gestite serie sparse (con molti valori pari a 0) e scartati valori eccessivamente bassi.

La fase di tuning dei pesi delle feature, menzionata precedentemente, ha toccato anche i vari parametri che determinano la tolleranza nella rilevazione degli outlier degli algoritmi citati, per evitare che rispetto ai dati esaminati questi non fossero eccessivamente sensibili o insensibili alle oscillazioni nelle osservazioni.

Essendo le serie temporali in questione composte da relativamente pochi punti e non essendoci sempre una stagionalità evidente, Holt Winters può generare dei risultati non opportuni.

La WMA e il SES sono i più indicati per le serie ottenute dal contesto in esame, ma in qualche caso i dati hanno mostrato un certo trend, dipendente dagli orari lavorativi, e Holt Linear ha fornito risultati utili.

5.2.1 Outlier nella singola finestra temporale

Nel capitolo introduttivo 3.1.4 è stata menzionata la possibilità di effettuare anche un'analisi aggiuntiva, ovvero determinare gli outlier in una singola finestra temporale, comparando i rating di host distinti, piuttosto che i rating ottenuti nel tempo associati ad uno stesso host; dunque tali outlier sono gli host che hanno ottenuto dei rating particolarmente diversi rispetto agli altri, considerando un intervallo di tempo di soli 60 minuti.

Per un'analisi di questo tipo si può usare il metodo dello scarto interquartile A.1.

Tuttavia, è necessario poter fare la forte assunzione che ci si aspetti dagli host esaminati un comportamento simile, altrimenti la comparazione fra questi perde di significato. È naturale che, ad esempio, un mail server possa avere un rating fortemente diverso da un dispositivo IoT, e semmai sarebbe strano il contrario.

Capitolo 6

Implementazione

6.1 Considerazioni preliminari

Il modulo è stato realizzato sfruttando l'API Python di `ntopng`, disponibile sulla pagina github, che in sostanza gestisce l'invio di query all'API RESTful di `ntopng` e la ricezione delle conseguenti risposte.

Prima di entrare nel dettaglio di ciò che è stato fatto è opportuno fornire alcune informazioni su come vengono esportati gli allarmi esportati da `ntopng`, anche e soprattutto per dare un'idea di quali sono i principali attributi di un allarme prodotto da un IDS, piuttosto che per discutere dello specifico formato usato da `ntopng`.

6.1.1 Formato allarmi

Le risposte dell'API RESTful sono fornite in formato `json`, e ogni allarme è un oggetto. Ricordiamo che uno dei nostri obiettivi sarà organizzare gli allarmi in *flussi*.

Elenchiamo sotto quali sono i principali campi per noi di interesse, benché ve ne siano anche altri non riportati per semplicità di lettura.

```
1 "srv_ip": "172.31.254.32",
```

```
2 "cli_ip": "192.168.1.160",
3 "vlan_id": 15,
4 "alert_id": 24,
5 "srv_name": "server.name.com",
6 "cli_name": "client.name.com",
7 "tstamp": "2022-12-30 10:30:38",
8 "score": 200,
9 "cli_port": 52848,
10 "srv_port": 443,
11 "is_srv_attacker": 0
12 "is_cli_attacker": 0,
13 "is_srv_victim": 0,
14 "is_cli_victim": 0,
15 "json": {}
```

I primi 4 campi costituiscono le tuple in base alle quali costruiremo i suddetti flussi di allarmi,

$$\langle srv_ip, cli_ip, vlan_id, alert_id \rangle$$

`alert_id` è un intero che indica il tipo di allarme generato da `ntopng` e a ciascun intero corrisponde una stringa, ad esempio

$$alert_id = 24 \Rightarrow alert_name = \texttt{tls_old_protocol_version}$$

`tstamp` indica, banalmente, il momento in cui l'allarme è stato generato. `score` è il punteggio assegnato da `ntopng`: più è alto, più l'allarme è critico. È un campo importante che fornisce una buona approssimazione della rilevanza di un allarme, e che, seppur non cruciale nel lavoro realizzato e qui discusso, si è rivelato molto utile in fase di studio e analisi dati.

Contrariamente a quanto succede tipicamente nei flussi di pacchetti, (e.g. Netflow) `(srv|cli)_port` *non* fanno parte della tupla che identifica i flussi di allarmi. Come verrà mostrato più avanti, per le nostre analisi risulta conveniente che sia così, in quanto ciò permette di riconoscere con facilità comportamenti potenzialmente sospetti.

`ntopng` in alcune situazioni è in grado di discernere host attaccante e host vittima. È un'informazione che risulta utile nell'individuazione di host che stanno potenzial-

mente facendo danni nella rete, permettendoci di escludere gli host "vittima" dalla ricerca.

Infine il campo `json` descrive informazioni variabili in base alla tipologia dell'allarme, codificate in un oggetto json, e risulta per noi di particolare interesse per eseguire analisi più specifiche. Ad esempio, in caso di allarmi associati a HTTP può contenere la richiesta ed eventuale risposta.

Discuteremo nel dettaglio le informazioni contenute nel campo `json` per noi di interesse più avanti.

6.2 Struttura codice

Nell'appendice A.1 è indicato il repository dell'implementazione realizzata:

Forniamo una breve descrizione dei file presenti nel repository:

- `analyzer_daily.py` : sfrutta l'API Python di `ntopng` per richiedere gli allarmi con severità uguale o maggiore di 5 e gli allarmi di tipo *Unidirectional Traffic*, relativi alle 24 ore di osservazione. Per ciascuna finestra temporale richiede allarmi di livello superiore e rispettivi rating, e infine determina gli outlier e rappresenta i risultati su dei grafici come quelli nella figura 6.1.

- `myenv.py` Contiene le informazioni necessarie all'API di `ntopng`

```
mykey = "password"
myusr = "username"
myurl = "http://IPntopng:porta"
myiface_id = 1
WINDOW_SIZE_MINUTES = 60
```

- `analyzer/alertdb.py` : cuore del tool, vi sono definite le funzioni per gestire gli allarmi ed eseguire le analisi su di essi.

- `analyzer/utils` : In questa cartella vi sono due file: `u.py` contiene alcune funzioni "general purpose", mentre `c.py` contiene alcune costanti, in particolare valori soglia, posizionati in un file a parte al fine di renderne più semplice il *tuning*.

  A questi due può essere aggiunto un file `ctx.py`, che contiene costanti con le informazioni di contesto, da realizzare e compilare opportunamente in base, chiaramente, al contesto di utilizzo. Nel codice potrà comparire la costante booleana `CONTEXT_INFO`, che nella logica del programma indica se sono disponibili (e se utilizzare) queste informazioni.

- Altri due file sono stato utilizzati in fase di test e sviluppo.

  – `analyzer_run.py` : sfrutta l'API Python di `ntopng` per richiedere gli allarmi con severità uguale o maggiore di 5 e gli allarmi di tipo *Unidirectional Traffic*, con riferimento solo a una singola finestra temporale. Gestisce ogni allarme ricevuto e infine richiede gli allarmi di livello superiore.

  – `analyzer_stream.py` : Tentativo iniziale di realizzare una versione un'implementazione del modello che funzionasse in streaming, non portato a compimento a causa di tempo.

Di seguito discuteremo alcune sezioni del codice. Alcune istruzioni non strettamente legate al workflow del programma, il cosiddetto "syntactic sugar", saranno omesse al fine di rendere il codice più leggibile.

6.3 Gestione iniziale allarmi

Per ogni allarme ricevuto, si controlla se l'allarme va ignorato, altrimenti viene gestito chiamando gli opportuni handler. Gli allarmi di tipo *Low-goodput*, *Long-lived flow* e *Unidirectional Traffic* sono di severità bassa e richiedono una gestione diversa dagli altri. I primi due ci interessano per individuare i DGA maggiormente sospetti, mentre *Unidirectional Traffic* ci interessa per scovare le vittime di probing.

Ogni allarme viene quindi gestito dal seguente (pseudo)codice:

```
if to_be_ignored(a):
    return

if (low_goodput_handler(a) or
    long_lived_flow_handler(a)):
    return

if unidirectional_handler(a):
    return

bkt_srv = add_to_bucket(a,bkt_srv,(SRV_ID, a["vlan_id"], a["alert_id"]))
bkt_cli = add_to_bucket(a,bkt_cli,(CLI_ID, a["vlan_id"], a["alert_id"]))
bkt_srvcli = add_to_bucket(a,bkt_srvcli,(SRV_ID,CLI_ID, a["vlan_id"], a["
    alert_id"]))

is_relevant_singleton(a)
```

Dopo aver controllato se l'allarme richieda o meno una gestione specifica (in caso negativo) viene aggiunto a tre dizionari, uno per ciascun tipo di raggruppamento: *SRV, CLI, SRVCLI*. Ogni entry di ciascun dizionario si può vedere come un bucket, da cui il prefisso `bkt` nel nome.

Con strutture di questo tipo possiamo facilmente ottenere tutti gli allarmi associati a una certa chiave `K = (SRV, VLAN_ID, ALERT_ID)` e dunque il *flusso di allarmi* di chiave K, facendo semplicemente `bkt_srv[K]`.

Infine, si verifica se l'allarme è critico in base agli altri criteri discussi precedentemente, escludendo quelli relativi ai flussi di allarmi.

6.4 Flussi di allarmi

Per ciascun flusso viene eseguita la funzione `compute_bkt_stats(s: list, GRP_CRIT: int)`, suddivisa e discussa nelle seguenti sottosezioni. Ricordiamo che la lista `s` che appare nei parametri contiene gli allarmi appartenenti al flusso in questione. La funzione restituisce un dizionario `d` contenente statistiche e informazioni calcolate su `s`.

6.4.1 Entropia e paradigma CLI-SRV

```
# Convert IP to int first, then compute entropy
def ip_to_numeric(x):
    try:
        return struct.unpack("!I", socket.inet_aton(x))[0]
    except:
        return 0
srv_ip_toN = list(map(ip_to_numeric,map(lambda x: x["srv_ip"],s)))
cli_ip_toN = list(map(ip_to_numeric,map(lambda x: x["cli_ip"],s)))

d["srv_ip_S"] = u.shannon_entropy(srv_ip_toN)
d["cli_ip_S"] = u.shannon_entropy(cli_ip_toN)
d["srv_port_S"] = u.shannon_entropy(list(map(lambda x: x["srv_port"],s)))
d["cli_port_S"] = u.shannon_entropy(list(map(lambda x: x["cli_port"],s)))

d["srv_port_count"] = len(set(map(lambda x: x["srv_port"],s)))
d["cli_port_count"] = len(set(map(lambda x: x["cli_port"],s)))
```

Per prima cosa si convertono gli indirizzi IP da stringhe a interi, al fine di poter calcolare l'entropia su di essi, usata similmente a come viene fatto per le porte, dopodiché viene calcolata l'entropia con `shannon_entropy()`, la cui implementazione si trova nell'appendice A.2.1.

Queste statistiche vengono usate per determinare host che si comportano in modo opposto a quanto descritto dal paradigma *client-server*, come spiegato nella sezione 4.1.1.1. Il codice per far ciò è semplice ed è riportato nell'appendice A.4.

6.4.2 Periodicità

```
# PERIODICITY
# assert that 's' is sorted on tstamp
def avg_delta(l):
    # init sum = 0
    delta_sum = dt.timedelta(seconds=0)
    for i in range(1,len(l)):
        # add tdiff to sum
        delta_sum += l[i] - l[i-1]
```

```
9     # return average tdiff
10    return delta_sum / (len(l) - 1)
11
12 # isolate tstamp series from alert flow
13 tstamp_list = list(map(lambda x: x["tstamp"],s))
14 tdiff_avg_unrounded = avg_delta(tstamp_list)
15 # round to seconds
16 d["tdiff_avg"] = dt.timedelta(seconds=round(tdiff_avg_unrounded.
       total_seconds()))
17 # If the avg period is close to 0...
18 # cannot divide by 0
19 if d["tdiff_avg"].total_seconds() == 0:
20    # 1.0 #... consider '1' as reference to compute CV
21    tdiff_avg_unrounded = dt.timedelta(seconds=1)
22 # Compute CV as stddev/avg
23 d["tdiff_CV"] = np.std(list(map(lambda x: (x - dt.datetime(1970, 1, 1)).
       total_seconds(),tstamp_list))) / tdiff_avg_unrounded.total_seconds()
24 d["tdiff_avg"] = str(d["tdiff_avg"])
```

La funzione `avg_delta()` calcola il periodo del flusso di allarmi, ovvero il `tdiff` medio, la differenza media fra i tempi di arrivo degli allarmi.

Tenendo presente che per calcolare il coefficiente di variazione, (o deviazione standard relativa) su `tdiff` è necessario dividere per il valore medio appena calcolato, dobbiamo gestire il caso particolare in cui quest'ultimo sia 0, per evitare di eseguire una divisione illecita. In tal caso, dividiamo per 1, ottenendo dunque la deviazione standard assoluta, e non relativa.

A questo punto abbiamo sia l'eventuale periodo, sia il CV, che ci permetterà di discernere flussi effettivamente periodici e non.

6.4.3 Binary Application Transfer

Si noti che la funzione `u.get_BAT_pathserver()` restituisce una tupla composta come segue: `(file_path:str,server_name:str)`, utilizzando stringhe vuote `""` in caso di errore, o `server_name` mancante.

```
1 # BAT Binary Application Transfer -> Check if same file
```

```
# get the first path
first_path = (u.get_BAT_path_server(s[0]["json"]))[0]
paths = map(lambda x: u.get_BAT_path_server(x["json"])[0],s)
if first_path != "":
    # Check if the file transferred is always the same
    for p in paths:
        if p != first_path:
            # If not, exit
            first_path = ""
            break
d["bat_same_file"] = first_path
```

Il senso di questo semplice snippet codice è semplicemente controllare se in un flusso di allarmi *BAT*, il percorso del file trasferito è sempre lo stesso; la stringa vuota è usata come valore speciale, in caso negativo.

6.5 Singoli Allarmi

Dopo essere stato aggiunto agli opportuni bucket, utilizzati per lavorare su flussi di allarmi come descritto poco fa, si determina se il singolo allarme rientra in uno dei criteri di criticità discussi nei paragrafi (4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6), o quantomeno, se è necessario manipolarlo per svolgere inferenze su tali criteri in un secondo momento.

Ciò viene svolto nella funzione `is_relevant_singleton()`, che in base al campo `alert_name` dell'allarme ricevuto esegue le opportune operazioni. Qualora l'allarme risultasse rilevante, la funzione restituisce la corrispondente chiave K insieme ad eventuali informazioni aggiuntive (come un sospetto dominio DGA), altrimenti `None`

Gli snippet di codice che seguono sono estratti della funzione sopra menzionata, ciascuno riguardante un tipo diverso di allarme.

6.5.1 Binary Application Transfer

Ricordiamo che per allarmi di questo tipo ci interessa individuare trasferimenti aperiodici di file, dunque evincere i percorsi di file che vediamo transitare una e una sola volta nelle finestra temporale di osservazione e in quelle precedenti. A tal proposito, il `set bat_paths` è una variabile persistente che mantiene i path dei file osservati sulla rete.

```
if (alert_name == "binary_application_transfer"
    and not (CONTEXT_INFO and whitelisted(path))
    and not (CONTEXT_INFO and whitelisted(srvname))):
    # if not learning and previously unseen path
    if (LEARNING_PHASE == False and path not in bat_paths):
        # add path and srvcli to singleton alerts
        sav[alert_name][path] = SRVCLI_ID
        # add to known paths
        bat_paths.add(path)
        return SRVCLI_ID
    # if not learning and path already seen
    elif (LEARNING_PHASE == False and path in bat_paths):
        # remove entry from singleton alerts, if present
        sav[alert_name].pop(path,None)
    # add to known paths regardless
    bat_paths.add(path)
```

Innanzitutto si controlla che né il percorso né il nome del server appartengano a una whitelist (che rientra nelle informazioni di contesto), e che il programma non stia attraversando l'eventuale fase di apprendimento (`LEARNING_PHASE`), in cui è di interesse solo memorizzare i percorsi dei file trasferiti.

Dopo aver fatto ciò, si verifica se il percorso del file fosse già stato visto precedentemente:

- **No** ⇒ si aggiunge a un dizionario (`sav["binary_application_transfer"]`) tale percorso, inserendo come valore la chiave K dell'allarme (`SRVCLI_ID`).
- **Sì** ⇒ si elimina dal dizionario l'entry corrispondente a tale percorso. Potrebbe anche non esistere, ovvero essere stata eliminata precedentemente, ma non è

un caso che ci interessa distinguere.

6.5.2 Suspicious DGA Domain

```
if (alert_name == "ndpi_suspicious_dga_domain"
    and (domain_name := get_domain_name())
    and (not CONTEXT_INFO or not any(x in domain_name.split(".") for x in
    ctx.WHITELIST_DOMAIN_TOKEN))):
    # We need both requestor and server in this case
    key = SRVCLI_ID
    # Find the most similar domain if existent and remove the unmatching
    portion
    # This results in the portion of the name used, e.g.
    #       '1564903955.dgadom.com'
    #       '6759204650.dgadom.com'
    #    Will collapse under 'dgadom.com'
    partial_name = u.add_to_domain_dict(dga_suspicious_domains,domain_name,
    key)
    dga_suspicious_domains[partial_name][key] += 1
```

Viene popolato un dizionario, al fine di offrire una vista simile a quella mostrata di seguito, il cui scopo è evidenziare i nomi sospetti e i corrispondenti host che li hanno generati.

Il fatto che più host abbiano generato domini sospetti DGA simili può essere indice o di normale traffico della rete causato da uno stesso applicativo o di un'infezione replicata su più macchine.

```
{"dgadom.com": {
  "('host1', 'host2', 32)": 5,
  "('host2', 'host3', 4)": 14,
},
"ccdm.link.uk": {
  "('host2', 'hostname5', 32)": 2
}}
```

Esaminando la funzione `add_to_domain_dict(d,name)` capiamo come viene popolato il

dizionario, passato a essa come parametro insieme al nome da aggiungervi.

`find_longest_common_subsequent_substrs()` per ogni chiave del dizionario (domini) determina la porzione di nome a dominio in comune, e restituisce la più lunga che trova. Ne riportiamo l'implementazione nell'appendice A.4.

.

```
def add_to_domain_dict(d:dict,name:str,key):
    best_match = [name]
    best_match_name = name
    name_tokens = name.split(".")
    for dga_name in (d.keys()):
        tmp = find_longest_common_subsequent_substrs(name_tokens,dga_name.split("."))
        best_match = tmp if (len(tmp) > len(best_match)) else best_match
        best_match_name = dga_name if (len(tmp) > len(best_match)) else best_match_name

    new_name = ".".join(best_match)
    d[new_name] = d.pop(best_match_name,{})
    d[new_name][key] = 0
    return new_nam
```

La gestione del singolo allarme *Suspicious DGA Domain* termina così, ma ricordiamo che eseguire dei semplici raggruppamenti, oltre ad essere poco interessante, non basta, vorremmo filtrare le chiavi K del dizionario così popolato.

Come menzionato in 4.1.2 vogliamo considerare solo le chiavi K il cui client è associato ad allarmi di tipo *Long-lived Flow* e *Low-Goodput*, e questa è una verifica che facciamo in un secondo momento iterando sul dizionario già popolato, svolta dalla funzione `get_dga_sus_domains()`, di seguito riportata e discussa.

```
# Each value is a tuple (srv,cli,vlan)
# But we want to detect only the clients' behavior,
# to check whether they are associated with
# longlived flows or lowgoodput ratio alerts,
# Which might mean they are actual DGAs

# Create a list using only keys, and get only (cli,vlan) discarding srv
```

```
8  toret = {k: list(set(map(lambda x: x[1:],v.keys())))
9          for k, v in dga_suspicious_domains.items()}
10 # For each client (potential attacker) perform a request
11 # to get its associated longlived or lowgoodput alerts
12 for k, v in toret.items():
13     req_str_hosts = u.request_builder_srvcli(v)
14     # 11 -> Longlived flows | 12 -> Lowgoodput
15     # Request first only Longlived flows
16     req_str = "(alert_id=11) AND (" + req_str_hosts + ")"
17     # 5 hit per key are sufficient for our purposes
18     # Note that if there are, for example, four keys, thus maxhits = 4*5 =
       20
19     # We might get 20 hits related to the same key.
20     # Not a big deal.
21     new_alerts = u.make_request(req_str,5 * len(v))
22
23     # Now request lowgoodput flows
24     req_str = "(alert_id=12) AND (" + req_str_hosts + ")"
25     new_alerts += u.make_request(req_str, 5 * len(v))
26     for a in new_alerts:
27         # since 'alert_id = 11 OR alert_id = 12'
28         # each alert will be handled properly and put in longlived or
       lowgoodput
29         new_alert(a)
30
31 return {k: x for k, v in dga_suspicious_domains.items()
32         # We want to consider only DGAs in which the attacker is associated
        with long-lived low-goodput flows
33         if len((x := list(filter(lambda x: (x[1],x[2]) in longlived and (x
       [1],x[2]) in lowgoodput, v.keys()))))) >= 1}
```

Notiamo che è necessario eseguire due richieste RESTful (quasi) per ogni client. Far ciò introduce ritardo dovuto all'overhead richiesto, ma ha un impatto molto meno significativo rispetto al richiedere *tutti* gli allarmi *Long-lived Flow* e *Low-Goodput*, tenerli in memoria, e poi verificare se fra essi ve ne sono alcuni associati agli host a noi di interesse. Questo perché tali allarmi sono di severità bassa e piuttosto comuni, dunque fin troppo importanti in termini di quantità.

Per ridurre la quantità di richieste, invece di mandarne una per ogni singolo client, come sarebbe "opportuno" fare, ne mandiamo una relativa a tutti i client

riguardanti uno stesso dominio sospetto DGA.
Ricordando la struttura del dizionario costruito, per ogni dominio (chiave del dizionario), a partire dalla corrispondente serie di chiavi K *SRVCLI* estraiamo la serie di chiavi *CLI*, eliminando dunque il campo *SRV* dalla tupla K. Otteniamo quindi una lista di client per ogni dominio, come mostrato a riga 8.
Dunque si formula una richiesta all'API REST ponendo nella clausola `WHERE` la disgiunzione fra i client in una lista, operazione realizzata da `u.request_builder_srvcli(v: list)`, e successivamente inviata, aggiungendo come prefisso la congiunzione con`alert_id = 11` e `alert_id = 12` la seconda volta.'
Ciò che può accadere è che tutti gli allarmi restituiti siano riferiti a un singolo client nella lista, o comunque non a tutti, ma in fondo non costituisce un problema: essendo tutti i client in una tale lista accomunati dall'aver generato lo stesso (o simile) dominio sospetto DGA, se anche uno solo si comportasse in modo maggiormente sospetto, allora potremmo aver ragione di voler esaminare anche gli altri.

Un'altra osservazione da fare è che fare un'unica richiesta del tipo `(alert_id=11 OR alert_id=12) AND (...)` non va bene, perché gli allarmi restituiti potrebbero essere *solo* di tipo 11 o *solo* di tipo 12, mentre a noi servono sia gli uni, sia gli altri.

6.5.3 TLS Certificate Self-signed

```
if (alert_name == "tls_certificate_selfsigned"
        and (ja3_hash := get_ja3_hash())
        and not (ip_address(a["srv_ip"]).is_private)):
        key = SRV_ID
        u.add_to_dict_dict_counter(snd_grp[alert_name],ja3_hash,key)
        # add domains to key tuple
        return key + (ja3_hash,)
```

Possiamo raggruppare questi allarmi in base alla firma hash *JA3* di client e server, che potrebbe rimanere uguale anche qualora gli host cambiassero.

Vengono considerati critici solo gli allarmi in cui il server, quello che è obbligato ad esibire il certificato TLS, è un host pubblico e non privato.
Tale controllo è svolto a riga 3. Il resto delle operazioni sono eseguite in `u.add_to_dict_dict_counter()`, riportata nell'appendice A.4.

6.5.4 Probing

Quando si riceve un allarme *Unidirectional Flow*, lo si utilizza per popolare il dizionario `unidir` in cui per ogni entry la chiave è un server e il valore i client e la porta su cui lo hanno contattato unidirezionalmente.

```
1  def get_unidir_probed():
2      probed = {}
3      for srv,t in unidir.items():
4          # If a server is a victim of probing, one or more client will be
       trying to
5          # unidirectionally communicate with it on many different ports
6          if (len(unidir[srv]) > MIN_PROBING_RELEVANT_SIZE and
7              is_server(srv[1]) and
8              (s := u.shannon_entropy(list(map(lambda x: x[1],unidir[srv]))))
        > PROBING_ENTROPY_THRESH):
9              probed[srv] = set(map(lambda x: x[0],unidir[srv]))
10     return probed
```

6.5.5 Remote Access

```
1  if (alert_name == "remote_to_local_insecure_proto"
2          and a_json["ndpi_category_name"] == "RemoteAccess"
3          u.addremove_to_singleton(remote_access,(SRV_ID,"srv"),1)
4          u.addremove_to_singleton(remote_access,(CLI_ID,"cli"),1)
```

Come nell'implementazione per i trasferimenti di file binari 6.5.1, vi è una variabile persistente, in questo caso il dizionario `remote_access`, in cui si tiene traccia di quali

host hanno avuto stabilito una connessione di accesso remoto nelle precedenti finestre temporali. La funzione `u.addremove_to_singleton()` è riportata nell'appendice A.4.

6.5.6 Blacklisted Peers

Nello snippet che segue, `blk_peers` è un dizionario che tiene traccia dei peer che comunicano con host blacklistati e segna il ruolo "SRV" o "CLI" assunto nella comunicazione. Il codice è piuttosto autoesplicativo.

```
if (alert_name == "blacklisted"):
    if (((CONTEXT_INFO and is_watched(SRV_ID)) or
        u.is_private(SRV_ID[0])) and
        a["srv2cli_bytes"] > 0):
        u.add_to_blk_peers(blk_peers,SRV_ID,"SRV",CLI_ID)
    if (((CONTEXT_INFO and is_watched(CLI_ID)) or
        u.is_private(CLI_ID[0])) and
        a["cli2srv_bytes"] > 0):
            u.add_to_blk_peers(blk_peers,CLI_ID,"CLI",SRV_ID)
            return CLI_ID
```

6.5.7 Host che generano allarmi eterogenei

L'implementazione, semplice ma verbosa, di quanto descritto nella sottosezione 4.1.8, fa affidamento alla collection `Counter` e alla dictionary comprehension per filtrare i risultati rispetto a delle soglie minime e rispetto a quanto previsto dal criterio (riga 11).

```
1 def get_higher_alert_types(GRP_CRIT: int):
2 # Firstly get groups which contain more than 1 alert and where key is not
    victim
3 #   Note: x should never be None, but it's safer to check regardless
4 relevant_keys = filter(lambda x: x and len(x) >= 1 and is_relevant(x),bkt.
    keys())
5 # Count the number of alert_types for each group ("ip","vlan_id") or ("srv
    ","cli","vlan_id")
6 nat = Counter(map(lambda x: ((x[0],x[1]) if len(x) == 3 else (x[0],x[1],x
    [2])), relevant_keys))
```

```
# Consider only groups which generated at least 2 different alert types
n_alert_types_per_key={x: count for x, count in nat.items() if count >= 2}
n_alert_types_mean=math.ceil(np.mean(list(n_alert_types_per_key.values())))
# return only keys s.t. n_alerts > mean
return {x: count for x, count in n_alert_types_per_key.items() if count >
    n_alert_types_mean}
```

6.6 Output allarmi

Discutiamo brevemente del formato in cui vengono esportati gli allarmi di livello superiore, anche per vedere nei fatti del tipo di dati di cui stiamo trattando.

Gli allarmi sono organizzati in un dizionario, di cui riportiamo un esempio sotto serializzato in JSON.

```
{
"TIME": {
  "start": "2023/17/04 12:21:36",
  "end": "2023/17/04 13:21:36"
},
"FLAT_GROUPINGS": {
  "SRV": {
    "higher_alert_types": {
        "('218.27.107.188', 30)": 5,
        "('96.95.117.166', 34)": 4,
        "('7.135.172.204', 16)": 4},
    "cs_paradigm_odd": {
        "('47.123.18.180', 10)": "odd_server",
        "('119.209.42.28', 28)": "odd_server"
    },
    "simultaneous": {
      "('165.205.230.226', 57, 42)": "0:00:00
    ndpi_http_suspicious_user_agent",
      "('189.4.125.168', 26, 42)": "0:00:00 ndpi_http_suspicious_user_agent
    "
    },
    "periodic": {},
    "similar_periodicity": {
```

```
22          "0:05:24": {
23              "data_exfiltration": [
24                ["214.137.121.184",34],["56.165.226.82",51]
25              ]
26            }
27      },
28      "bat_samefile": {}
29    },
30    "CLI": {
31      "higher_alert_types": {
32        "('160.191.208.2', 38)": 4,
33        "('97.227.249.72', 52)": 5
34      },
35      "cs_paradigm_odd": {},
36      "simultaneous": {},
37      "periodic": {
38        "('31.136.127.209', 23, 29)": "0:05:36 binary_application_transfer",
39        "('176.131.241.21', 10, 29)": "0:00:36 binary_application_transfer"
40      },
41      "similar_periodicity": {},
42      "bat_samefile": {
43        "('10.160.87.161', 65, 29)": "gFRnx-r0467.teamviewer.com/din.aspxast2
    5223client=Geme3"
44      }
45    },
46    "SRVCLI": {
47      "higher_alert_types": {},
48      "cs_paradigm_odd": {},
49      "simultaneous": {
50        "('94.79.24.204', '188.157.164.249', 63, 42)": "0:00:00
    ndpi_http_suspicious_user_agent"
51      },
52      "periodic": {},
53      "similar_periodicity": {},
54      "bat_samefile": {}
55    }
56  },
57  "BAT_ONE_TIME": {
58    "/xscan/download/srvconfig.ini":
59      "('112.158.48.118', '246.128.47.66', 22)",
60    "wirg4=01.plg.intra/DWNIROWQO222_%8492":
61      "('133.12.34.160', '163.58.191.15', 9)",
62    "microsoft.com/server/manifest.%543543JJF-FH.cab":
```

```
63      "('40.77.52.218', '180.42.154.116', 35)",
64    "kontiki.com/nofgo/584300203":
65      "('100.202.92.20', '115.62.178.187', 43)"
66  },
67  "DGA_DOMAINS": {
68    "dga.punyc0d3.64352FGWD65336GHF4442.co.tv" : [
69      "('32.43.142.56', '205.42.67.112', 57)",
70      "('205.67.33.22', '10.0.0.2', 58)"
71    ]
72  },
73  "PROBING_VICTIMS": {
74    "('252.178.140.35', 65)": [
75    "41.79.65.182","102.193.18.47","233.17.140.53","154.13.113.223","105.156.
      234.158","142.216.175.63"],
76    "('218.36.205.92', 7)": [
77      "172.216.113.17","73.191.29.15","112.10.215.58","211.10.80.144"]
78  },
79  "TLS_SELFSIGNERS_PEERS": {
80    "('253.130.31.82', 47)": 13,
81    "('131.195.0.60', 42)": 31
82  },
83  "BLK_PEER": {
84    "(('149.73.61.222', 27), 'SRV')": [
85      ["70.230.34.230",25],["36.0.151.30",66]
86    ]
87  },
88  "REMOTE_ACCESS": {
89    "(('117.197.53.146', 39), 'cli')": 1,
90    "(('10.182.252.112', 33), 'srv')": 1,
91    "(('177.251.173.255', 45), 'cli')": 1,
92    "(('177.131.20.17', 30), 'cli')": 1
93  }
94  }
```


In `"FLAT_GROUPINGS"` troviamo gli allarmi di livello superiore relativi ai flussi di alert, suddivisi sulle tre chiavi *SRV,CLI,SRVCLI* della formula 4.1.

Per ognuna delle tre chiavi vi sono gli host che soddisfano i criteri descritti nella sezione 4.1.1; a queste seguono i criteri svincolati dai flussi descritti nelle sezioni successive 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6. In ordine, associamo ad ogni *feature* (chiave

dei dizionari) la descrizione del corrispondente criterio:

1. `"higher_alert_types`:Host che generano allarmi eterogenei 4.1.8
2. `"cs_paradigm_odd"` : Paradigma client-server 4.1.1.1
3. `"simultaneous"` : Simultanei 4.1.1.3
4. `"periodic"` : Periodicità 4.1.1.3
5. `"similar_periodicity"` : Periodicità simile 4.1.1.3
6. `"bat_samefile"` : Binary Application Transfer 4.1.1.2
7. `"BAT_ONE_TIME"` : Binary Application Transfer 4.1.5
8. `"DGA_DOMAINS"` : Domain Generation Algorithm 4.1.2
9. `"PROBING_VICTIMS"` : Probing 4.1.4
10. `"TLS_SELFSIGNERS_PEERS"` : TLS Self-signed Certificate 4.1.3
11. `"BLK_PEER"` : Blacklisted host peers 4.1.7
12. `"REMOTE_ACCESS"` : Accesso Remoto 4.1.6

6.7 Rating e Outlier

6.7.1 Rating

Il rating (Capitolo 5) di ciascun host in una singola finestra temporale viene calcolato nella funzione `get_host_ratings()` [1] la quale prende in input un dizionario (A.2.4) rappresentante gli allarmi di livello superiore, le cui chiavi sono le sopracitate feature. Tale dizionario è illustrato nella precedente sezione 6.6.

Ad ogni allarme di livello superiore corrisponderà un *rate*, dipendente dal tipo e dalle proprietà dell'allarme in questione. Abbiamo fornito un esempio intuitivo di come venga calcolato il rate per un singolo allarme nella sezione 5.1.1.

Il dizionario `hostsR`, ritornato dalla funzione, viene popolato usando come chiavi gli host, e come valori dei dizionari composti da coppie `"feature" : rate`. In questo modo per ogni host teniamo traccia di quanto ciascuna feature ha contribuito al

[1] presente nella repo su GitHub e non riportata in quanto molto verbosa

rating complessivo, essendo questo pari alla somma del rate associato a ciascuna di esse.

La funzione esamina le feature una per volta, calcolando per ogni host associato a ciascuna di esse il corrispondente rate, partendo da quelle non riguardanti raggruppamenti di allarmi, indicati con lettere maiuscole, e poi esaminando quelle basate su flussi di allarmi. Considerando la serie temporale costituita dalla quantità di allarmi esportati da un host in una finestra temporale (*intensità allarmi*), se l'ultimo valore risulta essere un outlier, ovvero se nell'ultima finestra l'host ha generato molti più o molti meno allarmi rispetto a quanto aveva fatto prima, si aggiunge un rate che vale al massimo 60, sotto la feature `"N_alerts"`. Infine viene aggiunto un piccolo rate su base logaritmica sulla quantità di allarmi generati dall'host.

6.7.2 Rilevamento Outlier

Nel file `analyzer/utils/u.py` sono presenti, fra le altre utilities, le implementazioni dei metodi di rilevamento degli outlier menzionati nel capitolo 5.2.

A titolo esemplificativo, di seguito è riportato il codice relativo al *Single Exponential Smoothing* (SES)[33]. Le prime righe servono per gestire serie con valori iniziali nulli, mentre il parametro `lower_bound` per scartare valori eccessivamente bassi.
È importante eliminare valori iniziali pari 0 perché indicano che l'host in questione non aveva generato *allarmi di livello superiore* nelle prime n finestre temporali: considerandoli si rischierebbe che il primo valore non nullo della serie venga riportato come outlier, mentre nei fatti la serie temporale di quell'host inizia con tale *n-esimo* valore non nullo. Ne è un esempio la serie mostrata nella sezione successiva, in figura 6.2.
Ci interessa scartare valori eccessivamente bassi perché dai test svolti durante lo sviluppo è emerso che questi portavano a falsi positivi, o a ridondanza di informazioni: infatti, rating nulli (o molto bassi) indicano che i criteri di criticità non sono stati soddisfatti, dunque rappresentano situazioni, anche se non certamente esenti da

anormalità, di minore interesse rispetto ad altre con rating elevati. Inoltre in serie in cui si poteva osservare un valore anomalo più alto degli altri, veniva riportato come outlier sia il picco, sia il valore basso o nullo immediatamente successivo, duplicando inutilmente l'informazione.

```
def detect_outliers_exp_smooth(values, lower_bound : int, alpha=0.8,
    threshold=3.0):
    data = list(values)
    # Exclude leading zeros
    leading_zeros = 0
    while len(data) > 0 and data[0] == 0:
        data.pop(0)
        leading_zeros += 1
    if len(data) == 0:
        return []

    data = np.array(data)
    # Apply exponential smoothing
    smoothed = [data[0]]
    for i in range(1, len(data)):
        smoothed.append(alpha * data[i] + (1 - alpha) * smoothed[-1])
    # Calculate deviations from smoothed values
    deviation = np.abs(data - smoothed)

    # Calculate the standard deviation of the deviation
    std_dev = np.std(deviation)
    # Check that the standard deviation is not zero
    if std_dev == 0:
        return []
    # Calculate the threshold for outlier detection
    threshold = threshold * std_dev

    # Find the outliers
    outliers = np.where(deviation > threshold)[0]
    outliers = list(filter(lambda x: values[x] >= lower_bound,map(lambda x:
     x + leading_zeros,outliers)))
    return outliers
```

6.7.3 Rappresentazione Outlier

La funzione `plot_outliers` (non riportata, disponibile su GitHub) prende in input un dizionario di host, i cui elementi hanno la seguente forma:

```
    ('ip_or_hostname', vlan_id): [
    [7, 8],
    [0, 0, 0, 0, 0, 0, 85.2, 0, 0, 34.2, 36]
   ]
```

dove la prima lista sono gli indici delle finestre temporali in cui sono stati trovati valori outlier (5.2), mentre la seconda sono i rating corrispondenti a ciascuna finestra per quell'host.
Gli altri parametri servono per determinare quali sono le feature, quanto hanno contribuito per ciascun rating e per gestire chiamate ricorsive, per stampare lo storico dei rating di un host, o eventualmente tutti i rating relativi a una certa finestra.

La funzione utilizza la libreria python `matplotlib`[11] per realizzare dei grafici *Stacked Bar Chart* come quello che segue in figura 6.1. Vengono svolte diverse manipolazioni sulle variabili necessarie per fornire un input corretto all'API della libreria; queste sono strettamente dipendenti dall'implementazione e dalla rappresentazione dei dati nelle variabili python, dunque sorvoleremo su di esse.

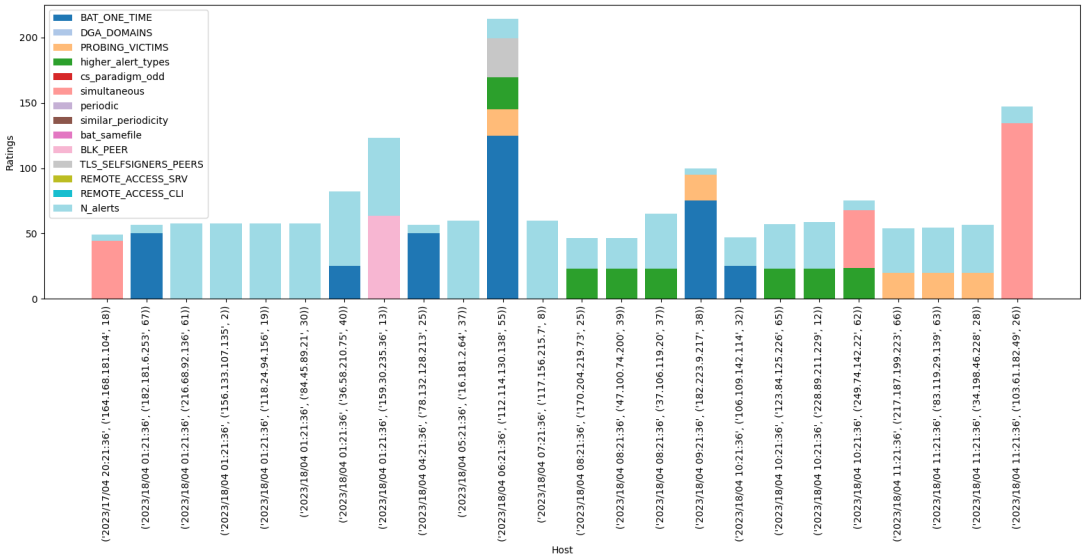


Figura 6.1: Output Plot Sample

Ogni elemento sull'asse X è un host, e la finestra temporale in cui è stato rilevato un outlier nei suoi rating. A ogni feature, elencate a destra, è associato un colore. Sull'asse Y troviamo i rating e quali feature vi hanno contribuito. In questo modo rappresentiamo in una vista leggibile, quali host, quando, e che allarmi hanno generato.

Passando col mouse sopra una delle barre impilate viene indicato il nome della feature corrispondente e la sua altezza.
Invece, cliccandovi vengono mostrati due grafici come quelli sottostanti (Fig. 6.2), uno che illustra la serie temporale dei rating dell'host, e l'altro la quantità di allarmi per ogni finestra.

Figura 6.2: Ratings/Sizes Output sample

Capitolo 7

Validazione

Avendo definito un modello e svolto il tuning dei parametri, dobbiamo provare ad applicarlo a una rete per vedere che informazioni restituisce sulla rete e se queste sono confermate dai dati empirici, che nel nostro caso sono singoli allarmi e flussi.

Sono stati presi ad esempio alcuni host e relativi rating segnalati come anomali e ne è stato indagato il traffico, controllando manualmente flussi e allarmi attraverso l'interfaccia browser di ntopng. Di tali controlli sono stati riportati i tratti salienti.

Per motivi di tempo, vista la necessità di controllare manualmente i dati empirici, non è stato possibile eseguire una validazione su larga scala, dunque si è preferito eseguire un'analisi ristretta per fornire dei risultati preliminari, ponendo come obiettivo della discussione che segue mostrare come il modello funziona, ed evidenziare delle situazioni in cui il modello risponde correttamente e altre in cui invece segnala dei falsi positivi.
In ogni caso, in chiusura al capitolo viene mostrata una tabella che riassume i risultati delle anomalie discusse secondo questa classificazione:

C.1 Un **problema**: l'host si è comportato in modo diverso rispetto al suo "standard", rivelando una minaccia o un comportamento indesiderato

C.2 Un **cambio di comportamento**: l'host si è comportato in modo diverso rispetto al suo "standard", non necessariamente in modo malevolo o indesiderato.

C.3 Un **falso positivo**: non c'è stato alcun cambio di comportamento, il rating e l'anomalia sono stati causati dal *rumore* degli allarmi.

7.1 Triage e Ambiente test

Ricordiamo che studio, analisi e test sono stati svolti in una rete di un'azienda della grande distribuzione (GDO). In reti complesse è frequente che vi sia del *rumore*, causato da comportamenti che appaiono come anomali ai software di sicurezza, ma che, nel contesto in cui avvengono, sono standard. Dunque prima di poter inferire proprietà degli allarmi e/o aggregarli può essere utile una fase preliminare di *triage*, per ridurre la quantità di eventi da analizzare e, sperabilmente, rendere più efficace tale processo.

Dunque, se possibile chiederle all'amministratore di rete, possono essere utili informazioni di contesto, ad esempio se esiste una duplicazione di parte del traffico fra determinate VLAN, o se alcune sono dedicate esclusivamente a macchine server. Se una VLAN (o magari un range di indirizzi) è popolata da soli server, possiamo dare maggiore risalto ad allarmi che un server non dovrebbe generare, o ad un utilizzo di porte eccesivamente diversificate. Così come se il traffico in una certa VLAN deve riguardare esclusivamente host locali, allora allarmi associati a collegamenti remoti dovranno assumere maggiore rilevanza.

Se appropriate, possono essere utili informazioni più specifiche riguardanti lo specifico ruolo di determinati host, se ne hanno uno. Sapere che un host è un server

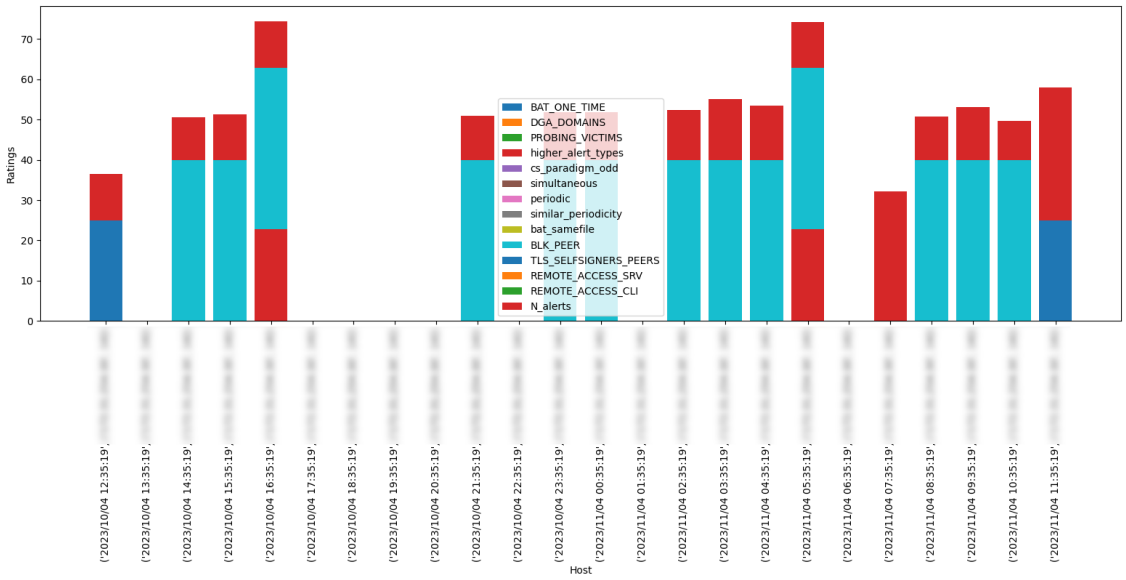
di posta, un proxy o un resolver DNS, può essere indicativo del tipo di traffico che lo attraversa e di quali siano i protocolli e gli applicativi di norma utilizzati, e dunque di quali tipi di allarmi saranno più o meno per esso rilevanti.

In seguito ai numerosi test e osservazioni fatte sulla rete in esame sono state utilizzate alcune whitelist e informazioni di contesto per filtrare parte del rumore.

Per motivi di riservatezza, nelle sezioni che seguono, gli indirizzi IP sono stati sostituiti da indirizzi casuali e alcuni URL sono stati alterati o oscurati.

7.2 Anomalia 1

Applicando il modello alla rete su 24 ore a partire dalle 12:35 del 10 aprile, usando finestre temporali di 60 minuti, esso ha riportato un'anomalia per l'host 172.3.43.75@41 nella finestra temporale che inizia alle 16:35, per intendersi, il quinto elemento sull'asse X nello stacked bar plot di seguito riportato, rappresentante lo storico dei rating di tale host.



Il rating da una finestra temporale alla successiva si è innalzato sufficientemente da essere riconosciuto come outlier.

Tale grafico ci indica che rispetto alla finestra temporale antecedente, continua ad essere un peer in flussi *Blacklisted* non unidirezionali, ma in più, ha generato tipi di allarmi diversi, di più rispetto agli altri host nella stessa finestra temporale; ciò ci viene detto dalla barra rossa in basso, che corrisponde ad `higher_alert_types`.

Questo è quanto ci è stato detto dal modello. A questo punto cerchiamo di verificare se ciò è confermato dai dati empirici.

Analisi

Esaminando attraverso la GUI di ntopng gli allarmi relativi alle finestre temporali dalle 15:35 alle 16:35 e dalle 16:35 alle 17:35, possiamo fare alcune osservazioni.

Nella seconda finestra i tipi di allarmi generati sono di più, in particolare vediamo che le tipologie prima assenti sono: *Binary Application Transfer, Remote to Local Insecure Protocol, Obsolete SSH Client or Cipher*

Partendo dagli allarmi Remote to Local Insecure Protocol, notiamo degli allarmi *Clear Text Credentials* (segnati anche come *Remote to Local Insecure Protocol*) con l'host 151.80.75.187, di OVH SAS, azienda di web hosting francese; tali allarmi c'erano anche prima, inoltre dall'analisi dei flussi con tale host, vediamo che in tutto l'arco della giornata vi sono frequenti piccoli download di pochi KB attraverso brevi sessioni FTP.
Questi allarmi fanno parte del rumore, del comportamento "standard" del server 172.3.43.75@41.

Vi è un allarme con l'host 216.218.206.82, Hurricane Electric, e si osserva che esso è associato all'unico flusso con tale host in tutta la giornata.
Dunque, innanzitutto 172.3.43.75@41 ha mostrato un cambio di comportamento:

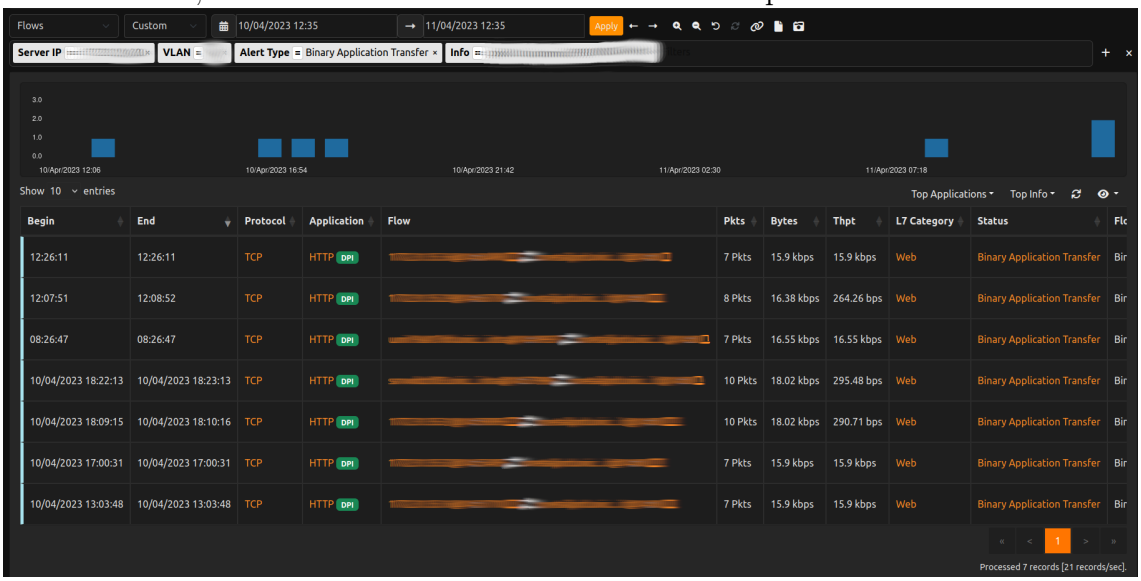
ha comunicato, usando un protocollo insicuro da locale a remoto, con un host con cui non aveva mai comunicato prima.
Digitando l'IP 216.218.206.82 nella barra degli indirizzi del browser si visualizza una pagina che ci informa che l'host è una sonda di un servizio volto a trovare server che hanno servizi ed eventuali vulnerabilità esposte per riportarle ai proprietari.
Il server ha rifiutato la connessione, dunque il "danno" si è limitato a del probing. Al di là del fatto che non si sia trattato di un prober malintenzionato, abbiamo comunque individuato una potenziale minaccia.

Per quanto riguarda il flusso TCP con il client russo 176.113.115.104 blacklistato, il server ha risposto, ma rifiutando la connessione; inoltre vediamo che si tratta di un tentativo unico nel corso della giornata, c'è solo quel flusso con il server da noi esaminato.
Essendo il flusso blacklistato, considerare questo caso un *problema* può essere eccessivo, l'amministratore di rete potrebbe considerarlo un evento di poco conto, ma al di là dell'interpretazione, rimane un dato indicante, se non altro, un comportamento anomalo per il server in questione. Mentre osservando i flussi del client russo vediamo quella che sembra essere attività di probing verso vari server, che, fortunatamente, non rispondono o rifiutano la connessione.

L'allarme di tipo *Obsolete SSH Client Version or Cipher*, corrisponde all'unico flusso SSH in tutta la giornata per il server in questione. Non si direbbe essere un problema, essendo il client un host locale, ma indubbiamente si tratta di un evento inaspettato per questo server, su cui l'amministratore di rete farà le opportune valutazioni.

Infine vi è il trasferimento del file binario `kiu1.busy.it/KIU/UNAsub.crl`. Quello in questione sembra essere l'unico flusso con il client 172.31.35.32.
Come vediamo nello screenshot sottostante e nei flussi di ntopng, tale file è stato

servito da 172.3.43.75@41 ad altri 7 client distinti nel corso della giornata, su base irregolare. Non sembrano essere una minaccia, ma piuttosto richieste che, per quanto asincrone, sono al limite di essere considerate "inaspettate".



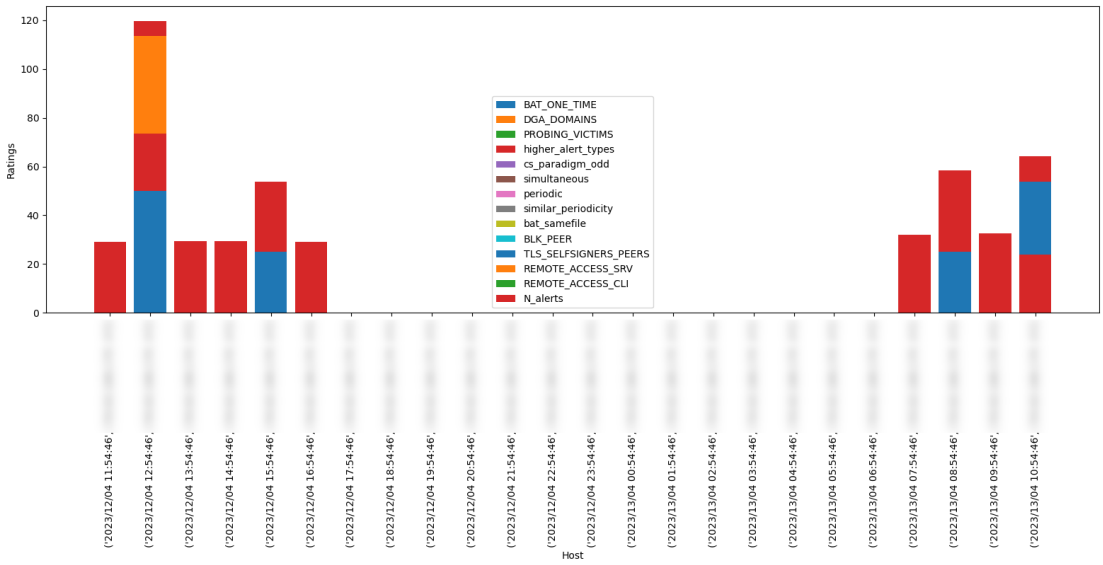
Osservazioni

Il modello ha evidenziato un'anomalia nel rating di un determinato host (172.3.43.75@41) in una certa finestra temporale ($16:35 \rightarrow 17:35$), fornendo una traccia di quale fosse il problema.

Ricordando le classificazioni C.1,C.2,C.3, dopo aver esaminato gli allarmi e alcuni flussi legati a tale host, abbiamo trovato dei **cambi di comportamento** che si sono distinti da allarmi costituenti rumore, sia causati da potenziali **minacce**, sia da incerta origine.

Oltre a queste considerazioni, un amministratore che conosce la propria rete sarebbe in grado di interpretare meglio i dati empirici, e fornire conclusioni più approfondite e legate al contesto, ma sono sufficienti per dare risposta alla domanda posta in testa a questo capitolo 7.

7.3 Anomalia 2



Il tool ha segnalato come outlier il secondo rating dell'host 102.56.178.53@62, corrispondente alla finestra temporale dalle 12:54 alle 13:54

Il grafico ci dice che rispetto alla finestra precedente le cause dell'innalzamento del rating sono principalmente due:

- L'host 102.56.178.53@62 è stato server in una sessione di accesso remoto, con un host interno alla rete e uno esterno.
- Ci sono stati almeno due trasferimenti di file binari con percorsi mai visti prima

Verificando gli allarmi su ntopng, vediamo che l'unico allarme di tipo *Remote Access* riguarda un client con IP pubblico di proprietà dell'azienda che ha messo a disposizione la rete di test. Questo lo può pressoché scagionare dal sospetto di minaccia, e inoltre il server non ha risposto.
Esaminando i flussi fra i due host nel corso della giornata, vediamo che il client ha fatto tre tentativi di aprire una connessione Telnet, tutti senza risposta.
Ricordiamo che consideriamo critica una sessione di accesso remoto solo quando non è stata effettuata in precedenza, dunque i successivi due tentativi non contribuiscono al rating nelle successive finestre temporali.

Begin	End	Protocol	Application	Flow	Pkts	Bytes	Thpt	L7 Category	Status
11:32:11	11:32:11	TCP	HTTP Guess	102.8.253.125@62:80	3 Pkts	220 Bytes	1.76 kbps	Web	TCP Connection Issues
10:22:49	10:22:50	TCP	HTTP DPI	102.8.253.125@62:80	13 Pkts	1.37 KB	5.62 kbps	Web	Binary Application Transfer
10:22:49	10:22:50	TCP	HTTP DPI	102.8.253.125@62:80	13 Pkts	1.5 KB	6.15 kbps	Web	Binary Application Transfer
12/04/2023 14:59:47	12/04/2023 14:59:48	TCP	HTTP DPI	102.8.253.125@62:80	12 Pkts	3.19 KB	13.07 kbps	Web	Normal
12/04/2023 14:59:47	12/04/2023 14:59:48	TCP	HTTP DPI	102.8.253.125@62:80	12 Pkts	3.17 KB	13.0 kbps	Web	Normal
12/04/2023 14:59:46	12/04/2023 14:59:47	TCP	HTTP DPI	102.8.253.125@62:80	13 Pkts	1.5 KB	6.15 kbps	Web	Binary Application Transfer
12/04/2023 14:59:46	12/04/2023 14:59:47	TCP	HTTP DPI	102.8.253.125@62:80	13 Pkts	1.37 KB	5.62 kbps	Web	Binary Application Transfer
12/04/2023 13:50:20	12/04/2023 13:50:21	TCP	HTTP DPI	102.8.253.125@62:80	15 Pkts	4.17 KB	17.09 kbps	Web	Binary Application Transfer
12/04/2023 13:50:19	12/04/2023 13:50:19	TCP	HTTP DPI	102.8.253.125@62:80	12 Pkts	3.18 KB	26.04 kbps	Web	Normal
12/04/2023 13:50:19	12/04/2023 13:50:19	TCP	HTTP DPI	102.8.253.125@62:80	12 Pkts	3.19 KB	26.14 kbps	Web	Normal

Figura 7.1: Flussi con IP 102.8.253.125

Per quanto riguarda i due allarmi *Binary Application Transfer*:

- Il percorso associato al prima inizia con *https://edgedl.me.gvt1.com/edgedl/delta-update/...* e prosegue con una codifica in esadecimale lunga circa 200 caratteri. Il server è di Google. Sia il percorso che la proprietà dell'host indicano che si tratta di un update.
  È comune che i percorsi di file associati a un update risultino univoci a causa di una parte dell'URL generata dinamicamente in base alla versione e al contenuto dell'aggiornamento.

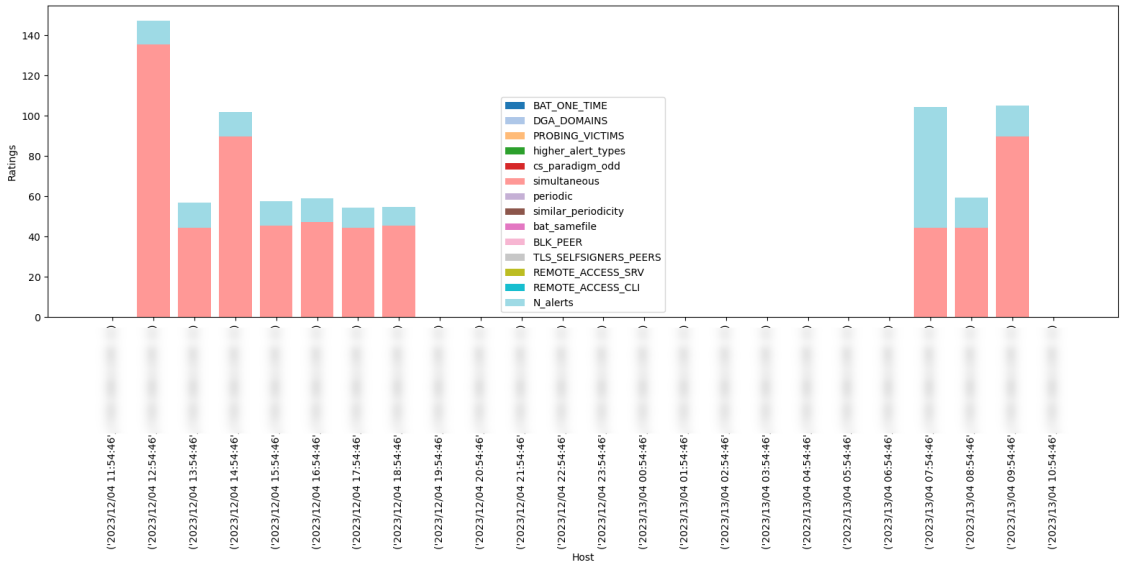
- il secondo percorso è *https://ma.dun.163.com/api/v1/s/m*, e il server è cinese, di proprietà di un azienda di cui non si trova molta documentazione online. Esaminando i 10 flussi associati a tale host, vediamo che 3 client (fra cui 102.56.178.53@62) hanno comunicato con esso con protocollo HTTP, sempre con sessioni di durata inferiore al secondo, e talvolta simultaneamente. I file trasferiti hanno lo stesso prefisso *ma.dun.163.com/api* e un suffisso leggermente variabile a volte seguito da una lunga codifica.

Non è possibile dire con certezza se questi trasferimenti siano dannosi o meno, tuttavia, vista la durata delle sessioni e la dubbia origine degli URL, non si direbbe

traffico causato dalla normale navigazione di un utente umano, né è chiaro perché tali client dovrebbero autonomamente contattare un server cinese pressoché sconosciuto.

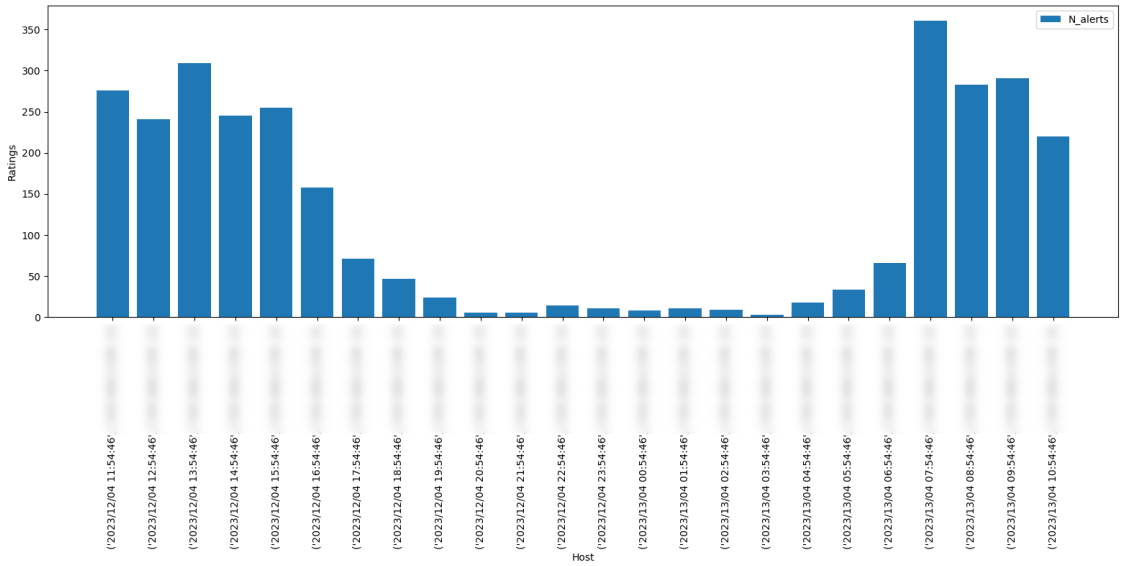
Lo si può considerare come un **problema**, essendo un comportamento indesiderato o quantomeno sospetto.

7.4 Anomalia 3



Il tool ha riportato come outlier il rating di 10.0.7.242@13 nella 21° finestra temporale, dalle 07:54 alle 08:54.

Vediamo che l'innalzamento del rating è dovuto ad alcuni allarmi simultanei e ad un'importante aumento dell'intensità degli allarmi. Osservando il grafico sottostante, che rappresenta la quantità di allarmi per ogni finestra temporale (*intensità*, per brevità), osserviamo che effettivamente vi è stato un picco:

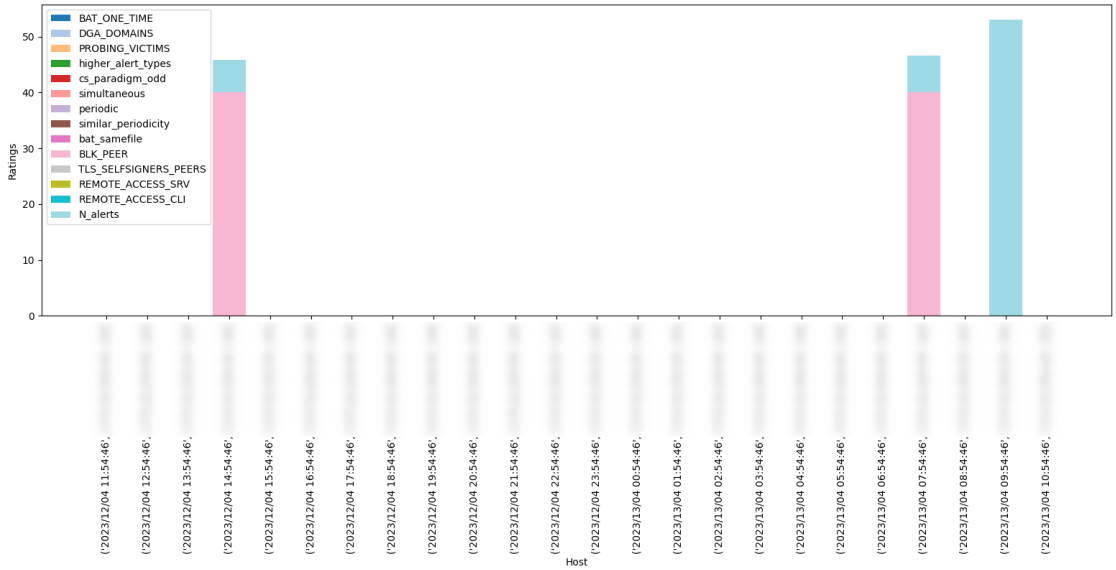


Osservando gli allarmi su `ntopng`, vediamo che sono quasi del tutto di tipo *HTTP Suspicious User-Agent*, e ve ne sono tre con uno stesso client locale esportati simultaneamente.

L'host 10.0.7.242@13 è un server proxy, questo spiega il gran numero di flussi HTTP e i relativi allarmi per User-Agent ritenuti sospetti, e anche il picco di allarmi, crescita avviata all'inizio della giornata lavorativa. Il fatto che appena una connessione con un client locale abbia portato a tre allarmi esportati simultaneamente si direbbe essere una coincidenza non degna di attenzione.

In questo caso abbiamo un **falso positivo**.

7.5 Anomalia 4



Il tool ha riportato come outlier il rating di 13.29.224.24@4 nella 21° finestra temporale, dalle 07:54 alle 08:54.

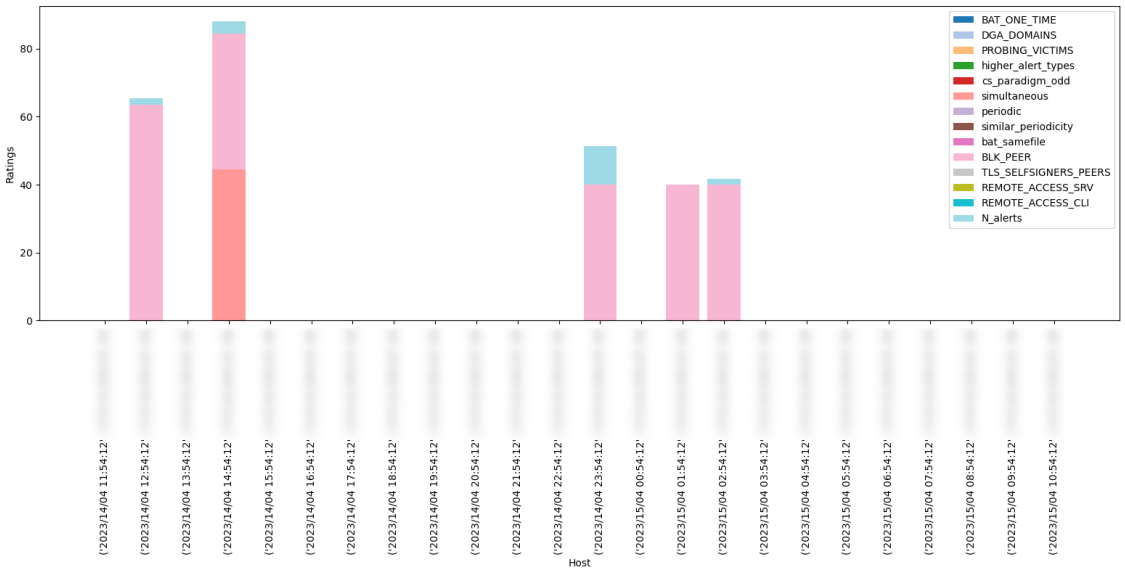
L'allarme è stato causato da una query Reverse DNS verso un server russo blacklistato, appartenente a un ISP americano. L'IP nella query è per metà numerico e per metà testuale, e la risposta del server è stata un record NXDOMAIN.

È stato l'unico flusso in tutta la giornata con quel server.

Dai flussi non appare chiaro il motivo dietro alla richiesta reverse DNS, né perché sia stata inviata a un server russo, che forse potrebbe essere un proxy di un servizio DNS.

Al di là della natura non chiara del server che ha servito la richiesta DNS, dietro all'allarme non sembra esservi una minaccia, o un cambio di comportamento rilevante.

7.6 Anomalia 5



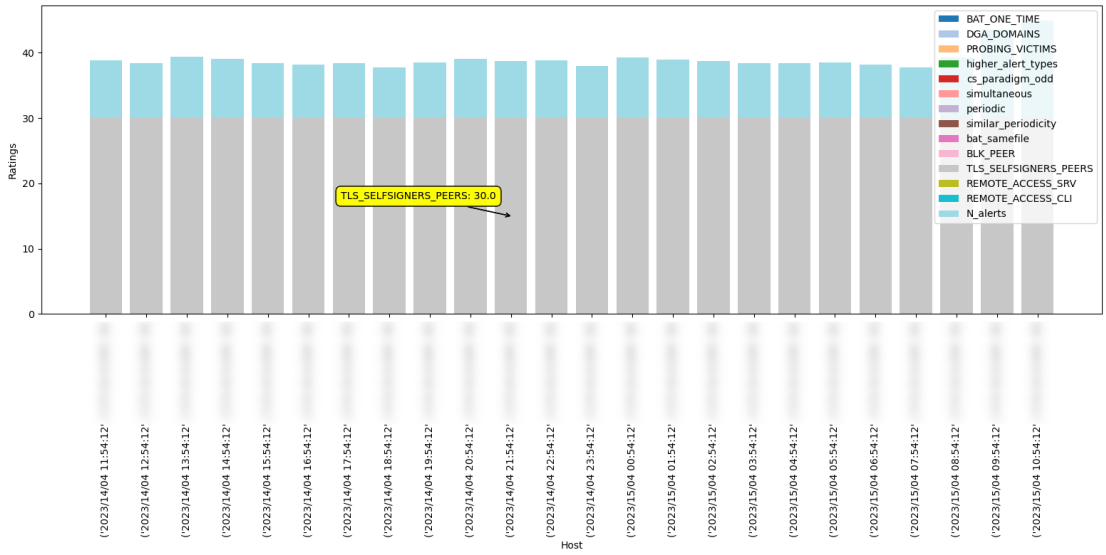
Il modello ha segnalato come anomalo il quarto rating 171.30.214.87@0, corrispondente alla finestra temporale dalle 14:54 alle 15:54.

Tale host ha esportato quattro allarmi *Blacklisted*, di cui solo uno di questi è bidirezionale e il server non ha rifiutato la connessione, causando la barra rosa superiore "BLK_PEER", mentre gli altri tre sono giunti simultaneamente causando quella sottostante. Questi ultimi tre sono poco rilevanti, in quanto il server non ha risposto e l'host client appartiene a Censys, azienda di sicurezza informatica.

Il flusso bidirezionale è con il client cinese 183.136.225.32, che è stato segnalato ben 11000 volte su abuseipdb.com. A tale client sono associati 10 flussi, i cui i server non hanno risposto o hanno rifiutato la connessione, tranne in due casi in cui tutti i byte inviati provengono dal server.

Abbiamo individuato un comportamento sospetto, probabile probing, ma non è chiaro se abbia portato alla esfiltrazione di informazioni sensibili.

7.7 Anomalia 6

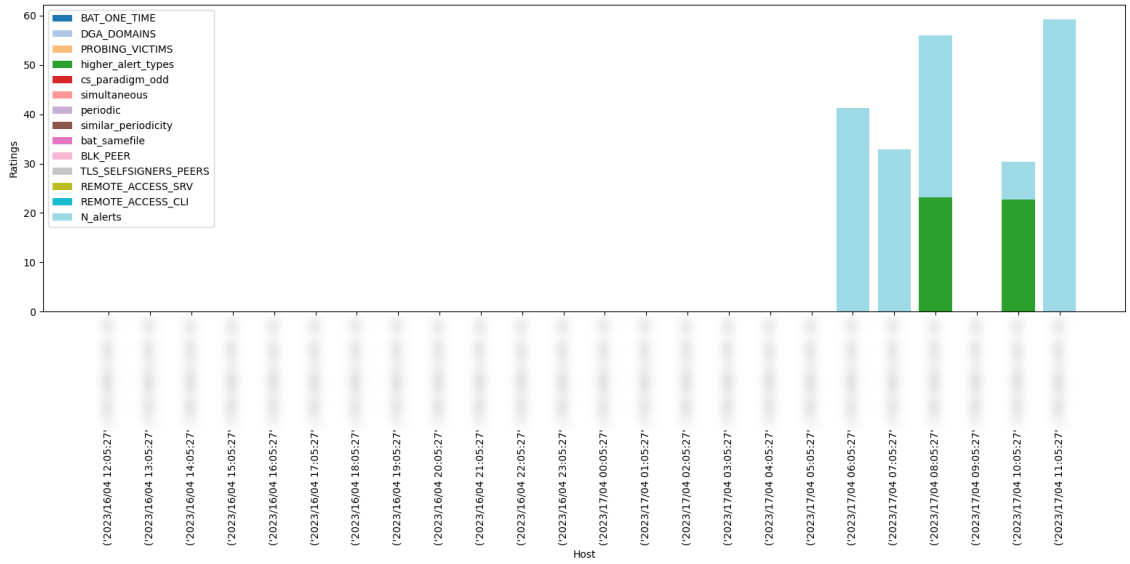


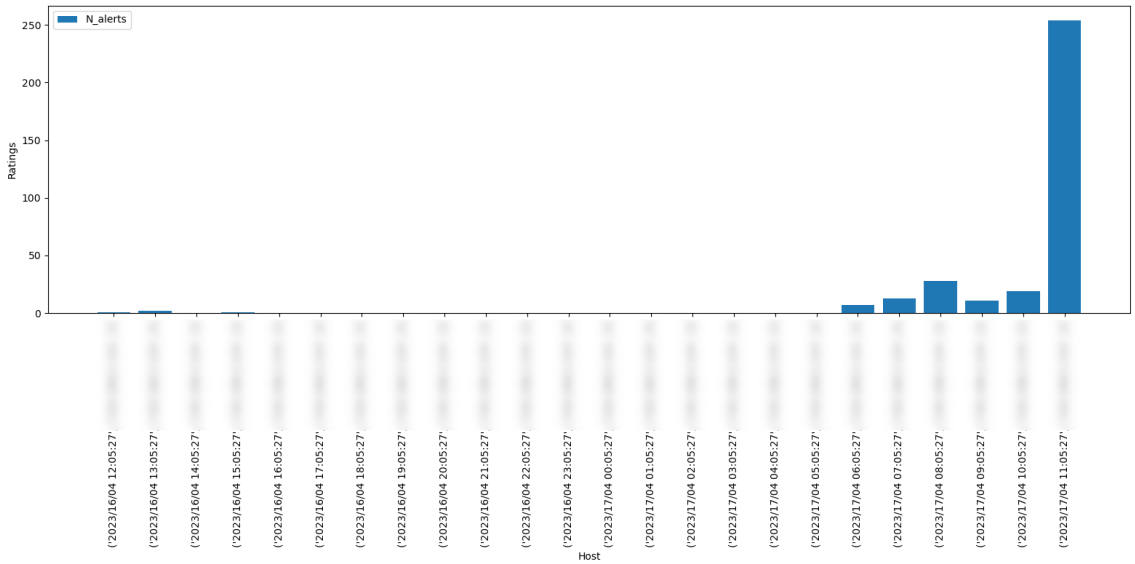
Il modello ha segnalato come anomalo l'ultimo rating 142.1.92.43@12, corrispondente alla finestra temporale dalle 10:54 alle 11:54.

L'host genera molti allarmi TLS Certificate Self Signed verso uno stesso server (remoto) appartenente ad un'azienda di sicurezza informatica.
Il grafico evidenzia come questi allarmi rientrino nel comportamento standard dell'host 142.1.92.43@12, e il cambio nel rating nell'ultima finestra temporale è unicamente dovuto al numero di allarmi, che non sembra avere una causa particolare.

Abbiamo dunque un **falso positivo**.

7.8 Anomalia 7





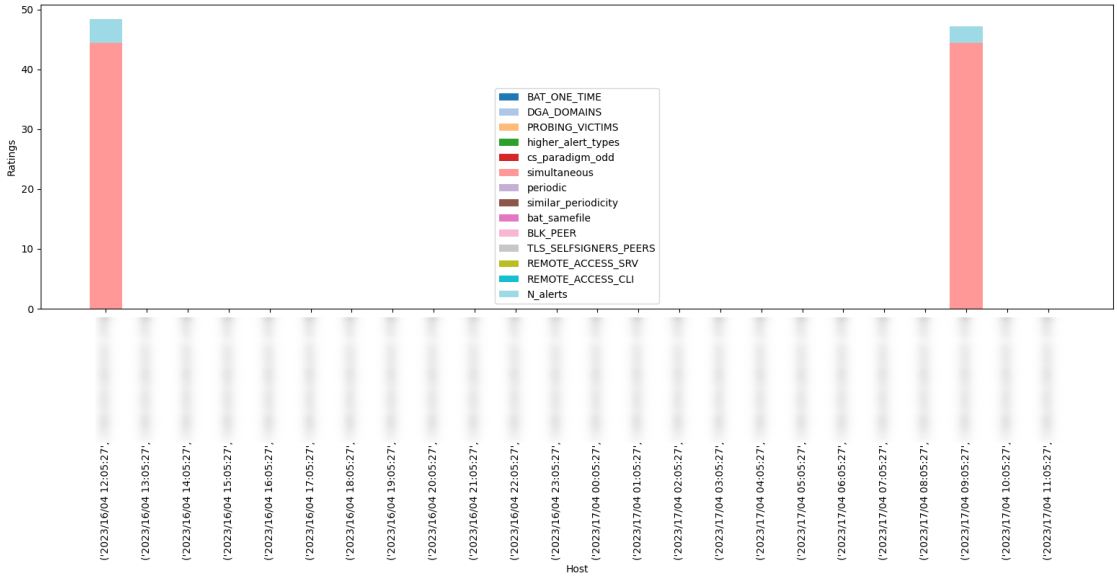
Il modello ha segnalato come anomalo l'ultimo rating 87.184.214.145@5, scaturito dal gran numero di allarmi da esso esportati.

Ad eccezione di alcuni *Suspicious DGA Domain* e *TLS Unsafe Cipher* di poco rilievo e scorrelati, tali allarmi sono quasi tutti HTTP Suspicious User-Agent, causati da traffico HTTP verso msftconnecttest.com/connecttest.txt in cui lo user agent è mancante. Analizzando i flussi vediamo che ve ne sono tanti associati a tale server e URL, e che spesso hanno lo User Agent *Microsoft NCSI* indicato.
La comunicazione è un banale check.

Non è chiaro come mai lo User-Agent sia mancante, ma in ogni caso non è un problema grave, probabilmente dovuto al fatto che il client richiedente sia un server proxy, il che spiegherebbe anche l'alta quantità di traffico che lo attraversa.

Abbiamo dunque un **falso positivo**.

7.9 Anomalia 8



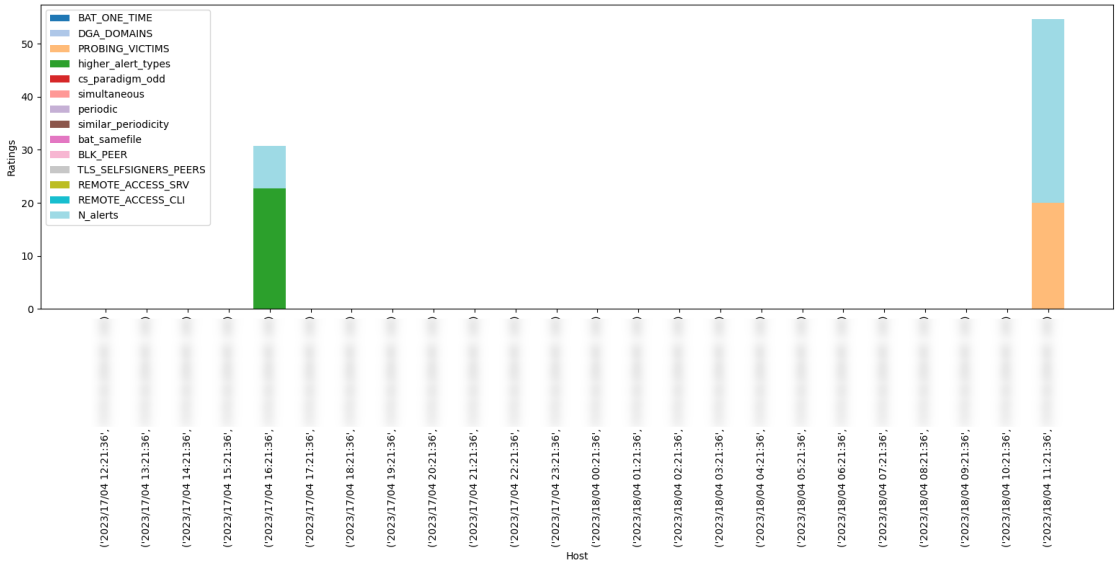
Il modello ha segnalato come anomalo l'ultimo rating non nullo dell'host russo 54.24.81.215@65, a causa di allarmi esportati simultaneamente.

Nella corrispondente finestra temporale vi sono 3 allarmi *Blacklisted*, in cui il client 54.24.81.215@65 ha inviato pacchetti SYN, bloccati da un successivo firewall, a tre server distinti che non hanno risposto.

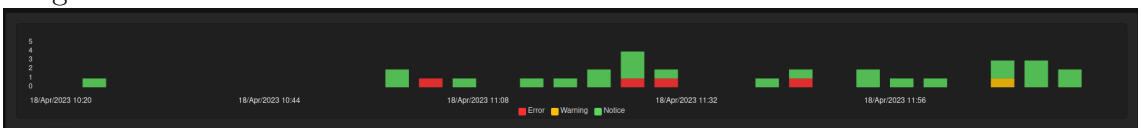
Analizzando i flussi durante le 24 ore di osservazione dell'host in considerazione, notiamo che ha tentato, fallendo, di aprire una connessione con svariati server.

Sappiamo che nella rete in cui si sono svolti i test, c'è un firewall configurato per bloccare il traffico da e per la Russia. Se non ci fosse stato, o fosse stato configurato diversamente, avremmo scoperto un prober.

7.10 Anomalia 9



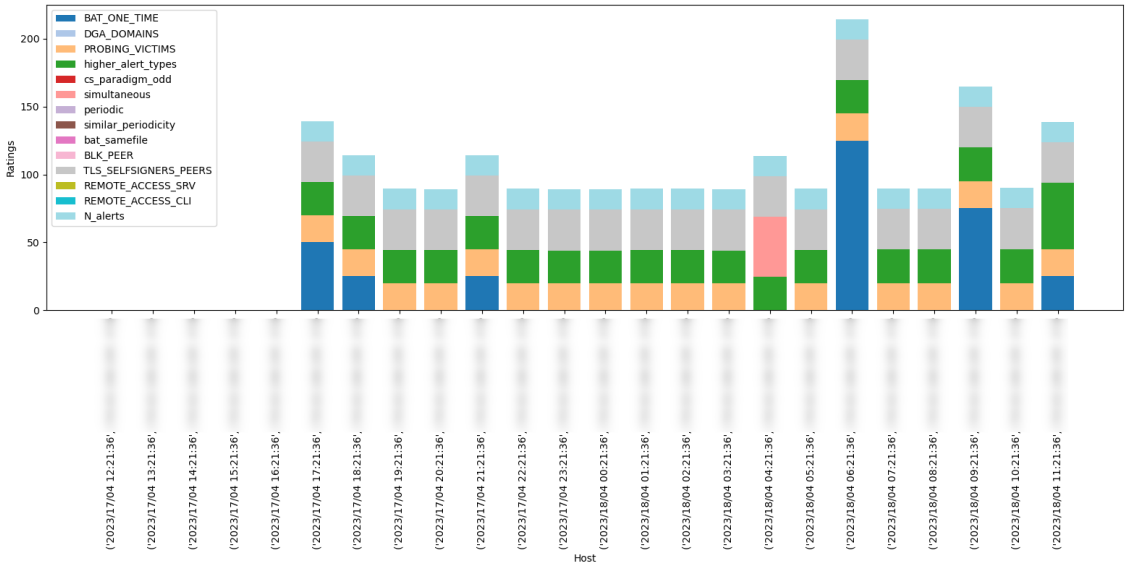
L'ultimo rating dell'host 192.168.224.41@32 è risultato un outlier. Tale punteggio è dovuto all'intensità degli allarmi e all'essere risultato una presunta vittima di probing.



Questo grafico di ntopng mostra il numero di allarmi relativi a flussi unidirezionali ricevuti nelle ultime due finestre temporali, dunque dalle 10:21 alle 12:21. Notiamo che effettivamente il numero di allarmi nella seconda metà del grafico, dunque nell'ultima finestra temporale, è più alto e più regolare. Gli host client appartengono a range IP eterogenei e i flussi, vengono associati a sospetto probing nonostante siano su protocollo UDP perché riguardano traffico DNS, che presuppone richiesta e risposta, dunque traffico bidirezionale. L'host 192.168.224.41@32 è una macchina server in una vlan di server, ma non è chiaro come mai si sia verificato questo traffico DNS unidirezionale, che non sembra essere stato causato da un problema di sicurezza.

Anomalie come questa evidenziano la difficoltà nel classificare situazioni non chiare, che presentano degli aspetti non nella norma ma al contempo non mostrano evidenti rischi. Una classificazione precisa richiede la conoscenza di un amministratore di rete, se necessario combinata con analisi più approfondite.

7.11 Anomalia 10



Il 19° rating di 214.32.138.105@42 è risultato anomalo, quello corrispondente alla finestra temporale dalle 6:21 alle 7:21.

Appare evidente dal grafico, ed è confermato analizzando gli allarmi, che il comportamento ordinario dell'host 214.32.138.105@42 genera un rating piuttosto stabile, dunque abbiamo ottenuto una buona classificazione del rumore attraverso i criteri da noi individuati.

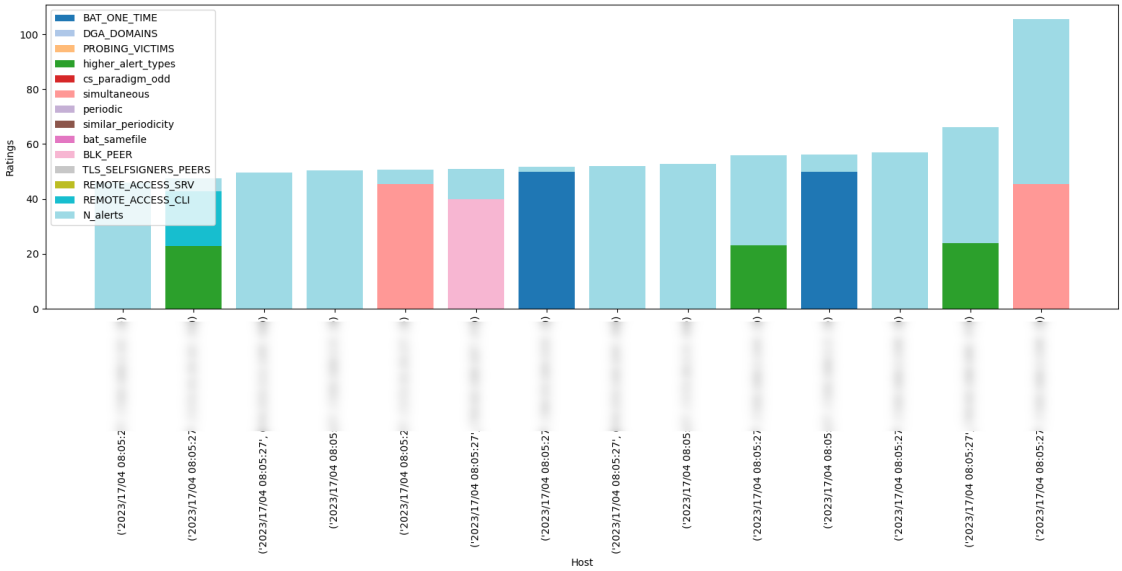
Ciò che ha causato l'innalzamento nel rating sono dei trasferimenti di file binari, con percorsi prima mai visti.
Avendo escluso alcuni percorsi "ordinari", rimangono dei trasferimenti di file con percorsi contenenti *xtrapath3.izatcloud.net*, che è un dominio di Qualcomm utilizzato per servizi di geolocalizzazione.

Qualora non fossero installati su tale server applicativi che richiedono geolocalizzazione, tali trasferimenti potrebbero essere causati da indesiderata telemetria o esfiltrazione, e meritare attenzione.

7.12 Analisi su Singola Finestra

Come menzionato nel paragrafo 5.2.1 è inopportuno rilevare degli outlier fra i rating di host diversi in una singola finestra temporale, in quanto equivale a confrontare tipi di traffico potenzialmente molto diversi.

A titolo esemplificativo, consideriamo il seguente grafico, rappresentante gli host con rating superiori a una soglia minima in una singola finestra temporale.



Anche in casi relativamente rari come questo in cui gli host hanno rating simili, seppur causati da feature diverse, l'ultimo rating, che risulterebbe outlier, è relativo di un server proxy, a cui sono associati molti più flussi della media, ed è dunque naturalmente soggetto a generare più allarmi e potenzialmente mostrare più criticità rispetto agli altri host.

Nella rete su cui sono stati svolti i test sono presenti anche delle subnet ristrette di host da cui ci si aspetta traffico simile.
È stato tentato di applicare il modello esclusivamente a tali subnet, ma senza successo, a causa dell'insufficienza di allarmi generati dagli host in queste sottoreti, eccessivamente sporadici e ridotti come quantità per poter eseguire confronti e analisi.

È stato tentato anche un retuning di pesi e parametri, ma non ha potuto sopperire a finestre temporali vuote, o con appena 2 o 3 host con rating non nullo.

7.13 Tabella Riassuntiva

La tabella che segue riporta i risultati ottenuti da ciascuna delle anomalie discusse nelle precedenti sezioni, secondo la classificazione C.1 descritta in testa al capitolo.

#Anomalia	Problema	Cambio Comportamento	Falso Positivo
1	X	X	
2	X		
3			X
4			X
5	X		
6			X
7			X
8	X		
9		X	
10	X		

Tabella 7.1: Tabella riassuntiva risultati anomalie

Riassumendo, sono state esaminate sia situazioni che hanno condotto a potenziali problemi sia a dei falsi positivi, causati da variazioni nel rumore degli allarmi generati, e in parte dal tuning dei parametri (come in Anomalia 7). Sono state viste anche situazioni di difficile catalogazione (Anomalia 1, Anomalia 9) in cui è stata riscontrata un anomalia non associata ad un evidente rischio. Inoltre la prima riga della tabella evidenzia come un rating anomalo possa essere indice di molteplici cambi originati da possibili minacce e non, rendendo le classificazioni C.1,C.2,C.3 non mutuamente esclusive.

Aver avuto a disposizione dei dataset firmati avrebbe permesso di realizzare un procedimento di verifica automatizzato e dunque una validazione più robusta, supportata da un maggior numero di dati e di statistiche.
Tuttavia, come riportato nel capitolo conclusivo 8, i risultati positivi discussi mostrano che l'analisi dei rating nel tempo può con successo evidenziare delle potenziali minacce e le cause che vi sono dietro, rendendo promettente l'applicazione del modello.

Matrici di Confusione

Le matrici di confusione [32] vengono usate come misura delle performance di un modello di classificazione comparando il numero di veri positivi/negativi con il numero di predetti positivi/negativi, e hanno rappresentazione simile a quella a destra.

		Predetti	
Veri		TN	FP
		FN	TP

Più in generale, definite N possibili classificazioni per gli elementi in input, si compara il numero di elementi *veramente* appartenenti con il numero di elementi *predetti* appartenenti a ciascuna classe.

La discussione dei casi esaminati *non* ambiva a fornire una significativa misura delle performance, ma piuttosto ad illustrare il funzionamento del modello realizzato e a fornire una panoramica della sua efficacia o mancanza in alcune situazioni, e la tabella 7.1 riportata vuole solo riassumere brevemente tale discussione.
Essa si distingue da una matrice di confusione in quanto non è dato sapere quanti siano i falsi negativi —conoscerli richiederebbe una minuziosa analisi di tutto il traffico sulla rete— e non si riferisce ad un unico dataset, dunque anche statistiche come *Recall* e *Precision* non sussistono.

Capitolo 8

Conclusioni

8.1 Requisiti

Riportiamo la tabella 3.2 che sintetizza i requisiti del tool posti inizialmente.

Inferire *allarmi di livello superiore* in una finestra temporale di 60 minuti
Calcolare *rating* per ciascun host in base a essi
Costruire *serie temporale* dei rating di ciascun host ripetendo il calcolo 24 volte
Individuare gli outlier sulle serie temporali generate
Organizzarli in una vista che indichi host, finestra temporale di riferimento, e criteri di criticità rispettati

Nei capitoli 4 e 5 è spiegato come sono stati raggiunti i primi 4 punti della tabella, e in 6 vi sono alcuni dettagli implementativi, mentre con la figura 6.1 forniamo un esempio della vista richiesta nell'ultimo requisito, realizzata con un grafico a barre impilate. Nel capitolo 7 abbiamo osservato l'effettivo funzionamento del modulo sviluppato discutendo dei risultati che ha generato in output, e confrontandoli con i dati grezzi per verificare la correttezza delle inferenze rispetto alla ground truth.

8.2 Obiettivo

È altrettanto importante stabilire se il modello implementato dal tool funzioni o meno rispetto all'**obiettivo** 1.1 di questo studio.
Guardando ai risultati discussi nel precedente capitolo 7 e riassunti nella matrice 7.1, vediamo che il modello ha portato in più di un'occasione risultati positivi, rispecchiati da dati empirici, ma anche alcuni falsi positivi, che si sono rivelati dei vicoli ciechi.
È opportuno menzionare che il tempo avuto per osservare la rete di test e un contatto con l'amministratore hanno permesso di determinare alcune informazioni di contesto che hanno aiutato ad eliminare una parte, seppur piccola, del rumore e dunque a rendere l'applicazione del modello più efficace; è stato possibile classificare alcuni di questi presunti problemi come rumore perché analizzati in seguito ad essere stati evidenziati dall'implementazione del modello e dai test realizzati durante il suo sviluppo.
Il confronto con l'amministratore di rete sui risultati intermedi ottenuti in corso d'opera è stato un importante parere tecnico, che ha aiutato a selezionare i criteri descritti nel capitolo 4 e a scartarne altri. Persone che lavorano in questo ambito hanno conoscenza concreta ed esperienza diretta con le problematiche di sicurezza e possono discernere aspetti di interesse maggiore rispetto ad altri.

Ricordiamo anche che la rete esaminata è protetta da numerosi dispositivi e software di sicurezza, dunque nonostante possano non esserci stati danni relativi alle potenziali criticità evidenziate dal tool, a noi interessa che siano state evidenziate, astraendo quanto basta dal contesto.

Fra i riscontri positivi del tool abbiamo trovato:

- Flussi asincroni con host blacklistati, fra cui probabili prober (7.2,7.6,7.9)
- Trasferimenti di file (7.3, 7.11)

- Sessioni accesso remoto (7.2, 7.3)

Il fatto che l'applicazione del modello abbia avuto esiti positivi è promettente e dimostra che il modello, rispetto all'obiettivo posto nell'introduzione 1.1, ha funzionato. Ha permesso di definire il comportamento degli host nella rete in termini di allarmi generati, rispetto a un insieme finito di categorie di questi per sfruttarne peculiarità e criticità, e di determinare attività asincrone e altre, invece, ripetute.

Tuttavia sulla base dei risultati osservati, di cui alcuni discussi in questa tesi e riportati nel capitolo 7, non possiamo concludere che il modello possa funzionare sempre e in qualsiasi rete, con qualsiasi traffico.
Il falso positivo ottenuto in 7.7, mostra come nonostante il modello abbia definito una buona traccia del comportamento dell'host, il valore outlier costituito dall'ultimo rating osservato non rispecchi né un problema né un vero cambio in tale comportamento, se non un leggero innalzamento dell'intensità degli allarmi, non significativo e di per sé di poco interesse.
Serie sparse come 7.9 e 7.10 sono relative ad host che hanno un traffico non nullo e esportano diversi allarmi nel tempo, ma che rispetto ai criteri da noi considerati non risultano particolarmente attivi; vista la natura (talvolta) asincrona del problema in 7.9, il rilevamento del valore "outlier" causato comunicazioni con host blacklistati è stato significativo e ha rivelato una potenziale minaccia, ma potrebbe essere opportuno distinguere una situazione del genere da una come quella in 7.7 che mostra un numero più cospicuo di osservazioni nella serie temporale, e fornire due classificazioni diverse.

8.3 Miglioramenti futuri

Avendo avuto ulteriore tempo e modo, per poter stabilire con più certezza l'affidabilità e l'efficacia del modello sarebbe stato opportuno:

- Svolgere dei test automatizzati su un training set composto da esempi firmati. Ciò permetterebbe di costruire una appropriata matrice di confusione con molte anomalie e per ciascuna di queste avere una classificazione binaria immediata del tipo "*individuata/non individuata dal modello*". Inoltre sarebbe possibile eseguire tuning più preciso dei pesi delle feature 5.1.1 e dei parametri degli algoritmi di rilevazione outlier 5.2 ed esaminare quali valori portino ad una maggiore accuratezza, per quanto maggiore accuratezza sul training set non sempre significhi maggiore accuratezza sui dati reali[7].

- Applicare il modello su altre reti con tipologie diverse di traffico, per vedere se l'affidabilità dei risultati è indipendente da questa o se, al contrario, vi è strettamente legata. Dunque determinare se i criteri di criticità 4.1.1 su cui si basa il modello si rivelano flessibili, o meno.

Secondariamente sarebbe interessante poter svolgere dei test su una rete, o una sottorete, di host da cui ci si aspetta lo stesso tipo di traffico, per verificare se è possibile, e soprattutto produttivo, determinare outlier in una singola finestra temporale, comparando i rating di host distinti.
Abbiamo illustrato in 7.12 perché sia inappropriato applicare il modello su host con traffico eterogeneo, e come non sia stato possibile testarlo sulle subnet a disposizione a causa di insufficienza di allarmi generati.

Si potrebbero individuare e aggiungere altri criteri di criticità e rispettive feature, nel caso in cui risultasse che questi migliorassero la definizione del comportamento degli host, e la conseguente rilevazione delle anomalie in essi.

Potrebbe essere interessante verificare se sia possibile applicare il modello a una finestra temporale mobile, realizzando un'implementazione che funzioni in streaming, dunque che lavori su dati ricevuti in tempo reale.

Ringraziamenti

Trovo doveroso dedicare qualche riga per esprimere i miei ringraziamenti. Innanzitutto al professor Luca Deri, mio relatore, e all'ing. Federico Vecchiatti, che ha messo a disposizione la rete su cui sono state svolte analisi e test: i loro consigli e la loro guida sono stati fondamentali nella stesura di questa tesi. In secondo luogo vorrei ringraziare il controrelatore e la commissione di laurea assegnati, per la valutazione di questo elaborato e del lavoro che c'è dietro, così come sono grato all'Università di Pisa per gli anni di studio, che hanno portato a numerose opportunità di imparare e apprendere. In ultimo, ci tengo a menzionare anche la mia famiglia, i miei amici e le persone a me più care: mi hanno accompagnato e supportato in questi mesi, contribuendo così alla realizzazione di questo elaborato.

Appendice A

Appendice

A.1 Codice sorgente

GitHub repo: github.com/frenzis01/ntopng_alert_analyzer

A.2 Richiami teorici

Per discutere quali sono gli allarmi di livello superiore che vogliamo generare e come individuarli, sono stati qui raccolti dei richiami ad alcuni strumenti matematici e strutture dati utilizzate.

A.2.1 Entropia

Il primo di questi è l'**entropia**, che noi calcoleremo sulle porte utilizzate da un host in un flusso di allarmi. L'entropia è adatta ai nostri bisogni, in quanto non tiene conto dell'ordine o della quantità in sé di porte usate, ma ci indica se le porte usate sono "poche" o "tante".

Intuitivamente, l'entropia misura quanto un sistema è "disordinato". In relazione alla teoria dell'informazione, spesso si parla di *Entropia di Shannon*, avendone quest'ultimo studiato estensivamente le proprietà. È un concetto comunemente discusso

in crittografia, e applicato su stringhe, ma è una questione di rappresentazione del dato, e nulla impedisce di applicarlo a degli interi come nel nostro caso, con alcune accortezze.

L'entropia di Shannon viene calcolata attraverso la seguente formula:

$$H(X) = -\sum_{x \in X} P(x) log_b(P(x))$$

Dove $P(x)$ è la probabilità che *l'evento x accada*, mentre $log_b(P(x))$ è spesso indicata come *autoinformazione* dell'evento x.
Quando si considera la codifica binaria dei dati si utilizza base del logaritmo $b = 2$, mentre scegliendo $b = e$ si ottiene la misura dell'automazione in *nats*, unità introdotta da Shannon.

Nel nostro scenario dobbiamo calcolare l'entropia di una sequenza a di m interi, possibilmente ripetuti, e infine normalizzarla.
Innanzitutto consideriamo l'insieme X di interi *distinti* che occorrono in a. Per ciascun $x \in X$ contiamo il numero di occorrenze o_x in a; in questo modo possiamo calcolare $P(x)$ nel seguente modo:

$$P(x) = \frac{o_x}{m}$$

Vogliamo fare in modo che $0 \leq H(X) \leq 1$, in modo che sia normalizzata; per far ciò scegliamo $b = \#X$.

Riassumiamo quanto detto nel sottostante snippet di codice Python.
La funzione `entropy(probabilities, base=1)` è importata dal package `scipy`, la cui documentazione è disponibile qui[1]; mentre `Counter(data)` restituisce un dizionario le cui chiavi sono gli elementi dell'oggetto `iterable` passato come parametro, e i cui valori sono il numero di occorrenze relativo a ciascun elemento. Maggiori dettagli sono disponibili nella documentazione[2].

[1] docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html
[2] docs.python.org/3/library/collections.html#collections.Counter

```python
def shannon_entropy(data):
    # Calculate the frequency of each element in the list
    frequency_dict = Counter(data)
    H_entropy = 0
    probabilities = []
    # Calculate the entropy
    for key in frequency_dict:
        # Calculate the relative frequency of each element
        # and the related probability
        probabilities.append(frequency_dict[key] / len(data))
    # Use l as the log base, to normalize the result and
    # get a value between 0 and 1
    l = len(frequency_dict)
    H_entropy = 0 if l == 1 else entropy(probabilities, base=l)
    return H_entropy
```

Riportiamo di seguito dei semplici esempi:

```
>>> shannon_entropy([1,2,3,4,5])
1.0000000000000002
>>> shannon_entropy([1,2,3,4,5,5])
0.9697238998682473
>>> shannon_entropy([5,5,5,5,5,4,5,5,5,5,5])
0.43949698692151334
>>> shannon_entropy([5,4])
1.0000000000000002
```

Gli ultimi due esempi mostrano come l'entropia sia nettamente più idonea rispetto a un banale "`COUNT DISTINCT`" a misurare quanto richiesto.

A.2.2 Deviazione Standard

La *deviazione standard* o *scarto quadratico medio*, tipicamente indicata con σ, fornisce una stima della variabilità di una popolazione o di un campione di dati.

Preso un campione di n unità, essa si calcola attraverso la formula:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n}(x_i - \bar{x})^2}$$

Dove $\bar{x}$ è la media campionaria sul campione considerato.

Anche per la deviazione standard vogliamo avere una misura normalizzata, per noi $\sigma = 10$ su un campione dove $\bar{x} = 30$ ha significato diverso rispetto alla stessa $\sigma = 10$ su un campione dove $\bar{x} = 500$.
A questo proposito risulta utile il *coefficiente di variazione*, o *deviazione standard relativa*, calcolato come segue:

$$\sigma^* = CV = \frac{\sigma}{\bar{x}}$$

Nel nostro scenario considereremo intervalli di tempo, e CV ci permetterà di determinare quando un campione di tali intervalli risulta tendenzialmente costante, con una tolleranza proporzionata al valore medio del campione.
In questo modo potremo classificare flussi di allarmi come periodici o meno, aspetto che discuteremo in dettaglio nella sezione successiva.

A.2.3 Media e outlier

Con **outlier** in statistica ci si riferisce a un valore anomalo, distante rispetto alle altre osservazioni disponibili, anche se non ne esiste una definizione formale.

Riportiamo due sistemi entrambi relativamente semplici per identificare degli outlier su un insieme di dati, ma che producono risultati leggermente diversi:

- La via più immediata è utilizzare la **media** sulle osservazioni eseguite.

  $$\bar{x} = \frac{\sum_{i=1}^{n}(x_i)}{n}$$

  Determinata la media, possiamo considerare outlier valori $x < \bar{x}$ o $x > \bar{x}$, a seconda del contesto e delle esigenze.

- Un altro sistema è sfruttare come indice di dispersione lo **scarto interquartile** I definito come segue:

$$I = q_3 - q_1 \tag{A.1}$$

  Ricordiamo che i *quartili* q_1, q_2, q_3, q_4 sono valori che ripartiscono in quattro parti di uguali quantità la popolazione di osservazioni.

  Determinato I, si considerano outlier i valori che non appartengono al seguente intervallo:

$$[q_1 - \mathbf{k}I, q_3 + \mathbf{k}I]$$

  Con $\mathbf{k}$ costante generalmente fissata a 1 o 1.5.

  In questo modo si ottiene una classificazione più stringente rispetto alla precedente.

A.2.4 Strutture dati

Nel codice è stato fatto uso estensivo di dizionari e liste. I dizionari sono costituiti da una serie di coppie $\langle chiave : valore \rangle$ e sono implementati come tabelle hash, le quali permettono di accedere in tempo costante $\mathcal{O}(1)$ al valore corrispondente a una certa chiave.

Inoltre le chiavi sono univoche e dunque costituiscono un *insieme*, in cui non esistono duplicati: proprietà che si rivela utile al fine di non ridondare eccessivamente le informazioni.

Ad esempio, piuttosto che avere dei valori ripetuti in una lista,

```
l = ["A","A","A","A","B","B"]
```

può essere più leggibile e comodo mostrare un dizionario che associa ad ogni valore un contatore

```
d = {"A" : 4, "B" : 2}
```

In altri casi, quando si utilizzano valori già univoci o il problema sopra menzionato non si pone in forma lampante, una lista è più opportuna e introduce meno overhead.

A.3 Flussi

Alcuni criteri riguardano *flussi* di allarmi, non singoli allarmi.

La nozione di flusso viene dall'aggregazione dei pacchetti realizzata da NetFlow e IPFIX, volta a fornire un livello di astrazione superiore al singolo pacchetto, che permetta di avere una visione più chiara e leggibile di quali sono le comunicazioni attive in una rete. NetFlow permette la definizione di chiavi personalizzate per la creazione dei flussi, ma dalla versione 5, di default raggruppa pacchetti utilizzando come chiave:

Source IP
Source port
Destination IP
Destination port
protocol L3
Type of Service
Input interface

Tabella A.1: NetFlow Key

Riprenderemo il concetto di flusso, sarà molto importante correlare allarmi che condividono host e tipologia allarme.

A.4 Snippet codice

Client server paradigm

```
# In the client-server paradigm, the common behavior is that
# srv uses always the same known port, while clients use ephimeral ones
# We can set an entropy threshold to determine when srv and clients are
# behaving oddly
def is_odd(x,vlan_id):
    # Note: exclude not client-server paradigm associated alerts
    excludes = ["blacklisted"]
    if (GRP_CRIT != GRP_CLI
        and x["alert_name"] not in excludes
        and x["srv_port_S"] >= SRV_ODD_PORT_S_TH
```

```
11          and x["srv_port_count"] >= SRV_ODD_PORT_COUNT_TH
12          and is_server(vlan_id)):
13          return "odd_server"
14      # A client is odd if uses the SAME port with MANY servers
15      if (GRP_CRIT != GRP_SRV
16          and x["alert_name"] not in excludes
17          and x["cli_port_S"] <= CLI_ODD_PORT_S_TH
18          # and x["cli_port_count"] <= CSODD_PORT_COUNT_TH
19          and x["srv_ip_S"] >= CSODD_IP_S_TH
20          and is_client(vlan_id)):
21          return "odd_client"
```

Subsequent common substrings

L'algoritmo utilizzato per determinare le porzioni condivise fra domini è il seguente, e si noti che prende come input liste di stringhe, ovvero il valore di ritorno di `"dumb.domain.com".split(".")`.

```
1  # @returns the longest sequence of subsequent common substrings
2  # between l1 and l2
3  def find_longest_common_subsequent_substrs(l1:list,l2:list):
4     # @returns the first sequence of subsequent common substrings
5     # and the index of the last matching substring of the longer list
6     def find_first_common_subsequent_substrs(l1:list,l2:list):
7        # a is the shortest list, b the longest
8        a = l1 if (len(l1) <= len(l2)) else l2
9        b = l2 if (len(l1) <= len(l2)) else l1
10
11       # res holds the FIRST sequence of substrings
12       res = []
13       for i in range(len(a)):
14          for j in range(len(b)):
15             # found some matches, but no more
16             # NB 'j-1'
17             if(len(res) and (a[i] != b[j])):
18                return res,j-1
19
20
21             # found another subsequent match
22             if (a[i] == b[j]):
```

```
23                # add to resulting list
24                res.append(b[j])
25
26                # if there are no more elements in the shortest
27                # list then exit
28                if i + 1 == len(a):
29                    return res,j
30                # else move onto the next element
31                i += 1
32        return res,j
33
34    # a is the shortest list, b the longest
35    a = l1 if (len(l1) <= len(l2)) else l2
36    b = l2 if (len(l1) <= len(l2)) else l1
37
38    res = []
39    for i in range(len(b)):
40        tmp,index = find_first_common_subsequent_substrs(a,b[i:])
41        # if res is longer than half of b
42        # we are sure there ain't longer sequences
43        if (len(tmp) > len(b)/2):
44            return tmp
45
46        # update res
47        res = tmp if (len(tmp) > len(res)) else res
48
49        index += 1 # consider element after the last matching
50        # if there isn't a chance to find a longer matching sequence
51        # exit
52        if (len(b) - index < len(res)):
53            return res
54        # else try to find a longer matching sequence
55        i = index
56    return res
```

Listing A.1: Longest common subsequent substrings

Si può notare come il "best match" per un nome X, determinato quando quest'ultimo viene inserito, potrebbe cambiare in un secondo momento a causa della natura mutevole del dizionario, le cui chiavi vengono aggiornate di continuo, ma ciò non costituisce un problema, ci è sufficiente fornire dei raggruppamenti qualitativi,

non "assoluti"

Nested dictionaries counter

La seguente funzione serve ad incrementare dei contatori che sono valori di un dizionario di dizionari.

Segue lo stile *Look Before You Leap* (LBYL), opposto a *Easier to Ask Forgiveness Than Permission* (EAFP), adottato in situazioni diverse.

```
def add_to_dict_dict_counter(s:dict,k,v):
    if (k not in s):
        s[k] = {v : 1}
    else:
        if (v not in s[k]):
            s[k][v] = 1
        else:
            s[k][v] += 1
```

Singleton dictionary

```
def addremove_to_singleton(a: dict, v, value):
    if (v in a.keys()):
        a[v] = -1
    else:
        a[v] = value
    return a[v]
```

Il valore `-1` è un valore speciale che indica che `v` è stato inserito nel dizionario più di una volta.

Bibliografia

[1] Ahmed, T., Siraj, M. M., Zainal, A., and Mat Din, M. A taxonomy on intrusion alert aggregation techniques. In *2014 International Symposium on Biometrics and Security Technologies (ISBAST)* (2014), pp. 244–249.

[2] Apruzzese, G., Marchetti, M., Colajanni, M., Zoccoli, G. G., and Guido, A. Identifying malicious hosts involved in periodic communications. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)* (2017), pp. 1–8.

[3] Bahrbegi, H., Habibizad Navin, A., Ahrabi, A. A. A., Mirnia, M. K., and Mollanejad, A. A new system to evaluate ga-based clustering algorithms in intrusion detection alert management system. In *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)* (2010), pp. 115–120.

[4] Cai, S., Zhang, L., Palazoglu, A., and Hu, J. Clustering analysis of process alarms using word embedding. *Journal of Process Control 83* (2019), 11–19.

[5] Cronk, R. N., Callahan, P. H., and Bernstein, L. Rule-based expert systems for network management and operations: an introduction. *IEEE Network 2*, 5 (1988), 7–21.

[6] Debar, H., and Wespi, A. Aggregation and correlation of intrusion-detection alerts. vol. 2212, pp. 85–103.

[7] EliteDataScience. Overfitting in machine learning: What it is and how to prevent it, Jul 2022.

[8] Fernandez, S. Uc santa barbara to lead nsf-funded research institute for next-level ai-powered cybersecurity, May 2023.

[9] Hofstede, R., Celeda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *Communications Surveys Tutorials, IEEE 16* (04 2014), 2037–2064.

[10] Hubballi, N., and Goyal, D. Flowsummary: Summarizing network flows for communication periodicity detection. vol. 8251, pp. 695–700.

[11] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering 9*, 3 (2007), 90–95.

[12] Iglesias Vázquez, F., and Zseby, T. Analysis of network traffic features for anomaly detection. *Machine Learning 101* (12 2014).

[13] Jiaxuan Sun, Lize Gu, K. C. An efficient alert aggregation method based on conditional rough entropy and knowledge granularity. *Entropy (Basel) 22*, 3 (Mar. 2020).

[14] Julisch, K. Mining alarm clusters to improve alarm handling efficiency. pp. 12– 21.

[15] Julisch, K. Clustering intrusion detection alarms to support root cause analysis. *ACM transactions on information and system security (TISSEC) 6*, 4 (2003), 443–471.

[16] Khaleghy, H., and Izadi, I. Detection of correlated alarms using graph embedding.

[17] Lee, K., Kim, J., Kwon, K. H., Han, Y., and Kim, S. Ddos attack detection method using cluster analysis. *Expert Systems with Applications 34*, 3 (2008), 1659–1665.

[18] Lifen, L., and Changming, Z. Alert clustering using integrated som/pso. *2010 International Conference on Computer Design and Applications, ICCDA 2010 2* (06 2010).

[19] Noah, C., and Steinberg-Zwirek, D. Shining a light on malware beaconing - blogs.blackberry.com, Mar 2023.

[20] Odogwu, C. What is network scanning and how does it work?, Dec 2021.

[21] Qin, X., and Lee, W. Statistical causality analysis of infosec alert data. In *Recent Advances in Intrusion Detection: 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003. Proceedings 6* (2003), Springer, pp. 73–93.

[22] Rice, A., and Ringold, J. Command-and-control servers: The puppet masters that govern malware: Techtarget, Jun 2014.

[23] Salah, S., Maciá-Fernández, G., and Díaz-Verdejo, J. E. A model-based survey of alert correlation techniques. *Computer Networks 57*, 5 (2013), 1289–1317.

[24] Sanna Passino, F., and Heard, N. A. Classification of periodic arrivals in event time data for filtering computer network traffic. *Statistics and Computing 30*, 5 (Sep 2020), 1241–1254.

[25] Shahzad, H., Sattar, A. R., and Skandaraniyam, J. From real malicious domains to possible false positives in dga domain detection. In *2021 IEEE 13th International Conference on Computer Research and Development (ICCRD)* (2021), pp. 6–10.

[26] Shojaie, A., and Fox, E. B. Granger causality: A review and recent advances. *Annual Review of Statistics and Its Application 9*, 1 (2022), 289–319.

[27] Siraj, A., and Vaughn, R. B. Multi-level alert clustering for intrusion detection sensor data. In *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society* (2005), IEEE, pp. 748–753.

[28] Viinikka, J., and Debar, H. Monitoring ids background noise using ewma control charts and alert information. 166–187.

[29] Viinikka, J., Debar, H., Mé, L., and Séguier, R. Time series modeling for ids alert management. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2006), ASIACCS '06, Association for Computing Machinery, p. 102–113.

[30] Waita Njogu, H., and Jiawei, L. Using alert cluster to reduce ids alerts. In *2010 3rd International Conference on Computer Science and Information Technology* (2010), vol. 5, pp. 467–471.

[31] Wang, J., and Cui, B. Clustering ids alarms with an iga-based approach. In *2009 International Conference on Communications, Circuits and Systems* (2009), pp. 586–590.

[32] Wikipedia. Confusion matrix — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Confusion%20matrix&oldid=1148699071, 2023. [Online; accessed 29-April-2023].

[33] Wikipedia. Exponential smoothing — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Exponential%20smoothing&oldid=1144756953, 2023. [Online; accessed 19-April-2023].

[34] Wikipedia. Granger causality — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Granger%20causality&oldid=1150950996, 2023. [Online; accessed 24-April-2023].

[35] Wikipedia. Knowledge graph embedding — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Knowledge%20graph%20embedding&oldid=1131460653, 2023. [Online; accessed 10-March-2023].

[36] Wikipedia. Moving average — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Moving%20average&oldid=1150380012, 2023. [Online; accessed 19-April-2023].

[37] Wikipedia. Outlier — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Outlier&oldid=1144860671, 2023. [Online; accessed 23-April-2023].

[38] Wikipedia. Similarity measure — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Similarity%20measure&oldid=1152896701, 2023. [Online; accessed 07-May-2023].

[39] Wikipedia. Time series — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Time%20series&oldid=1150313391, 2023. [Online; accessed 19-April-2023].

[40] Xiao, S., Zhang, Y., Liu, X., and Gao, J. Alert fusion based on cluster and correlation analysis. In *2008 International Conference on Convergence and Hybrid Information Technology* (2008), pp. 163–168.

[41] Zhuang, X., Xiao, D., Liu, X., and Zhang, Y. Applying data fusion in collaborative alerts correlation. In *2008 International Symposium on Computer Science and Computational Technology* (2008), vol. 2, pp. 124–127.