UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

La mia rete è sicura?

Relatore:

Prof. Luca Deri

Candidato:

Diego Milletti

ANNO ACCADEMICO 2025/2026

Indice

1 Introduzione **7**
1.1 Obiettivo 8

2 Background e stato dell'arte **11**
2.1 Monitoraggio passivo e origine dei dati 11
2.2 Deep Packet Inspection e nDPI 12
2.2.1 Deep Packet Inspection 12
2.2.2 Architettura e classificazione di nDPI 13
2.2.3 Il protocollo TLS e il Server Name Indication 13
2.2.4 Indicatori di rischio (*flow risk*) 14
2.3 ntopng 15
2.3.1 Architettura e obiettivi 15
2.3.2 Analisi comportamentale e meccanismi di allarme 16
2.3.3 Punteggio nativo ed esportazione 16
2.4 ClickHouse 17
2.4.1 Funzioni del modello colonnare 17
2.4.2 Modello dei dati e primitive utilizzate 17
2.5 Lavori correlati 18
2.5.1 Valutazione passiva dei singoli host 18
2.5.2 Interpretabilità del punteggio 20
2.5.3 Posizionamento del lavoro 21

3 Architettura del sistema **23**
3.1 Principio operativo 23
3.2 L'equazione del rischio 24
3.3 Metriche deterministiche 24
3.4 Metriche statistico-comportamentali 25
3.4.1 Analisi dello scostamento statistico 25
3.4.2 Rilevamento delle novità 26
3.5 Classificazione dell'host 27
3.5.1 Derivazione delle soglie 27
3.6 Valutazione dell'intera rete 28

4 Selezione delle metriche **31**
4.1 Criteri di selezione . . . 31
4.2 Scenari di minaccia . . . 32
4.2.1 Ransomware e movimento laterale . . . 32
4.2.2 Esfiltrazione e tunneling . . . 32
4.2.3 Botnet e DGA . . . 33
4.3 L'insieme delle metriche adottato . . . 33
4.4 Metriche in dettaglio . . . 33
4.5 Elementi comuni . . . 34
4.5.1 Soggetti analizzati e filtraggio della rete locale . . . 34
4.5.2 Finestra di osservazione . . . 34
4.5.3 Origine dei segnali . . . 34
4.6 Metriche deterministiche . . . 34
4.6.1 M_{rep} – Destination reputation . . . 35
4.6.2 M_{cert} – TLS certificate anomalies . . . 35
4.6.3 M_{sni} – SNI evasion . . . 36
4.6.4 M_{proto} – Non-standard port/protocol . . . 36
4.6.5 M_{srv} – Server port detected . . . 37
4.6.6 M_{scan} – Network discovery . . . 37
4.7 Metriche statistico-comportamentali . . . 38
4.7.1 Caratteristiche comuni . . . 38
4.7.2 M_{vol} – Asimmetria volumetrica . . . 39
4.7.3 M_{fail} – Connection failure rate . . . 40
4.8 Calibrazione e riepilogo dei pesi . . . 41
4.8.1 Criteri di ordinamento . . . 41
4.8.2 Gruppi di gravità e vincoli . . . 42
4.8.3 Calibrazione . . . 42
4.8.4 Riepilogo . . . 43

5 Implementazione **45**
5.1 Sistema a tre livelli . . . 45
5.2 Acquisizione e persistenza dei dati . . . 46
5.3 Configurazione centralizzata . . . 47
5.4 Struttura modulare del sistema . . . 48
5.5 Realizzazione delle query . . . 49
5.6 Il motore di scoring . . . 52
5.7 Modello di esecuzione . . . 55

6 Validazione e risultati **57**
6.1 Metodologia di validazione . . . 57
6.2 Malware e botnet analizzati . . . 59
6.3 Metriche statistiche e baseline artificiale . . . 60
6.4 Host normali e controllo dei falsi positivi . . . 61
6.5 Inclusione di CIC-IDS2017 . . . 62

6.6 Matrice di confusione e metriche di performance 62
6.7 Limiti osservati 65
6.8 Costo computazionale del modello 65

7 Conclusioni **69**
7.1 Sviluppi futuri 70

A Codice sorgente e documentazione **71**

Capitolo 1

Introduzione

Il monitoraggio del traffico di rete [19] è diventato un'attività centrale nella gestione di qualunque infrastruttura informatica, poiché consente di comprenderne il comportamento e di garantirne sicurezza, affidabilità e prestazioni. Attraverso la raccolta e l'analisi dei dati di rete, le organizzazioni sono in grado di individuare malfunzionamenti, valutare l'utilizzo delle risorse e, soprattutto, rilevare attività malevole. L'aumento della quantità di traffico e la diffusione della cifratura rendono però sempre più difficile rispondere in modo sintetico alla domanda che ogni amministratore di rete si pone: *la mia rete è sicura?*

Una delle tecniche più diffuse per individuare anomalie nel traffico è l'utilizzo di sistemi IDS (Intrusion Detection System) [21], in grado di generare allarmi al verificarsi di accessi o comportamenti non autorizzati. Gli IDS si distinguono in due famiglie.

I sistemi basati su firme (*signature-based*) confrontano il traffico con pattern di attacco noti, per questo sono accurati su minacce catalogate, ma dipendono dalla frequenza di aggiornamento delle firme e risultano incapaci di rilevare minacce non ancora scoperte (*zero-days*).

I sistemi basati su anomalie (*anomaly-based*) definiscono invece un modello di ciò che è considerato normale per la rete e ne segnalano gli scostamenti, per questo riescono a intercettare attacchi sconosciuti, ma scontano un alto numero di falsi positivi. A seconda dell'infrastruttura osservata si distinguono IDS orientati alla rete (NIDS), all'host (HIDS) o ibridi.

Il sistema proposto in questo lavoro si configura come un IDS *network-based* e *anomaly-based* orientato al singolo host. La fonte dei dati è costituita esclusivamente dalla cattura passiva del traffico di rete, mentre la *baseline* comportamentale e la valutazione del rischio vengono modellate singolarmente per ciascun host.

Negli ultimi anni la necessità di individuare attività malevole che si nascondono in traffico formalmente lecito ha spinto verso l'adozione di tecniche di apprendimento automatico [3]. Questi approcci offrono una buona capacità di generalizzazione, ma presentano un limite rilevante [26]: i risultati prodotti sono spesso difficili da giustificare, e un allarme di cui non si comprende la causa ha un costo elevato per l'analista chiamato a intervenire.

Sul fronte degli strumenti, l'analisi passiva del traffico costituisce un approccio non invasivo che osserva i pacchetti in transito senza interferire con la comunicazione. Le tecniche di *Deep Packet Inspection* (DPI), come quelle offerte da nDPI [10], non si limitano a ispezionare l'intestazione dei pacchetti ma ne analizzano i metadati, permettendo di riconoscere protocolli e applicazioni e di generare segnali altrimenti non visibili.

Strumenti come ntopng [43], che integrano nDPI, sono in grado di calcolare indicatori di rischio e di generare allarmi a partire dal traffico osservato, mentre basi di dati analitiche (OLAP) come ClickHouse [35] consentono di archiviare ed interrogare in modo efficiente i flussi con i segnali raccolti. Questi strumenti producono segnali grezzi, ma non forniscono una risposta complessiva e facilmente leggibile sullo stato di sicurezza della rete.

1.1 Obiettivo

Questa tesi nasce con l'obiettivo di rispondere alla domanda: *la mia rete è sicura?*, definendo un modello matematico in grado di valutare lo stato di sicurezza di un'infrastruttura informatica attraverso la sola analisi passiva del traffico.

Il sistema assume come principio operativo che una rete sia sicura quando nessuno dei suoi host interni risulti compromesso o presenti comportamenti maliziosi rilevabili dall'analisi del traffico; i relativi limiti saranno discussi nel seguito. L'approccio proposto supera la tradizionale rilevazione delle sole minacce note, considerando anche i comportamenti anomali, ovvero il traffico apparentemente lecito ma non riconducibile al profilo abituale dell'host, che spesso accompagna un'attività malevola.

A ciascun host viene attribuito un punteggio di rischio, ottenuto combinando metriche deterministiche e statistico-comportamentali; un host viene segnalato non sulla base del singolo evento isolato, ma quando più indicatori anomali si sommano nel descriverne il comportamento. I punteggi dei singoli host vengono infine aggregati in una valutazione complessiva dello stato di sicurezza della rete.

La caratteristica distintiva del modello è che ogni penalità attribuita è riconducibile a un segnale concreto, così che la valutazione di ciascun host risulti sempre giustificabile. Il sistema è realizzato al di sopra di strumenti di analisi passiva del

traffico e di una base di dati analitica (OLAP), con un motore di calcolo che valuta periodicamente le metriche e ne aggrega i risultati.

Nei capitoli successivi verranno richiamati i concetti e gli strumenti su cui si fonda il lavoro (Capitolo 2); si descriverà il modello di valutazione del rischio e la classificazione degli host e della rete (Capitolo 3), per poi illustrare i criteri di selezione delle metriche e l'insieme effettivamente utilizzato con la descrizione dettagliata per ognuna (Capitolo 4), l'implementazione del sistema (Capitolo 5), i test effettuati per la validazione del sistema (Capitolo 6) e infine le conclusioni con gli sviluppi futuri (Capitolo 7).

Capitolo 2

Background e stato dell'arte

Il lavoro svolto si basa sull'integrazione dei tre strumenti già citati. Per comprendere le scelte progettuali descritte nei capitoli successivi è necessario avere una visione generale del funzionamento di questi componenti e dei concetti su cui si basano.

Il capitolo si apre con i principi del monitoraggio passivo e con l'origine dei dati su cui opera un analista (Sezione 2.1). Seguono i tre strumenti su cui il sistema è costruito, nell'ordine in cui i dati li attraversano: il motore di Deep Packet Inspection nDPI (Sezione 2.2), la sonda ntopng (Sezione 2.3) e il database colonnare ClickHouse (Sezione 2.4). La Sezione 2.5 esamina infine i lavori che hanno affrontato il medesimo problema, la valutazione passiva dei singoli host, e ne discute l'interpretabilità, collocando rispetto ad essi il contributo di questa tesi.

2.1 Monitoraggio passivo e origine dei dati

L'analisi del traffico di rete può essere condotta in modo *attivo* o *passivo*. Un approccio attivo introduce traffico nella rete per sollecitare i nodi, mentre un approccio passivo si limita a osservare i pacchetti in transito senza interferire con la comunicazione. Il monitoraggio passivo è preferibile in un contesto di sicurezza perché non altera il comportamento della rete osservata e non può essere scoperto da un eventuale attaccante.

Nell'attività di monitoraggio, i dati possono essere classificati in quattro categorie [8]. I *full content data* comprendono ogni informazione che è transitata in rete, payload compreso; sono completi ma comportano un costo di archiviazione e di elaborazione molto elevato, e soprattutto risultano in gran parte inutili in presenza di traffico cifrato. I *session data* (dati di flusso) descrivono in forma sintetica lo scambio di informazioni tra due host, riportandone indirizzi, porte, protocolli e metadati senza conservare l'intero contenuto. Gli *statistical data* aggregano il traffico in indicatori numerici, come il volume di byte scambiati o il numero di connessioni in una certa

finestra temporale. Gli *alert data*, infine, sono i segnali generati automaticamente da un sistema di rilevamento quando individua un evento di interesse.

Il sistema proposto opera esclusivamente su questi ultimi tre tipi di dati: ricostruisce il comportamento degli host a partire dai *session data*, ne misura gli scostamenti tramite gli indicatori statistici e utilizza gli allarmi prodotti dalla sonda. Rinuncia volutamente ai *full content data*, sia per ragioni di efficienza, sia perché la diffusione della cifratura ne ridurrebbe comunque l'utilità.

2.2 Deep Packet Inspection e nDPI

Tra i tre strumenti su cui si fonda il sistema, nDPI è quello che fornisce la maggior parte dei segnali a livello di singolo flusso. Per comprenderne il ruolo è necessario introdurre la tecnica su cui si basa, la *Deep Packet Inspection*, descriverne l'architettura interna e, infine, il meccanismo di valutazione del rischio che il modello proposto utilizza.

2.2.1 Deep Packet Inspection

La *Deep Packet Inspection* (DPI) [12] è una tecnica di analisi del traffico che non si limita a ispezionare l'intestazione dei pacchetti, ma ne esamina anche il *payload* alla ricerca di metadati significativi. Rispetto al semplice *packet sniffing*, che osserva i soli campi di intestazione, la DPI consente di riconoscere con precisione il protocollo applicativo e l'applicazione che ha generato un flusso, anche quando questi non corrispondono alla porta utilizzata, e di individuare condizioni importanti per la sicurezza, come violazioni di protocollo, malware o tentativi di esfiltrazione, che altrimenti non sarebbero visibili.

Affinché l'analisi sia possibile, il traffico deve essere reso disponibile allo strumento. Nelle reti commutate (*switched network*) ciò avviene tipicamente tramite *port mirroring* (o SPAN, *Switched Port Analyzer*), che replica il traffico di una o più porte di uno switch verso una porta di monitoraggio, oppure tramite *network tap*, dispositivi inseriti sul collegamento che ne duplicano il traffico verso il sistema di analisi. In entrambi i casi l'analisi rimane puramente passiva senza alterare la comunicazione.

La maggiore profondità di analisi ha però un costo. L'ispezione del contenuto richiede capacità di calcolo e memoria superiori rispetto alla sola lettura delle intestazioni [12], e solleva questioni di riservatezza, non approfondite in questo lavoro, legate all'esame del contenuto delle comunicazioni. A ciò si aggiunge la progressiva diffusione della cifratura, che rende il *payload* non ispezionabile e sposta il valore della DPI dall'analisi del contenuto a quella dei metadati osservabili, aspetto centrale per lo strumento utilizzato. Questa stessa limitazione ha favorito lo sviluppo di approcci di rilevazione basati sui flussi, che operano sui soli metadati della comunicazione anziché sul suo contenuto [37, 39].

2.2.2 Architettura e classificazione di nDPI

nDPI [10] è la libreria di DPI open-source utilizzata in questo lavoro, nata come evoluzione del precedente progetto OpenDPI. Offre il riconoscimento di un'ampia gamma di protocolli applicativi, l'analisi del traffico cifrato, l'estrazione di metadati dai protocolli supportati e l'attribuzione di indicatori di rischio ai flussi.

Internamente, la libreria opera sui flussi già suddivisi in base agli estremi della comunicazione (indirizzi, porte di sorgente e destinazione, protocollo di trasporto) secondo la definizione consolidata nel *flow monitoring* [17], e ne classifica il protocollo applicativo tramite un insieme di *dissector* specializzati, ciascuno registrato in corrispondenza di un protocollo e della relativa porta tipica. I pacchetti di un flusso non ancora classificato vengono sottoposti ai *dissector* a partire dal più probabile, così da ridurre il numero di tentativi; per ogni flusso viene mantenuto lo stato dei dissector già esclusi, e l'analisi si arresta non appena un protocollo viene riconosciuto, cosa che nella maggior parte dei casi avviene entro pochi pacchetti.

Particolarmente rilevante per questo lavoro è il trattamento del traffico cifrato. Anche quando il contenuto non è ispezionabile, infatti, nDPI estrae i metadati esposti durante l'*handshake*, aspetto approfondito nella sezione seguente.

2.2.3 Il protocollo TLS e il Server Name Indication

Il *Transport Layer Security* (TLS) è il protocollo che fornisce riservatezza, integrità e autenticazione del server alle comunicazioni su rete. Le versioni di riferimento sono la 1.2 [31] e la più recente 1.3 [30], che riduce la latenza della negoziazione e ne cifra una porzione maggiore. Prima che il canale cifrato venga stabilito, client e server svolgono una fase di *handshake* in cui concordano i parametri della sessione e il server si autentica.

È proprio in questa fase che risiede l'informazione utile al monitoraggio passivo. Sebbene il contenuto applicativo sia cifrato, alcuni campi dell'*handshake* transitano in chiaro e sono quindi osservabili senza alcuna decifratura. Per questo lavoro due elementi sono rilevanti:

- il nome del server richiesto dal client
- il certificato presentato dal server.

Il primo è trasmesso dall'estensione *Server Name Indication* (SNI), definita tra le estensioni TLS [1], tramite la quale il client inserisce nel messaggio iniziale, in chiaro, il nome dell'host che intende contattare, così che un server che ospita più domini su un unico indirizzo IP possa presentare il certificato corretto. Per un osservatore passivo lo SNI è determinante, poiché consente di associare un flusso cifrato al dominio di destinazione senza accedere al contenuto. I client legittimi lo includono normalmente, per cui la sua assenza costituisce un comportamento anomalo e può segnalare malware che cerca di sottrarsi all'identificazione basata sul nome dell'host.

Il secondo elemento è il certificato con cui il server si autentica. Diverse caratteristiche sono indicative di un rischio:

- un certificato autofirmato (non emesso da un'autorità di certificazione riconosciuta)
- scaduto (al di fuori della finestra di validità)
- firmato con algoritmi deboli o deprecati come SHA-1 (vulnerabile a collisioni).

L'osservabilità di questi campi dipende però dalla versione del protocollo. Nelle versioni fino alla 1.2 il certificato è scambiato in chiaro ed è quindi ispezionabile direttamente; nella 1.3, invece, la parte di *handshake* successiva al *ServerHello*, certificato incluso, viaggia cifrata, per cui gli indicatori basati sul certificato restano osservabili solo sugli *handshake* che lo espongono in chiaro. Per questa ragione alcune organizzazioni preferiscono mantenere versioni precedenti del protocollo, così da conservare una maggiore visibilità sul traffico degli utenti della rete. Lo SNI permane invece in chiaro nel messaggio iniziale del client in entrambe le versioni, con l'emergente *Encrypted ClientHello* [32] come eccezione futura. In sintesi, i campi in chiaro dell'handshake (SNI e certificato) sono esattamente ciò che nDPI estrae e traduce negli indicatori di rischio descritti nella sezione seguente.

2.2.4 Indicatori di rischio (*flow risk*)

Oltre alla classificazione, nDPI può essere impiegato per stabilire se un flusso presenti caratteristiche sospette [9]. A ciascun flusso la libreria associa un insieme di indicatori di rischio (*flow risk*), ognuno corredato da una descrizione testuale del problema rilevato (ad esempio protocollo noto su porta non standard o certificato TLS scaduto). Tali indicatori sono derivati da meccanismi diversi, tra cui le impronte dell'*handshake* TLS, la verifica delle firme e della validità dei certificati e l'analisi dell'entropia dei byte del flusso. A ogni rischio nDPI assegna una penalità, e la somma dei rischi rilevati lato client e lato server costituisce il punteggio complessivo del flusso.

Il sistema proposto non utilizza tale punteggio aggregato, bensì la presenza dei singoli indicatori, che reinterpreta come segnali logici da ripesare autonomamente (la motivazione è discussa nel Capitolo 5). Gli indicatori sono esposti in forma di *bitmask*, un campo numerico in cui ogni bit corrisponde a un rischio specifico ed interrogabile singolarmente. La Figura 2.1 riporta i bit di rischio di nDPI.

Vale infine la pena osservare che nDPI calcola anche le impronte (*fingerprint*) della comunicazione TLS, come JA3 [5] e la più recente JA4 [4], ottenute sintetizzando i parametri dell'*handshake* per riconoscere il software che genera una connessione. Confrontate con basi di firme note, tali impronte permettono di identificare comunicazioni malevole anche su traffico cifrato [6]. Il sistema non costruisce attualmente alcuna metrica su di esse; le ragioni di questa scelta, legate alla disponibilità delle

```
nDPI supported risks:
 Id Risk                                          Severity Score
  1 XSS attack                                    Severe   250
  2 SQL injection                                 Severe   250
  3 RCE injection                                 Severe   250
  4 Binary application transfer                   Severe   250
  5 Known protocol on non standard port           Low      10
  6 Self-signed Certificate                       Medium   50
  7 Obsolete TLS version (< 1.1)                  Medium   50
  8 Weak TLS cipher                               Medium   50
  9 TLS Expired Certificate                       High     100
 10 TLS Certificate Mismatch                      High     100
 11 HTTP Suspicious User-Agent                    Medium   50
 12 HTTP Numeric IP Address                       Low      10
 13 HTTP Suspicious URL                           High     100
 14 HTTP Suspicious Header                        Medium   50
 15 TLS (probably) not carrying HTTPS             Low      10
 16 Suspicious DGA domain name                    High     100
 17 Malformed packet                              Low      10
 18 SSH Obsolete Client Version/Cipher            Medium   50
 19 SSH Obsolete Server Version/Cipher            Medium   50
 20 SMB Insecure Version                          Medium   50
 21 TLS Suspicious ESNI Usage                     Medium   50
 22 Unsafe Protocol                               Low      10
 23 Suspicious DNS traffic                        Medium   50
 24 SNI TLS extension was missing                 Medium   50
 25 HTTP suspicious content                       Medium   50
 26 Risky ASN                                     Medium   50
 27 Risky domain name                             Medium   50
 28 Possibly Malicious JA3 Fingerprint            Medium   50
 29 Possibly Malicious SSL Cert. SHA1 Fingerprint Medium   50
 30 Desktop/File Sharing Session                  Low      10
 31 Uncommon TLS ALPN                             Medium   50
```

Figura 2.1: Bit di rischio nDPI

firme [23], sono discusse nel Capitolo 4, invece i possibili sviluppi sono discussi nel Capitolo 7.

2.3 ntopng

ntopng [43, 11] è uno strumento open-source per il monitoraggio passivo del traffico, sviluppato dal progetto ntop e costruito al di sopra di nDPI, da cui eredita le capacità di classificazione e di analisi dei flussi cifrati descritte nella sezione precedente.

2.3.1 Architettura e obiettivi

ntopng nasce per superare il limite dei modelli basati su flussi esportati (NetFlow, IPFIX, sFlow), ovvero la loro intrinseca incapacità di fornire una visione in tempo reale, dovuta alla latenza di esportazione e al fatto che le statistiche di flusso riportano valori medi che nascondono i picchi di traffico. Per questo ntopng è realizzato interamente in software, senza dipendenze da hardware specializzato, ed elabora il traffico aggiornando i contatori in tempo reale.

L'architettura è organizzata su più livelli [11]. Un livello di ingresso acquisisce i dati, che possono essere pacchetti grezzi catturati da una o più interfacce di rete oppure flussi raccolti da sonde esterne. Un motore di monitoraggio ne costituisce il nucleo e li consolida organizzandoli in due tipologie: i flussi e gli host. Un flusso è

l'insieme dei pacchetti che condividono la stessa tupla identificativa della comunicazione (VLAN, protocollo, indirizzi, porte di sorgente e destinazione), e ogni flusso fa riferimento a due host, sorgente e destinazione, dei quali mantiene le statistiche. Questa organizzazione, in cui ciascun host accumula nel tempo i contatori del proprio traffico, è il presupposto su cui si fonda la valutazione per host adottata in questo lavoro.

Al di sopra del motore opera un sistema di scripting in linguaggio Lua, che disaccoppia l'accesso ai dati dal trattamento del traffico ed esegue periodicamente attività di consolidamento. Già nella formulazione originale ntopng usava questo meccanismo per riversare periodicamente su una base di dati lo storico degli host osservati, realizzando una vista persistente dell'attività recente; nel sistema qui descritto questo ruolo è svolto da ClickHouse, come illustrato nella sezione seguente.

2.3.2 Analisi comportamentale e meccanismi di allarme

Osservare il comportamento abituale di una singola entità e segnalare quando se ne discosta è l'idea alla base della *User and Entity Behavior Analytics* (UEBA) [20], un approccio che non cerca la firma di un attacco noto ma i cambiamenti nel profilo dell'entità osservata. Il modello proposto segue questa logica applicandola all'host di rete, e usa ntopng per costruire tali profili.

Fin dalla sua progettazione ntopng non si limita a classificare il protocollo, ma caratterizza il traffico in base alla sua natura e costruisce profili di comportamento degli host, così da poter segnalare i casi in cui un host modifica nel tempo il proprio profilo di traffico, eventualità che può indicare la presenza di software malevolo o indesiderato. L'attribuzione di un punteggio di sicurezza agli host e l'integrazione di servizi di reputazione degli indirizzi, indicate come sviluppi futuri già nella formulazione originale [11], sono state in seguito incorporate nello strumento e costituiscono parte dei segnali che il sistema qui descritto utilizza.

Nelle versioni adottate, quando un controllo individua una condizione anomala ntopng genera un allarme, distinguendo tra allarmi di flusso e allarmi di host. Alcuni controlli, come il rilevamento di un host che inizia a contattare una porta server non prevista, richiedono un *learning period* durante il quale viene costruito il profilo di riferimento, così da non segnalare come anomalo un comportamento semplicemente nuovo. Gli host, inoltre, sono distinti in locali e remoti sulla base delle reti dichiarate come interne, e questa classificazione determina quali controlli vengono applicati; in particolare l'attenzione è rivolta agli host locali, dei quali si vuole controllare il comportamento.

2.3.3 Punteggio nativo ed esportazione

A ogni evento anomalo ntopng associa un punteggio, secondo un approccio cumulativo e privo di un limite superiore, ovvero l'indicatore complessivo di un host può

così crescere all'infinito al ripetersi degli eventi. Come si discuterà nel Capitolo 5, questa caratteristica è incompatibile con la scala limitata adottata dal modello proposto, ragione per cui gli allarmi nativi vengono reinterpretati come semplici segnali logici e ripesati da un motore di calcolo esterno. Coerentemente con il meccanismo di esportazione dello storico previsto fin dalla progettazione originale [11], le versioni adottate esportano flussi e allarmi verso ClickHouse, il cui ruolo è descritto nella sezione seguente.

2.4 ClickHouse

ClickHouse [35] è un sistema di gestione di basi di dati colonnare, orientato all'analisi (*OLAP*) e progettato per interrogazioni ad alte prestazioni su tabelle con un numero molto elevato di righe. Nel sistema proposto svolge il ruolo di memoria storica: vi confluiscono i flussi e gli allarmi prodotti da ntopng, sui quali il motore di scoring calcola periodicamente le proprie metriche.

2.4.1 Funzioni del modello colonnare

A differenza dei database relazionali tradizionali, che memorizzano i record riga per riga, ClickHouse organizza i dati per colonna, archiviando in modo contiguo i valori di uno stesso campo [2]. Da questa scelta derivano i vantaggi che lo rendono adatto ai carichi analitici. Una query che coinvolge poche colonne legge dal disco soltanto i dati di quelle colonne, anziché l'intera riga, riducendo drasticamente il volume di dati spostato in memoria; la contiguità di valori dello stesso tipo rende inoltre la compressione molto efficace, e le operazioni di aggregazione possono essere eseguite in modo vettoriale su interi blocchi di valori anziché elemento per elemento.

Sul piano dell'archiviazione, ClickHouse adotta motori di tabella della famiglia *MergeTree*, ispirati alle strutture *log-structured merge-tree* (LSM): i dati sono suddivisi in parti ordinate e immutabili, fuse periodicamente da un processo in background, e l'esecuzione delle query si avvale di tecniche di *pruning* che evitano di leggere le parti non pertinenti al filtro. Rispetto a un database orientato alle righe, questo si traduce in tempi di risposta nettamente inferiori sulle interrogazioni di tipo analitico [41].

2.4.2 Modello dei dati e primitive utilizzate

ntopng offre un'integrazione nativa con ClickHouse per l'esportazione dei flussi e degli allarmi, che vengono così resi disponibili nello storico interrogato periodicamente dal motore di scoring tramite query SQL. I dati sono organizzati principalmente in due tabelle, una per i flussi (*flows*) e una per gli allarmi di host (*host_ alerts*), in cui ogni record conserva, tra gli altri campi, gli estremi della comunicazione, i contatori di traffico e, per i flussi, la *bitmask* dei rischi nDPI descritta in precedenza.

Per calcolare le metriche presentate nei capitoli successivi, il motore di scoring interroga questi dati usando le funzioni SQL offerte da ClickHouse. Grazie a esse raggruppa gli eventi per intervalli di un'ora (bucket orari), li aggrega per host, controlla i singoli bit della bitmask dei rischi, estrae i campi annidati degli allarmi in formato JSON e conta gli eventi e i valori distinti.

2.5 Lavori correlati

Il sistema proposto si colloca tra gli strumenti di rilevamento delle intrusioni basati sui flussi (*flow-based intrusion detection*), in cui l'analisi si fonda sui metadati delle comunicazioni (indirizzi, porte, protocolli, contatori) anziché sul contenuto completo dei pacchetti. Sperotto et al. [37] ne forniscono una panoramica generale, mentre Umer et al. [39] ne discutono tecniche e limiti aperti. L'approccio conserva efficacia anche in presenza di traffico cifrato ed è sostenibile su volumi di traffico elevati, le stesse esigenze che motivano il presente lavoro.

All'interno di questa famiglia, le sezioni seguenti ripercorrono i sistemi che valutano lo stato di salute di una rete a partire dal comportamento osservato dei suoi host interni. Segue una discussione sull'interpretabilità del risultato e sul posizionamento di questa tesi.

2.5.1 Valutazione passiva dei singoli host

Il primo sistema a costruire una valutazione per singolo host accumulando segnali indipendenti è BotHunter [16]. L'osservazione di partenza è che un'infezione da bot non è un evento isolato, ma una comunicazione con una struttura ricorrente: l'host viene scandito dall'esterno, subisce lo sfruttamento di una vulnerabilità, scarica il file binario del malware, contatta il server di comando e controllo (C2) e infine comincia a scandire l'esterno a sua volta. Tre sensori passivi osservano il traffico al perimetro e producono cinque tipi di segnali corrispondenti a queste fasi. Un motore di correlazione mantiene una matrice con una riga per ogni host interno, vi accumula i segnali osservati in una finestra temporale e dichiara l'host infetto quando la loro combinazione supera una soglia. Il sistema è stato valutato su 2 019 infezioni reali, riconoscendone 1 920, e su una rete universitaria in esercizio per quattro mesi, dove ha prodotto meno di un falso positivo al giorno.

Due scelte di BotHunter anticipano il modello descritto in questa tesi. La prima è la condizione di allarme, poiché nessun segnale preso da solo è sufficiente: occorre l'evidenza di un'infezione locale accompagnata da almeno un segnale in uscita, oppure almeno due segnali distinti di attività in uscita. È il principio per cui più evidenze indipendenti, ciascuna debole se isolata, si rafforzano a vicenda. La seconda è la distribuzione dei pesi, stimati dagli autori con un modello di regressione: la scansione in ingresso pesa 0,094, mentre il download del file binario, il traffico verso il server di comando (C2) e la scansione in uscita pesano 0,344 ciascuno. I segnali genera-

ti dall'host interno valgono quasi quattro volte quelli che l'host subisce, la stessa asimmetria che nel presente lavoro porta a penalizzare soltanto l'host che origina il traffico anomalo. Le differenze restano però nette. BotHunter richiede sensori che ispezionano il payload per riconoscere gli *exploit* e le firme del file binario scaricato, quindi perde gran parte della propria efficacia sul traffico cifrato, e il suo esito è binario; l'host è infetto oppure non lo è, senza una gradazione del sospetto.

Beehive [42] sposta l'attenzione dalle firme al comportamento abituale. Per ciascun host di una rete aziendale viene calcolato ogni giorno un vettore di quindici caratteristiche, raggruppate in quattro famiglie: destinazioni esterne nuove o poco frequentate, cambiamenti nella configurazione software dedotti dalle stringhe *user-agent*, violazioni delle politiche aziendali e picchi di traffico. I vettori vengono ridotti con l'analisi delle componenti principali e raggruppati con una variante di *k-means*, e gli host che finiscono nei gruppi periferici sono segnalati come incidenti all'analista. Nella valutazione, condotta su due settimane di log di una grande azienda, Beehive ha prodotto 784 segnalazioni, di cui soltanto otto già note agli strumenti di sicurezza in uso. Alcune sue caratteristiche corrispondono per intento alle metriche qui adottate, dalle destinazioni mai contattate prima ai picchi di connessioni fino ai domini bloccati dal proxy. Beehive però non osserva il traffico, bensì i registri prodotti dai diversi servizi aziendali, in primo luogo il proxy web e i controller di dominio. Ha quindi bisogno di vedere il contenuto delle richieste HTTP e le autenticazioni degli utenti, mentre il sistema proposto si limita ai metadati dei flussi. Soprattutto, il suo esito non è un punteggio: un host viene segnalato quando finisce fuori dal gruppo, e la sua posizione dipende dalla popolazione osservata quel giorno.

Il lavoro più vicino al presente è quello di Marchetti et al. [22], che definiscono esplicitamente un punteggio di sospetto per ogni host interno a partire dai soli record di flusso, raccolti nel loro prototipo con la sonda nprobe. Vengono considerati soltanto i flussi uscenti iniziati dagli host interni, sulla base dell'osservazione che in una rete protetta da firewall le comunicazioni malevole partono dall'interno. Il comportamento di ogni host viene confrontato con una *baseline* costruita sulle proprie osservazioni recenti, su una finestra di quattordici giorni. Da tre caratteristiche (byte caricati, flussi iniziati, destinazioni distinte contattate) vengono derivati tre punteggi parziali, combinati poi in una somma pesata. Il sistema è stato valutato su una rete reale di circa 10 000 host iniettando esfiltrazioni artificiali di dimensioni comprese tra 50 MB e 40 GB, con una tecnica di iniezione controllata analoga, nello spirito, a quella impiegata nella Sezione 6.3 per esercitare le metriche statistiche.

Nonostante la forma additiva comune, il punteggio di Marchetti è di natura diversa da $S(h)$. È illimitato, dipende dalla popolazione osservata, poiché sia il centroide di riferimento sia i pesi vengono ricalcolati ogni giorno sull'insieme degli host attivi, e serve a ordinare gli host anziché a classificarli. L'output è infatti una classifica in cui l'analista esamina i primi k elementi, per cui un punteggio elevato significa soltanto che l'host è il più sospetto tra quelli osservati in quel giorno, e due giorni diversi non sono confrontabili tra loro. I tre punteggi parziali, inoltre, sono distanze

calcolate nello spazio definito dalle tre caratteristiche: dicono di quanto e in quale direzione l'host si sia allontanato dalla propria posizione abituale, ma non quale comportamento anomalo abbia tenuto.

Kitsune [25] rappresenta infine l'approccio non supervisionato applicato al traffico grezzo. Per ogni pacchetto in transito il sistema aggiorna 115 statistiche incrementali su finestre temporali smorzate, aggregate per indirizzo sorgente, canale e socket, e le fornisce a un insieme di *autoencoder* di piccole dimensioni. Ciascun *autoencoder* tenta di ricostruire la porzione di caratteristiche di sua competenza, e l'errore di ricostruzione viene aggregato in un punteggio di anomalia. Gli autori motivano la rinuncia all'apprendimento supervisionato con le stesse ragioni addotte in questa tesi, poiché etichettare il traffico è costoso, ciò che è normale dipende dalla singola rete e un classificatore riconosce soltanto le classi che gli sono state mostrate. Il sistema è stato valutato su nove attacchi in una rete di videosorveglianza e in una rete IoT, incluso il malware Mirai, con prestazioni superiori a quelle di un rilevatore basato su firme.

Il confronto con Kitsune è importante proprio perché il sistema condivide con questa tesi la natura passiva, l'indipendenza dal payload e l'assenza di etichette. Ciò che lo distingue è la forma del risultato. Il punteggio di Kitsune è un errore di ricostruzione, un numero non limitato superiormente il cui valore dipende dalla specifica rete su cui il modello è stato addestrato, e non è scomponibile in alcuna causa leggibile, poiché nasce dall'interazione di un insieme di *autoencoder*. Il suo soggetto, inoltre, è il singolo pacchetto e non l'host, per cui il sistema produce un flusso continuo di valori da confrontare con una soglia, non una valutazione dello stato di una macchina. Nella parte finale dell'articolo gli autori osservano inoltre che, poiché durante l'addestramento tutto il traffico viene assunto come benigno, un avversario già presente nella rete al momento dell'installazione entra a far parte del modello di normalità e sfugge al rilevamento. È lo stesso limite di contaminazione della *baseline* discusso nella Sezione 6.7.

Va infine ricordato che l'idea di assegnare un punteggio di rischio al singolo host è già presente nella sonda ntopng [11], che calcola un proprio score cumulativo. Il presente lavoro lo reinterpreta su una scala fissa e limitata, ricalcolandolo esternamente a partire dai segnali esportati, come descritto nella Sezione 3.2.

2.5.2 Interpretabilità del punteggio

Il tratto che distingue maggiormente il sistema proposto riguarda l'interpretabilità del risultato. Diversi articoli recenti osservano che gli IDS basati su *machine learning*, pur essendo accurati, si comportano come una *black box* i cui allarmi risultano difficili da giustificare. Neupane et al. [29] distinguono esplicitamente i modelli interpretabili (white-box) da quelli complessi resi spiegabili tramite tecniche come SHAP e LIME (black-box), e Mohale e Obagbuwa [26] esaminano l'integrazione di queste tecniche di *explainable AI* negli IDS. Il sistema descritto in questa tesi appar-

tiene alla prima categoria, poiché il punteggio $S(h)$ è la somma esplicita di segnali concreti e resta quindi scomponibile nelle metriche che lo hanno determinato, senza necessità di un modello esterno che ne interpreti l'output. Anche i lavori orientati alla sola accuratezza nel rilevamento di botnet e attacchi IoT raggiungono risultati elevati [40, 24] ma condividono la dipendenza da una fase di addestramento e la difficoltà di ricondurre un allarme a una causa comprensibile per l'analista.

Il confronto con Kitsune mostra però che rinunciare all'addestramento supervisionato non basta a rendere interpretabile un sistema. Kitsune non ha bisogno che qualcuno gli mostri esempi di attacco già riconosciuti come tali, eppure il suo punteggio resta opaco, perché è l'errore di ricostruzione di un insieme di reti neurali. La stessa osservazione vale, in senso opposto, per i punteggi parziali di Marchetti et al. [22], che sono perfettamente calcolabili a mano ma dicono che l'host si è spostato molto e in una direzione insolita, non che ha effettuato una scansione o presentato un certificato anomalo.

La differenza non è dunque tra un modello trasparente e uno opaco. Tutti questi sistemi producono un valore scomponibile nei contributi che lo hanno formato: ciò che cambia è la natura di tali contributi. In Marchetti et al. [22] sono distanze, in Kitsune sono errori di ricostruzione, mentre in questo lavoro ciascun contributo corrisponde a un comportamento con un nome preciso, come una scansione o un certificato anomalo, e quindi a una verifica che l'analista può ripetere sui dati grezzi.

2.5.3 Posizionamento del lavoro

Sul piano delle singole tecniche, il sistema eredita dalla letteratura più di un elemento. L'analisi dei metadati TLS come sostituto dell'ispezione del contenuto [6] e l'osservazione dei domini generati algoritmicamente [7] motivano rispettivamente alcune metriche adottate dal sistema, mentre le impronte del client TLS JA3 e JA4 [5, 4] indicano una direzione futura. L'uso di stimatori robusti come la *Median Absolute Deviation* per il rilevamento degli scostamenti trova riscontro in Romo-Chavero et al. [33]. Queste tecniche sono riprese nei Capitoli 4 e 7, dove motivano le singole metriche.

Il contributo di questa tesi non sta dunque nell'idea di valutare un host accumulando segnali indipendenti, già presente in [16], né nella costruzione del profilo comportamentale per host, introdotta da [42], né nell'uso dei soli metadati di flusso, comune a [22], né nel fare a meno di esempi di attacco già classificati, come avviene in [25]. Sta nella combinazione di tre proprietà che nessuno dei lavori esaminati presenta contemporaneamente.

La prima è che il punteggio è assoluto e non relativo. Il punteggio $S(h)$ vive su una scala fissa da 0 a 100 con soglie prefissate, per cui due host, due reti e due giorni diversi sono confrontabili tra loro, e una rete in cui nessun host è compromesso non produce comunque un primo classificato. I sistemi basati sul ranking [22] o

sul raggruppamento [42] misurano invece una posizione all'interno della popolazione osservata, mentre il punteggio di [25] non ha un limite superiore e assume significato soltanto rispetto alla rete su cui è stato addestrato.

La seconda è la scomposizione del punteggio. Ogni addendo nomina un comportamento, come una scansione, un certificato anomalo o una porta non standard, e corrisponde quindi a un controllo che l'analista può rifare direttamente sui flussi. Il punteggio non indica soltanto quanto un host sia sospetto, ma per quale ragione lo sia.

La terza è che il soggetto della valutazione è l'host e l'unità di analisi è l'ora. Al termine di ogni ciclo il sistema restituisce, per ciascuna macchina, un punteggio e la fascia di rischio corrispondente, e per aggregazione una valutazione dell'intera rete. Non produce quindi un allarme sul singolo pacchetto, come in [25], né una classifica dei sospetti del giorno, come in [22].

Restano visibili anche i limiti che questo posizionamento comporta, e la letteratura esaminata aiuta a inquadrarli. Un canale di comando cifrato, a bassa intensità e diluito nel tempo, non attiva alcuna metrica: è il caso delle due catture Torii descritte nella Sezione 6.7, ed è la stessa evasione che gli autori di [16] indicano tra i limiti dichiarati del proprio sistema. Il confronto con la sola storia dell'host, inoltre, rende il modello cieco davanti a un host compromesso fin dall'inizio dell'osservazione, limite che gli autori di [25] riconoscono negli stessi termini per il proprio modello di normalità. Non si tratta quindi di difetti accidentali dell'implementazione, ma di confini strutturali dei sistemi che apprendono il comportamento normale dal traffico osservato. Il confronto tra host, praticato sia da [42] sia da [22], costituisce l'estensione indicata tra gli sviluppi futuri nella Sezione 7.1.

Capitolo 3

Architettura del sistema

Questo capitolo descrive la struttura alla base del sistema, indipendentemente dalle singole metriche e dai dettagli implementativi, trattati nei capitoli successivi.

3.1 Principio operativo

Il modello assume come principio operativo che una rete sia sicura quando nessuno dei suoi host interni risulta compromesso o presenta comportamenti maliziosi rilevabili dall'analisi del traffico. Si tratta di una condizione necessaria ma non sufficiente; ovvero, il qualificatore "rilevabili dall'analisi del traffico" delimita esplicitamente il perimetro del modello, i cui limiti saranno discussi nel Capitolo 7.

La scelta dell'host come unità di analisi non è casuale. L'host è l'entità che viene effettivamente compromessa e su cui l'analista può intervenire. È inoltre la granularità corretta tra due estremi: il singolo flusso è troppo effimero per sostenere un giudizio, poiché un flusso anomalo isolato può essere del tutto legittimo, mentre la rete nel suo complesso è troppo vasta per localizzare il problema. La valutazione per host trova infine un supporto negli strumenti adottati, poiché ntopng accumula già contatori e profili per ciascun host (Sezione 2.3.1).

Anziché limitarsi a verificare la presenza di minacce note, ovvero un giudizio binario di tipo legittimo/malevolo, il modello misura quanto il comportamento di ciascun host si discosta dal proprio profilo abituale (*baseline*), attribuendogli un punteggio di rischio graduale. In questo modo è possibile prendere in considerazione comportamenti che sembrano leciti ma anomali, che frequentemente accompagnano un'attività malevola e che un approccio basato esclusivamente su firme non rileverebbe.

Da questo principio discendono tre criteri che guidano l'intera progettazione descritta nel resto del capitolo. Il primo è l'interpretabilità: ogni giudizio deve essere riconducibile a una causa concreta e osservabile, così che l'analista possa sempre ve-

rificarlo. Il secondo è il contenimento dei falsi positivi: un host non viene segnalato per un singolo indizio, ma solo quando più evidenze indipendenti si sommano. Il terzo è la copertura di ciò che è noto e di ciò che non lo è, poiché un sistema che riconosca soltanto minacce catalogate resterebbe cieco di fronte a comportamenti mai visti. Le scelte strutturali che seguono, dall'analisi del singolo host alla forma del punteggio, sono basate su questi tre criteri.

3.2 L'equazione del rischio

A ciascun host h viene associato, in un dato istante, uno *score* che ne quantifica il grado di sospetto. Il punteggio è ottenuto come somma delle penalità attribuite da ogni metrica per le anomalie rilevate, limitata superiormente a un valore massimo:

$$S(h) = \min\left(100,\ \sum_{i=1}^{n} p_i \cdot M_i(h)\right)$$

dove p_i è la penalità associata alla i-esima metrica e $M_i \in \{0, 1\}$ è il valore booleano da essa restituito: vale 1 quando la metrica rileva l'anomalia, 0 altrimenti. Il limite superiore a 100 mantiene il punteggio su una scala fissa e interpretabile, indipendentemente dal numero di metriche attive, e costituisce una delle differenze rispetto al punteggio cumulativo e illimitato adottato nativamente da ntopng.

La scelta di sommare le penalità, anziché combinarle in altro modo, è la traduzione matematica del principio per cui più indizi indipendenti si rafforzano a vicenda. Le anomalie rilevate dalle diverse metriche sono per costruzione segnali indipendenti, ciascuno debole se preso da solo, dato che un certificato anomalo, una porta non standard o una singola scansione possono avere spiegazioni legittime. Sommandoli, il modello fa sì che nessuno di essi da solo determini un allarme, ma che la loro coincidenza sullo stesso host ne rafforzi il sospetto fino a superare la soglia. Questa forma additiva è preferita a un'alternativa più semplice, il massimo: due segnali distinti da venticinque punti descrivono un host più sospetto di uno solo, ma il massimo li tratterebbe come equivalenti.

Ogni penalità p_i è definita in modo esplicito e riconducibile a un segnale concreto. È proprio questa caratteristica a rendere il modello trasparente e verificabile, poiché il punteggio di un host può sempre essere scomposto nelle metriche che lo hanno determinato.

3.3 Metriche deterministiche

Le metriche deterministiche valutano eventi e segnali confrontando in tempo reale l'osservazione corrente con un insieme di dati noti. Il risultato è binario:

$$M_i(h) = \begin{cases} 1 & \text{in presenza di corrispondenza} \\ 0 & \text{in assenza di corrispondenza} \end{cases}$$

Rientrano in questa classe, ad esempio, metriche che verificano la presenza di un host o di una destinazione all'interno di insiemi noti, o che recepiscono un allarme già prodotto dalla sonda. Dato che si tratta di un confronto diretto, questo insieme di metriche non richiede la costruzione di un profilo storico (*baseline*) e ha un costo computazionale ridotto.

La presenza di questa classe non è però sufficiente. Per costruzione, una metrica deterministica riconosce solo ciò che è già noto, e resta quindi cieca di fronte a un comportamento malevolo mai visto prima, esattamente il limite dei sistemi basati su firme richiamato nel Capitolo 1. Per coprire anche comportamenti sconosciuti il modello affianca alle deterministiche una seconda classe, le metriche statistico-comportamentali, che non confrontano l'osservazione con minacce catalogate ma con il profilo abituale dell'host stesso, segnalando gli scostamenti. Le due classi corrispondono così alle due famiglie di IDS descritte nell'introduzione, quella basata su firme e quella basata su anomalie. La scelta di usarle insieme risponde ai limiti reciproci. Le deterministiche sono precise ed economiche ma non generalizzano oltre ciò che conoscono; le statistiche generalizzano al nuovo ma, prese da sole, sono soggette a un numero maggiore di falsi positivi, poiché ogni comportamento inatteso ma legittimo può apparire anomalo. Utilizzandole all'interno dello stesso punteggio additivo, il modello ottiene la copertura delle une senza subire per intero il tasso di errore delle altre, dato che un'anomalia statistica isolata raramente basta a superare la soglia e un host viene segnalato solo quando l'anomalia trova conferma in altri segnali. Questa complementarità ha anche un risvolto di costo, poiché le deterministiche leggono un segnale già pronto mentre le statistiche devono ricostruire la *baseline* dell'host, differenza che si ritroverà nella misura dei tempi di esecuzione (Sezione 6.8).

3.4 Metriche statistico-comportamentali

Queste metriche individuano i comportamenti anomali rispetto alla baseline dell'host specifico, come il traffico eccessivo, un'attività fuori dal solito orario o l'utilizzo di servizi mai osservati prima. Il modello proposto affronta questo problema con due approcci distinti.

3.4.1 Analisi dello scostamento statistico

Per grandezze continue, come il volume di byte scambiati o la frequenza delle connessioni, è necessario definire cosa costituisca un comportamento normale per un dato host. A tal fine, si considera una finestra temporale di riferimento e se ne riassume il comportamento mediante due indicatori statistici robusti: la **mediana** $\tilde{x}$,

che rappresenta il valore tipico della grandezza osservata, e la **Median Absolute Deviation** (MAD), che misura la dispersione ed è definita come la mediana degli scarti assoluti dalla mediana:

$$MAD = \text{Mediana}\big(\,|x_i - \tilde{x}|\,\big)$$

La scelta di mediana e MAD, in luogo di media e deviazione standard, è motivata dalla loro robustezza e dalla validazione nella rilevazione delle anomalie di rete [33]. A differenza degli stimatori classici, esse sono poco influenzate dai valori estremi e non vengono distorte da picchi di traffico legittimi e occasionali, fornendo risultati più affidabili sui comportamenti ordinari degli host.

Lo scostamento di un'osservazione corrente x dal comportamento atteso è quindi misurato tramite un **Modified z-score** [13], ottenuto rapportando la distanza dalla mediana alla dispersione:

$$Z_{modified} = \frac{|x - \tilde{x}|}{MAD}$$

La definizione classica del *modified z-score* prevede di moltiplicare il rapporto per una costante fissa (0,6745). Questa costante serve a far sì che la MAD possa essere letta come una normale deviazione standard, ma solo quando i dati seguono una distribuzione normale. Nel sistema proposto la costante non viene utilizzata, come del resto in [13], perché la soglia di anomalia non viene ricavata dalla distribuzione normale, ma fissata in base ai risultati dei test (Capitolo 6). Moltiplicare tutti i valori per uno stesso numero sposterebbe semplicemente la soglia della stessa quantità, senza cambiare quali host vengono segnalati. Per questo motivo la costante viene omessa. Nella formula attuale è presente il valore assoluto, cosa che non ritroveremo nella formula finale e motivata durante la descrizione delle metriche (Capitolo 4). Un valore di $Z_{modified}$ superiore a una soglia prestabilita segnala un'anomalia. Tale limite è stato impostato a 3, un valore confermato empiricamente durante la fase di test (Capitolo 6).

3.4.2 Rilevamento delle novità

Alcuni eventi, come l'utilizzo di un protocollo applicativo o di un servizio mai osservato in precedenza per quell'host, sono strutturalmente nuovi e non sono adatti per un confronto numerico.

Per individuarli si ricorre alla teoria degli insiemi applicata ai dati estratti tramite *Deep Packet Inspection*. Si definisce l'insieme $P_{storico}$ dei servizi abitualmente utilizzati dall'host e l'insieme P_{oggi} dei servizi osservati nella finestra temporale corrente; la novità N è data dalla loro differenza insiemistica:

$$N = P_{oggi} \setminus P_{storico}$$

Se l'insieme N non è vuoto, l'host si è comportato in modo diverso rispetto al proprio profilo storico e la corrispondente metrica di anomalia viene attivata.

3.5 Classificazione dell'host

Il modello non trae conclusioni dal valore di una singola metrica isolata, ma dalla loro combinazione. Questa scelta risponde direttamente al problema dei falsi positivi; infatti, un singolo evento anomalo, come ad esempio una scansione della rete interna, può essere condotto da un amministratore di sistema e non è di per sé indicativo di una compromissione.

Un host è pertanto considerato sospetto soltanto quando più anomalie si sovrappongono, facendo crescere la somma $S(h)$ delle penalità oltre una soglia predefinita, nella cosiddetta fascia rossa. In base al valore di $S(h)$ ogni host è collocato in una delle tre fasce di rischio:

- **verde**: punteggi inferiori alla prima soglia di 30 punti
- **gialla**: valori intermedi, fino a 59 punti
- **rossa**: al superamento della soglia limite della fascia gialla.

L'adozione di tre fasce, anziché di un giudizio binario o del solo punteggio numerico, risponde a un'esigenza operativa. Le tre fasce corrispondono a tre azioni distinte per l'analista: ignorare (verde), sorvegliare (gialla) e intervenire (rossa). Una classificazione binaria collasserebbe proprio lo stato intermedio, quello in cui un host è sospetto ma non ancora confermato e in cui i segnali si stanno ancora accumulando. Il punteggio $S(h)$ resta comunque disponibile nel report, per cui il giudizio non nasconde l'informazione, ma vi sovrappone soltanto uno strato di decisione azionabile.

3.5.1 Derivazione delle soglie

I valori 30 e 60 non sono scelti in modo arbitrario, né sono indipendenti dalle penalità p_i associate alle singole metriche. Soglie e penalità formano un unico sistema, vincolato da tre requisiti che discendono dai criteri enunciati nella Sezione 3.1.

Il criterio del contenimento dei falsi positivi impone che nessuna metrica, da sola, possa collocare un host in fascia rossa. La soglia della fascia rossa deve quindi essere maggiore della penalità più alta attribuita a una singola metrica, che nella configurazione predefinita vale 50 punti (Tabella 4.2). Il valore adottato è 60. Un host raggiunge la fascia rossa soltanto attraverso il concorso di almeno due segnali

indipendenti, il che è precisamente la traduzione operativa della forma additiva discussa nella Sezione 3.2.

Le metriche si dividono in due gruppi per la gravità del comportamento che descrivono. Il primo comprende i segnali che rivelano un'attività sospetta originata dall'host, il secondo gli indizi deboli e le esposizioni subite, che su una rete reale ricorrono anche in assenza di compromissione. La soglia della fascia gialla è posta in modo da separare esattamente i due gruppi: deve superare la penalità più alta del secondo gruppo, pari a 25 punti, e non superare la penalità più bassa del primo, pari a 30 punti. I valori ammissibili sono quindi compresi tra 26 e 30; si adotta 30, che fa coincidere la soglia con la penalità più bassa del primo gruppo. Ogni metrica del primo gruppo, se attiva da sola, colloca l'host in fascia gialla e ne impone la sorveglianza; ogni metrica del secondo, se attiva da sola, lascia l'host in fascia verde.

Il terzo requisito impone che l'accumulo di segnali deboli possa comunque raggiungere la fascia rossa. È il requisito opposto al primo: se le soglie fossero troppo alte, la somma di più segnali deboli non raggiungerebbe mai la fascia rossa e la forma additiva perderebbe significato. Con la configurazione adottata, tre metriche del secondo gruppo attive contemporaneamente producono $25 + 20 + 20 = 65$ punti e collocano l'host in fascia rossa, mentre le due metriche statistico-comportamentali, da sole, si fermano a $20 + 30 = 50$ punti e restano in fascia gialla. Quest'ultimo comportamento è voluto, poiché tali metriche sono più esposte ai falsi positivi e non devono mai, da sole, provocare un intervento.

Il limite superiore di cento punti completa il quadro. Al di sopra della fascia rossa non esiste un'azione ulteriore che l'analista possa intraprendere, per cui un host con cinque metriche attive non richiede una risposta diversa da un host con tre. La saturazione mantiene inoltre il punteggio confrontabile fra host, fra reti e fra istanti di osservazione diversi, mentre un punteggio illimitato tornerebbe a essere, di fatto, una graduatoria dei sospetti del momento.

Le soglie delle fasce e le penalità restano parametri configurabili del sistema, raccolti nel file di configurazione descritto nella Sezione 5.3.

3.6 Valutazione dell'intera rete

L'ultimo obiettivo del modello non è la classificazione del singolo nodo, ma la valutazione complessiva dello stato di sicurezza della rete, secondo il principio per cui la sicurezza di un sistema è pari a quella del suo anello più debole [34].

In particolare la rete è considerata a rischio quando anche un solo host supera o eguaglia la soglia critica $\tau = 60$ corrispondente alla fascia rossa. Il solo valore di picco, tuttavia, non distingue un'anomalia isolata da una compromissione diffusa. Per distinguere queste situazioni, si definisce l'insieme degli host critici:

$$C = \{\, h \in H \mid S(h) \geq \tau \,\}$$

la cui cardinalità $|C|$ qualifica lo stato della rete che cambia a seconda del valore assunto:

- 0: indica una rete sicura.
- 1: anomalia localizzata.
- > 1: diffusione sistemica, potenzialmente riconducibile a un movimento laterale in corso.

Combinando i risultati, il modello restituisce lo stato di sicurezza della rete e, al tempo stesso, una classifica di priorità degli host, così che l'analista possa individuare rapidamente i nodi che richiedono attenzione e intervenire in modo tempestivo.

Capitolo 4

Selezione delle metriche

Questo capitolo presenta l'insieme delle metriche su cui si fonda il sistema, dai criteri che ne hanno guidato la scelta fino alla loro definizione puntuale e alla calibrazione dei pesi. Le prime sezioni illustrano i criteri di selezione (Sezione 4.1), gli scenari tipici di minaccia che ne motivano l'adozione (Sezione 4.2) e l'insieme effettivamente implementato (Sezione 4.3). Le sezioni successive descrivono in dettaglio ciascuna metrica, distinguendo tra metriche deterministiche e statistico-comportamentali, per chiudersi con la calibrazione e il riepilogo dei pesi (Sezione 4.8).

4.1 Criteri di selezione

Tra il vasto spettro di anomalie osservabili nel traffico di rete, è stato selezionato un sottoinsieme ristretto di metriche, cercando un compromesso tra un'ampia copertura delle minacce e basso costo computazionale. La selezione è stata guidata da tre criteri.

Il primo è l'**efficienza computazionale** sugli strumenti adottati. Sono state privilegiate le metriche che sfruttano la capacità di aggregazione nativa del database colonnare di ClickHouse e i segnali già calcolati da nDPI, evitando elaborazioni molto costose. Un esempio significativo riguarda il rilevamento di *Domain Generation Algorithm* (DGA). Anziché calcolare l'entropia di *Shannon* sulle stringhe dei domini interrogati, operazione costosa per via dell'elevato numero di richieste, si osserva il tasso di fallimento dei flussi dell'host (metrica M_{fail}), tra i cui criteri rientrano le risoluzioni DNS senza risposta e i domini inesistenti [7]. Un algoritmo di generazione dei domini produce infatti un'ampia maggioranza di richieste destinate a fallire, e il segnale che ne deriva si ottiene con interrogazioni SQL dal costo trascurabile.

Il secondo criterio è la **copertura delle fasi di un attacco**. Il sottoinsieme è stato bilanciato in modo da intercettare ogni stadio osservabile in rete della cosiddetta *Cyber Kill Chain* [18], dal primo contatto con il server di comando e controllo (C2) fino all'esfiltrazione finale dei dati, così che un attacco non possa attraversare tutte le

fasi senza essere notato. A questo criterio si affianca il framework MITRE ATT&CK [38], che classifica in modo sistematico le tecniche impiegate dagli attaccanti e offre un vocabolario di riferimento per verificare che le metriche selezionate corrispondano ad azioni concrete e osservabili nel traffico, anziché ad anomalie astratte.

Il terzo criterio è la **contenuta probabilità di falsi positivi**, strettamente legato all'affidabilità dell'origine dei dati. Sono state preferite metriche basate su segnali oggettivi e verificabili, evitando quelle che avrebbero richiesto fonti esterne inaffidabili o soglie arbitrarie. Lo stesso principio si riflette, come visto nel Capitolo 3, nella scelta di non segnalare un host sulla base di una singola metrica isolata, ma solo al sovrapporsi di più anomalie.

4.2 Scenari di minaccia

Per mostrare come l'insieme delle metriche copra le principali tipologie di attacco, si considerano tre scenari, descrivendo per ciascuno quali metriche si attivano e in quale precisa fase.

4.2.1 Ransomware e movimento laterale

Un dipendente apre un allegato di *phishing* e scarica inconsapevolmente un *malware*, che prima di cifrare i dati esplora la rete per propagarsi. Nella fase di contatto con il server di comando e controllo (C2) intervengono M_{rep}, che scatta se la destinazione contattata è già nota in una delle *blacklist* integrate in ntopng, e M_{cert}, che segnala l'uso di un certificato TLS problematico (ad esempio autofirmato), tipico delle infrastrutture di attacco improvvisate. Nella fase di ricognizione, M_{scan} rileva il tentativo di identificare altri dispositivi attivi nella sottorete, distinguendo i vari tipi di scansione. Infine, mentre tenta di propagarsi, l'host inizia a contattare servizi interni su porte server che non aveva mai contattato in precedenza, e M_{srv} ne rileva il comportamento anomalo.

4.2.2 Esfiltrazione e tunneling

Un attaccante che ha già ottenuto l'accesso a un server interno tenta di sottrarre dati mascherando il trasferimento, incapsulando il traffico in un tunnel cifrato e instradandolo sulla porta 443, sperando che il *firewall* lo scambi per normale traffico HTTPS. In questo caso M_{sni} rileva l'assenza del campo SNI nell'*handshake*, tipica di chi contatta un server direttamente per indirizzo IP, mentre M_{proto}, grazie alla *Deep Packet Inspection*, riconosce la firma di un protocollo diverso nascosto sulla porta 443, una classica tecnica di tunneling per aggirare i controlli perimetrali. Al momento del trasferimento, M_{vol} osserva il volume del traffico in uscita e, rilevando che supera notevolmente il valore tipico dell'host, ne segnala lo scostamento.

4.2.3 Botnet e DGA

Un host compromesso entra a far parte di una *botnet* e, per ricevere istruzioni eludendo le blacklist, impiega un *Domain Generation Algorithm* (DGA), che tenta in sequenza centinaia di domini casuali, di cui uno solo è effettivamente valido. Poiché la maggior parte di quei domini non esiste, il tasso di fallimento delle connessioni dell'host, storicamente trascurabile secondo la *baseline*, cresce bruscamente, e M_{fail} ne rileva l'anomalia. Come anticipato, questo segnale è ottenuto a costo quasi nullo, senza ricorrere al calcolo dell'entropia testuale delle interrogazioni DNS.

4.3 L'insieme delle metriche adottato

Sulla base dei criteri e degli scenari descritti, il sistema implementa otto metriche, riportate nella Tabella 4.1 insieme al tipo e all'anomalia che ciascuna rileva.

Metrica	Tipo	Anomalia rilevata
M_{rep}	Deterministica	Destination reputation
M_{cert}	Deterministica	TLS certificate anomalies
M_{sni}	Deterministica	SNI evasion
M_{srv}	Deterministica	Server port detected
M_{proto}	Deterministica	Non-standard port/protocol
M_{scan}	Deterministica	Network discovery
M_{vol}	Statistica	Asimmetria volumetrica
M_{fail}	Statistica	Connection failure rate

Tabella 4.1: Insieme delle metriche implementate nel sistema.

Sei metriche sono di natura deterministica e operano per confronto diretto con insiemi noti o recependo allarmi già prodotti dalla sonda; le restanti due, M_{vol} e M_{fail}, sono statistico-comportamentali e si basano sulla *baseline* storica dell'host secondo il meccanismo del *Modified z-score* descritto nel Capitolo 3.

Inoltre, una nona metrica basata sull'impronta crittografica del client (*client fingerprinting* tramite lo standard JA4) è stata progettata ma non inclusa nell'implementazione corrente. Pur essendo concettualmente valida, non è stata mantenuta per la ragione descritta di seguito. Allo stato attuale, infatti, l'insieme delle firme JA4 di *malware* disponibili pubblicamente è troppo limitato perché la metrica sia efficace, e permette di riconoscere con certezza solo poche famiglie. La metrica e le sue prospettive di integrazione sono discusse fra gli sviluppi futuri (Capitolo 7).

4.4 Metriche in dettaglio

Definito l'insieme delle metriche, le sezioni seguenti ne descrivono il funzionamento in dettaglio, esaminando per ciascuna il fenomeno analizzato, il segnale concreto da

cui è ricavata, la condizione di attivazione e la penalità assegnata. La Sezione 4.5 raccoglie gli elementi comuni a tutte le metriche, mentre le Sezioni 4.6 e 4.7 trattano rispettivamente le metriche deterministiche e quelle statistico-comportamentali. La calibrazione dei pesi e il loro riepilogo chiudono il capitolo (Sezione 4.8).

4.5 Elementi comuni

4.5.1 Soggetti analizzati e filtraggio della rete locale

Tutte le metriche hanno come soggetto gli host interni della rete monitorata, ovvero quelli di cui si vuole classificare il comportamento. Ogni metrica applica un filtro che seleziona come soggetto soltanto gli indirizzi appartenenti agli intervalli locali configurati (per impostazione predefinita le reti private RFC-1918 [27], estendibili alle subnet del contesto in esame). Le metriche che osservano una destinazione esterna applicano un secondo filtro che esclude le destinazioni non instradabili (come gli indirizzi di *loopback* e *link-local*), in modo da considerare soltanto le comunicazioni effettivamente dirette verso l'esterno e non il traffico interno alla rete.

4.5.2 Finestra di osservazione

Le metriche deterministiche operano su una finestra corrente della durata di un'ora, valutando gli eventi e i segnali registrati nell'ultimo intervallo. Le metriche statistiche confrontano l'ora corrente con una *baseline* costruita sui sette giorni precedenti, secondo il meccanismo descritto nella Sezione 4.7.1.

4.5.3 Origine dei segnali

Le metriche utilizzano tre fonti di dati distinte, tutte materializzate su ClickHouse a partire dai dati e segnali prodotti dalla sonda ntopng:

- la tabella `flows`, che contiene i metadati di ogni flusso (indirizzi, byte scambiati, *bitmask* dei rischi nDPI e indicatori per la reputazione);
- la vista `flow_alerts_view`, derivata da `flows` e ristretta ai soli flussi che hanno generato un allarme, usata dalle metriche che ispezionano singoli bit di rischio;
- la tabella `host_alerts`, che raccoglie gli allarmi prodotti da ntopng a livello di host e non di singolo flusso.

4.6 Metriche deterministiche

Le metriche deterministiche, come anticipato nel Capitolo 3, non richiedono uno storico: confrontano l'osservazione corrente con un insieme di dati noti o recepiscono un allarme già prodotto dalla sonda, restituendo un esito binario. In diversi casi, pur restando la metrica binaria nel suo valore $M_i \in \{0, 1\}$, la penalità p_i non è

fissa ma graduata in base alla gravità del segnale osservato, così da distinguere un evento isolato (possibile falso positivo) da un comportamento ripetuto e quindi più indicativo.

4.6.1 M_{rep} – Destination reputation

La metrica rileva le comunicazioni tra un host interno e indirizzi noti come malevoli (server di C2, nodi di uscita Tor, domini di *phishing*), sfruttando le *blacklist* integrate nella sonda ntopng. Il segnale è già disponibile nella tabella `flows` sotto forma di due indicatori booleani per flusso: uno segnala che la destinazione contattata è in *blacklist*, l'altro che è l'host interno a essere stato contattato da un indirizzo in *blacklist*.

Su questi due indicatori la metrica conta, nella finestra temporale corrente, il numero di occorrenze di contatti verso una destinazione malevola (n_{srv}) e il numero di occorrenze come bersaglio di un contatto malevolo (n_{cli}). La metrica si attiva alla presenza di almeno un riscontro:

$$M_{rep}(h) = 1 \iff n_{srv} + n_{cli} > 0.$$

La penalità è graduata su due dimensioni, la direzione del contatto e la sua intensità. La direzione verso l'esterno (l'host che contatta un indirizzo malevolo) è considerata più grave, poiché segnala una compromissione attiva, mentre la direzione opposta (l'host bersaglio di una scansione) è un segnale di esposizione più debole. L'intensità distingue il contatto isolato, che potrebbe derivare da una *blacklist* obsoleta, dal contatto ripetuto, tipico di una comunicazione persistente con un server di controllo (C2). La penalità è quindi:

$$p_{rep} = \begin{cases} 50 & \text{se } n_{srv} \geq 3 \quad \text{(contatto persistente verso server malevolo)} \\ 30 & \text{se } 1 \leq n_{srv} \leq 2 \quad \text{(contatto isolato verso server malevolo)} \\ 15 & \text{se } n_{srv} = 0 \text{ e } n_{cli} \geq 11 \quad \text{(scansione mirata in ingresso)} \\ 5 & \text{se } n_{srv} = 0 \text{ e } 1 \leq n_{cli} \leq 10 \quad \text{(scansione rara in ingresso)} \end{cases}$$

Il caso verso server malevolo prevale sempre su quello in ingresso, ovvero, se entrambi sono presenti, si applica la penalità più grave. La penalità non viene mai azzerata in presenza di un riscontro, per non perdere attacchi *low and slow* che generano un solo contatto periodico.

4.6.2 M_{cert} – TLS certificate anomalies

La metrica rileva comunicazioni verso server che presentano un certificato TLS problematico, condizione frequente nelle infrastrutture di attacco improvvisate. Il segnale proviene dalla *bitmask* dei rischi prodotta da nDPI, di cui la metrica ispeziona

tre bit corrispondenti al certificato autofirmato, al certificato scaduto e al certificato firmato con algoritmo SHA-1 ritenuto debole. Considerando B_{cert} l'insieme di tali bit di rischio, la metrica conta i flussi della finestra corrente in cui almeno uno di essi è acceso e si attiva se tale conteggio è positivo:

$$M_{cert}(h) = 1 \iff \exists \text{ almeno un flusso dell'host con un bit in } B_{cert} \text{ acceso.}$$

La penalità è statica e pari a $p_{cert} = 40$. Si tratta di un'anomalia strutturale grave, ma non sufficiente da sola a portare l'host in fascia rossa. Il valore è calibrato in modo che il superamento della soglia critica richieda comunque l'aggiunta di una seconda metrica.

4.6.3 M_{sni} – SNI evasion

La metrica rileva i flussi TLS privi del campo *Server Name Indication* (SNI) nell'*handshake*. I *client* legittimi includono sempre il nome del dominio di destinazione in tale campo; la sua assenza è un'anomalia strutturale tipica di chi contatta un server direttamente per indirizzo IP, evitando la risoluzione DNS per non lasciare tracce. Il segnale corrisponde a un singolo bit di rischio nDPI. La metrica conta i flussi della finestra corrente in cui quel bit è acceso e si attiva se il conteggio è positivo:

$$M_{sni}(h) = 1 \iff \exists \text{ almeno un flusso TLS dell'host privo di SNI.}$$

La penalità è statica e pari a $p_{sni} = 50$, il valore massimo previsto per una singola metrica, condiviso con i casi più gravi di M_{rep} e M_{scan}. Il valore è giustificato dalla bassissima incidenza di falsi positivi associata a questo segnale.

4.6.4 M_{proto} – Non-standard port/protocol

La metrica rileva i tentativi di nascondere traffico non autorizzato all'interno di canali considerati legittimi, come un protocollo applicativo incapsulato su una porta che di base ospita un servizio diverso (tecnica classica di *tunneling* per aggirare i controlli perimetrali). Il riconoscimento si basa sulla capacità di nDPI di identificare il protocollo reale dal contenuto del pacchetto e di confrontarlo con la porta utilizzata; in caso di discordanza, si accende il relativo bit di rischio. La metrica conta i flussi della finestra corrente con quel bit acceso e si attiva se il conteggio è positivo:

$$M_{proto}(h) = 1 \iff \exists \text{ almeno un flusso dell'host con protocollo su porta non standard.}$$

La penalità è statica e pari a $p_{proto} = 25$. Il valore contenuto è dato dalla natura debole del segnale: la discordanza tra protocollo e porta indica un'anomalia, non necessariamente un attacco, e per portare l'host in fascia rossa è richiesta l'aggiunta di un'ulteriore metrica.

4.6.5 M_{srv} – Server port detected

La metrica segnala gli host interni che iniziano a contattare una nuova porta server non presente nel loro profilo. A differenza delle metriche precedenti, il segnale non è un bit di flusso ma un allarme prodotto dalla sonda a livello di host. ntopng costruisce, per ciascun host e durante un periodo di apprendimento (*learning period*), l'insieme delle porte server abitualmente contattate; quando, terminato l'apprendimento, compare un contatto verso una nuova porta server, ntopng genera l'allarme nativo.

La metrica rileva quindi una comunicazione in uscita verso un servizio o una porta non osservati in precedenza, segnale potenzialmente associabile a un primo contatto con infrastruttura di comando e controllo (C2) o a una fase di movimento laterale, ma di per sé debole, dato che basta una semplice interrogazione verso un servizio mai contattato prima per attivarlo.

La metrica recepisce l'allarme dalla tabella `host_alerts` e si attiva alla presenza di almeno un'occorrenza nella finestra corrente:

$M_{srv}(h) = 1 \iff \exists$ almeno un allarme di contatto verso una nuova porta server per l'host.

La penalità è statica e pari a $p_{srv} = 20$, indipendente dal numero di porte coinvolte; il numero e l'elenco delle porte vengono comunque estratti a fini diagnostici, ma non incidono sul punteggio. Il valore è inferiore alla fascia gialla (fissata a 30): per questo la sola attivazione di M_{srv} lascia l'host in fascia verde, e la metrica contribuisce a portarlo in fascia gialla o rossa unicamente in concorso con altri segnali. Questa scelta riflette la natura della metrica, un indizio debole di contatto anomalo che acquista valore come conferma di evidenze prodotte da altre metriche.

4.6.6 M_{scan} – Network discovery

La metrica rileva gli host che effettuano scansioni della rete, segnale caratteristico delle fasi di ricognizione e di movimento laterale. Anche in questo caso il segnale è un allarme prodotto da ntopng a livello di host. L'analisi della sonda ha mostrato che un unico allarme di scansione racchiude, in un campo strutturato, il codice del tipo di scansione rilevata.

Tali codici distinguono cinque casi: flussi incompleti, scansione di sola ricezione (che identifica la vittima e non l'attaccante), scansione SYN, scansione FIN e scansione

RST. La metrica esclude volutamente il caso della vittima, che non viene penalizzata, e si attiva negli altri casi:

$$M_{scan}(h) = 1 \iff \text{l'host risulta originatore di almeno una scansione.}$$

La penalità viene assegnata considerando il caso peggiore osservato per l'host, ordinando le tecniche per gravità:

$$p_{scan} = \begin{cases} 50 & \text{scansione FIN o RST (evasione di firewall stateful)} \\ 40 & \text{scansione SYN (ricognizione classica)} \\ 30 & \text{soli flussi incompleti (scansione base)} \end{cases}$$

Le scansioni FIN e RST ricevono la penalità più alta perché, non aprendo nuove connessioni, sfuggono ai controlli che ispezionano i soli pacchetti SYN e sono tipiche di strumenti avanzati. Se per lo stesso host si osservano più tecniche, prevale la penalità più grave.

4.7 Metriche statistico-comportamentali

Le due metriche statistiche, M_{vol} e M_{fail}, non confrontano l'osservazione corrente con un insieme di dati noti, ma con il profilo storico dell'host stesso, attivandosi quando il comportamento corrente se ne discosta in modo significativo. Condividono diverse caratteristiche, descritte nella Sezione 4.7.1, e si differenziano per la grandezza osservata: un volume di byte per M_{vol}, una frazione di connessioni fallite per M_{fail}.

4.7.1 Caratteristiche comuni

Per ciascun host si costruisce una *baseline* aggregando il traffico dei sette giorni precedenti in intervalli orari (*bucket*). Per rendere il confronto valido, l'ora corrente non viene confrontata con tutte le ore passate, ma soltanto con quelle appartenenti alla stessa categoria temporale. Si distinguono tre categorie:

- ore lavorative dei giorni feriali
- ore non lavorative dei giorni feriali
- fine settimana

riflettendo il fatto che il traffico di un host varia in modo prevedibile a seconda del momento. Un picco di traffico che sarebbe anomalo in piena notte può essere del tutto normale in orario lavorativo.

Una volta introdotto il concetto di *baseline*, se ne calcolano i due indicatori definiti nel Capitolo 3: la mediana $\tilde{x}$ dei valori orari e la loro *Median Absolute Deviation*

$MAD = \text{Mediana}\big(|x_i - \tilde{x}|\big)$. Lo scostamento dell'osservazione corrente x è misurato dal **modified z-score**.

Come anticipato nel Capitolo 3, nella formulazione utilizzata nel sistema il modified z-score perde il valore assoluto presente nella definizione generale e assume la forma direzionale:

$$Z_{modified} = \frac{x - \tilde{x}}{MAD_{eff}}$$

La ragione è che entrambe le metriche sono interessate ai soli scostamenti in eccesso, ovvero un volume di uscita o un tasso di fallimenti superiori alla norma sono sospetti, mentre valori inferiori alla mediana (un host meno attivo del solito) non lo sono. Mantenendo il segno, i valori negativi vengono scartati dalla condizione $Z_{modified} > 3$, evitando di segnalare diminuzioni innocue.

Come si nota dalla formula, il denominatore non è la MAD grezza ma una MAD rivista MAD_{eff} (che sta per effettiva), dotata di un limite inferiore. Senza tale accorgimento, un host molto regolare avrebbe una MAD prossima a zero e qualsiasi minima variazione genererebbe uno z-score molto grande, producendo falsi positivi. Il limite inferiore è specifico per ciascuna metrica e viene discusso nelle rispettive sezioni.

La soglia $Z_{modified} > 3$ è stata fissata sulla base dei risultati della fase di test (Capitolo 6). Per garantire che la *baseline* sia statisticamente affidabile, un host viene valutato solo se dispone di almeno trenta *bucket* orari validi nella finestra storica; al di sotto di tale soglia l'host è considerato in fase di apprendimento (*learning period*) e non viene segnalato.

4.7.2 M_{vol} – Asimmetria volumetrica

La metrica rileva l'esfiltrazione di dati osservando il volume di byte trasmessi dall'host verso destinazioni esterne. Un host aziendale tipicamente riceve molti più dati di quanti ne invii; un'inversione improvvisa di questa dinamica, con un volume in uscita anomalo, suggerisce che dei dati siano copiati verso un server esterno. La grandezza osservata V_{out} è la somma dei byte in uscita dell'ora corrente verso destinazioni esterne, confrontata con la mediana $\tilde{V}$ della *baseline*.

Per questa metrica il limite inferiore sulla dispersione è triplo, a protezione di diversi scenari:

$$MAD_{eff} = \max\big(MAD,\ 0{,}10 \cdot \tilde{V},\ 1\,\text{MB}\big)\,.$$

Il termine proporzionale (10% della mediana) adatta il limite alla scala dell'host, mentre il termine assoluto (1 MB) evita che un host abitualmente quasi inatti-

vo generi falsi positivi con fluttuazioni minime. La metrica si attiva quando sono soddisfatte simultaneamente quattro condizioni:

$$M_{vol}(h) = 1 \iff \begin{cases} Z_{modified} > 3 & \text{(significatività statistica)} \\ V_{out} > \tilde{V} & \text{(solo eccessi)} \\ V_{out} > 50\,\text{MB} & \text{(significatività operativa)} \\ \text{bucket di baseline} \geq 30 & \text{(baseline sufficiente)} \end{cases}$$

La terza condizione garantisce che un'anomalia significativa ma di entità trascurabile in termini di byte non venga segnalata. La penalità è $p_{vol} = 20$, la più bassa del modello, in ragione della maggiore probabilità di falsi positivi rispetto alle metriche deterministiche.

4.7.3 M_{fail} – Connection failure rate

La metrica rileva un aumento anomalo delle connessioni fallite, classica caratteristica di un *Domain Generation Algorithm* (DGA), che tenta in sequenza molti domini inesistenti. La grandezza osservata è il tasso di fallimento $r = |F_{fail}|/|F_{tot}|$, rapporto tra flussi falliti e flussi totali, confrontato con la mediana $\tilde{r}$ della *baseline*.

Un flusso è considerato fallito quando si verifica almeno una delle seguenti condizioni:

- l'assenza completa di risposta dal lato destinazione (server irraggiungibile o richiesta DNS in *timeout*)
- l'accensione di uno tra i bit di rischio nDPI che segnalano un errore esplicito di protocollo (per esempio una risposta DNS *NXDOMAIN* o un codice di errore HTTP)
- traffico unidirezionale
- problemi a livello TCP (come un *reset* della connessione)
- un *hostname* mai risolto in precedenza (indicatore tipico di DGA)
- tentativo di *probing*

Poiché il tasso di fallimento si trova nell'intervallo $[0, 1]$, il limite inferiore sulla dispersione in questo caso è soltanto assoluto, senza il termine proporzionale presente in M_{vol}, che sarebbe quasi sempre dominato dal primo:

$$MAD_{\text{eff}} = \max(MAD,\ 0{,}05)$$

La metrica si attiva quando sono soddisfatte simultaneamente cinque condizioni:

$$M_{fail}(h) = 1 \iff \begin{cases} Z_{modified} > 3 & \text{(significatività statistica)} \\ r > \tilde{r} & \text{(solo aumenti)} \\ r > 0{,}30 & \text{(significatività operativa)} \\ |F_{tot}| \geq 50 & \text{(grandezza campione corrente)} \\ \text{bucket di baseline} \geq 30 & \text{(baseline sufficiente)} \end{cases}$$

La quarta e la quinta condizione assicurano che vi siano abbastanza dati, sia nella finestra corrente che nella *baseline*, perché il confronto sia significativo. La condizione di significatività operativa ($r > 0{,}30$) garantisce inoltre che un host passato, ad esempio, dallo 0,1% al 2% di fallimenti non venga segnalato pur avendo uno z-score elevato. La penalità è $p_{fail} = 30$.

4.8 Calibrazione e riepilogo dei pesi

I pesi p_i definiti nel Capitolo 3 sono parametri configurabili del sistema, ma i valori adottati nella configurazione predefinita non sono casuali. Essi risultano da due fasi distinte: la struttura del modello delimita gli intervalli entro cui ciascuna penalità può ricadere, e una successiva calibrazione su un dataset dedicato ne fissa il valore all'interno di tali intervalli.

4.8.1 Criteri di ordinamento

Tre criteri, applicati nell'ordine, stabiliscono la gravità relativa delle metriche.

Il primo è la **rarità** del segnale, cioè quanto raramente esso compare su traffico legittimo. L'assenza del campo SNI in un handshake TLS non ha praticamente alcuna spiegazione benigna, e M_{sni} riceve per questo la penalità massima prevista per una singola metrica. All'estremo opposto, la discordanza fra protocollo e porta ricorre anche in reti sane, dove un servizio interno può risultare legittimo se ascolta su una porta non convenzionale, e M_{proto} riceve quindi una penalità contenuta.

Il secondo è la **direzionalità** del comportamento osservato. Un host che contatta un indirizzo noto come malevolo manifesta un'attività propria, mentre un host contattato da un indirizzo malevolo subisce soltanto un'esposizione, condizione che su una rete connessa a Internet costituisce rumore di fondo. La stessa metrica M_{rep} assume perciò penalità comprese fra 30 e 50 punti nel primo caso e fra 5 e 15 nel secondo. La medesima asimmetria si ritrova nei pesi di BotHunter [16], stimati dagli autori con un modello di regressione, in cui i segnali generati dall'host interno pesano circa quattro volte quelli che l'host riceve.

Il terzo è la **natura del segnale**. Le metriche deterministiche confrontano l'osservazione con un insieme noto e producono un'evidenza puntuale, mentre le metriche

statistico-comportamentali confrontano l'osservazione con una *baseline* stimata sul passato dell'host e sono quindi più esposte ai falsi positivi. A parità di gravità apparente, le seconde ricevono penalità più basse. Fra le due, M_{fail} pesa più di M_{vol} perché la sua attivazione richiede cinque condizioni simultanee anziché quattro, e il segnale che ne risulta è di conseguenza più affidabile.

4.8.2 Gruppi di gravità e vincoli

L'applicazione dei tre criteri distribuisce le metriche nei due gruppi introdotti nella Sezione 3.5.1. Al gruppo dei segnali forti, quelli che da soli collocano l'host in fascia gialla, appartengono M_{sni}, M_{cert}, M_{scan}, M_{fail} e M_{rep} nella direzione uscente. Al gruppo dei segnali deboli, che da soli lasciano l'host in fascia verde, appartengono M_{proto}, M_{srv}, M_{vol} e M_{rep} nella direzione entrante.

La coerenza fra pesi e soglie impone tre vincoli, che nessuna assegnazione dei valori può violare. Nessuna penalità raggiunge i 60 punti della fascia rossa, e la più alta si ferma a 50, così che nessun segnale isolato possa da solo ordinare un intervento. Nessuna penalità cade fra 26 e 29 punti, così che il confine dei due gruppi coincida esattamente con la soglia della fascia gialla. Infine le due metriche statistico-comportamentali, se attive contemporaneamente e in assenza di altri segnali, sommano 50 punti e restano in fascia gialla, coerentemente con la loro maggiore esposizione ai falsi positivi.

4.8.3 Calibrazione

Entro i margini lasciati liberi da questi vincoli, i valori sono stati fissati mediante una calibrazione condotta sul dataset CICIoT2023 [28], una raccolta recente e di ampia scala di traffico di attacco generato su una topologia estesa di dispositivi IoT reali. Il dataset è stato mantenuto distinto dai dataset impiegati per la validazione. Le penalità sono state regolate osservando la risposta delle metriche al traffico etichettato del dataset, così che i segnali riconducibili ad attività malevola producessero punteggi coerenti con le fasce di rischio previste, mantenendo al contempo basso il punteggio del traffico benigno.

I risultati numerici di questa calibrazione non sono riportati perché CICIoT2023 non documenta gli indirizzi degli host malevoli e benigni, e non consente quindi di ricostruire la valutazione per singolo host su cui si fonda il modello. Il dataset è stato perciò impiegato unicamente per la taratura dei pesi e non rientra nel conteggio dei risultati di validazione presentati nel Capitolo 6. Una volta calibrati, i pesi non sono più stati modificati durante la validazione, così da mantenere separata la fase di calibrazione da quella di misura delle prestazioni.

4.8.4 Riepilogo

La Tabella 4.2 riassume le penalità di tutte le metriche nella configurazione predefinita, con l'indicazione del gruppo di appartenenza. Dove la penalità è graduata, sono riportati i valori minimo e massimo applicabili.

Metrica	**Tipo**	**Penalità**	**Gruppo**
M_{rep}	Deterministica (graduata)	5 – 50	Debole/Forte
M_{cert}	Deterministica (statica)	40	Forte
M_{sni}	Deterministica (statica)	50	Forte
M_{proto}	Deterministica (statica)	25	Debole
M_{srv}	Deterministica (statica)	20	Debole
M_{scan}	Deterministica (graduata)	30 – 50	Forte
M_{vol}	Statistica	20	Debole
M_{fail}	Statistica	30	Forte

Tabella 4.2: Penalità associate a ciascuna metrica nella configurazione predefinita.

Capitolo 5

Implementazione

I capitoli precedenti hanno definito il modello di rischio (Capitolo 3) e descritto le metriche che lo compongono (Capitolo 4). Questo capitolo illustra come i componenti sono collegati tra loro, dove risiedono e come vengono calcolati i dati, e in che modo i risultati delle singole metriche vengono aggregati in un punteggio per host e in una valutazione della rete. La descrizione degli strumenti (ntopng, nDPI, ClickHouse) è già stata data nel Capitolo 2; qui se ne descrive il ruolo e l'integrazione nel sistema realizzato.

5.1 Sistema a tre livelli

Il sistema è organizzato in tre livelli con responsabilità distinte, attraversati dai dati in un'unica direzione, dalla cattura del traffico fino alla produzione del report finale.

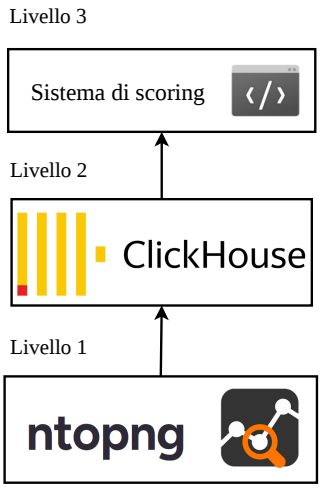

Figura 5.1: Sistema a tre livelli

Il primo livello è la sonda **ntopng**, che osserva passivamente il traffico, ne riconosce i flussi tramite nDPI e produce due tipi di informazioni:

- i metadati di ogni flusso (indirizzi, porte, byte scambiati, protocollo riconosciuto e *bitmap* dei rischi)
- gli allarmi nativi generati dai propri controlli, sia a livello di singolo flusso sia a livello di host

Il secondo livello è il database **ClickHouse**, verso cui ntopng esporta con continuità sia i flussi che gli allarmi. ClickHouse costituisce la memoria storica del sistema; infatti, è qui che si conserva il traffico dei giorni precedenti, indispensabile alle metriche statistiche per confrontare il comportamento corrente con la *baseline* dell'host.

Il terzo livello è il **motore di scoring**, un componente esterno scritto in Python che, a intervalli regolari, interroga ClickHouse tramite query, calcola tutte le metriche, aggrega i risultati per host e infine produce la valutazione finale. È importante sottolineare che l'aggregazione e il calcolo del punteggio non avvengono all'interno di ntopng, poiché la sonda si limita ad osservare e segnalare, mentre l'intera logica del modello risiede nel motore di scoring esterno. Questa separazione mantiene il sistema indipendente dal codice sorgente della sonda e consente di modificare pesi, soglie e metriche senza intervenire su ntopng.

5.2 Acquisizione e persistenza dei dati

Tutte le metriche leggono i propri dati da ClickHouse, utilizzando le fonti già introdotte nel Capitolo 4:

- la tabella `flows`, che raccoglie i metadati di ogni flusso
- la tabella `host_alerts`, che contiene gli allarmi relativi agli host
- la vista `flow_alerts_view`, derivata da `flows` e ristretta ai soli flussi che hanno generato un allarme, impiegata dalle metriche che ispezionano i singoli bit di rischio

La scelta di ClickHouse come componente per la persistenza dei dati non è casuale, ma è stata dettata da esigenze di efficienza. Trattandosi di un database colonnare, ClickHouse organizza i dati per colonna anziché per riga; questo permette di scandire la *baseline* dell'host, dell'ordine di milioni di flussi, in una frazione di secondo, senza rallentare l'attività di monitoraggio.

Su questa caratteristica si basa l'intera implementazione: i calcoli vengono effettuati all'interno del database anziché trasferire i dati verso gli script Python. Ogni metrica è realizzata come un'interrogazione SQL che esegue il filtraggio e l'aggregazione lato ClickHouse e restituisce al motore soltanto l'elenco degli host anomali con i relativi

dettagli. In questo modo il volume di dati che attraversa il confine tra database e applicazione resta minimo, indipendentemente dalla quantità di traffico analizzato.

5.3 Configurazione centralizzata

Tutti i parametri del sistema sono raccolti in un unico file di configurazione esterno, `config.ini`, letto a tempo di esecuzione da un modulo dedicato, `readconfig.py`, che si occupa di trasferire i valori al resto del sistema.

Il file è suddiviso in diverse sezioni:

- i parametri di connessione a ClickHouse
- la definizione delle reti locali ed esterne
- le soglie delle fasce di rischio
- le finestre temporali
- i pesi delle metriche, oltre ai parametri specifici di ciascuna

```
[reti]
locali       = 192.168.0.0/16, 10.0.0.0/8, 172.16.0.0/12

non_esterne = 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16,
127.0.0.0/8, 169.254.0.0/16

[scoring]
score_max     = 100
soglia_giallo = 30
soglia_rosso  = 60

[finestre]
finestra_corrente_ore   = 1
finestra_storico_giorni = 7
min_baseline_hours      = 30
ore_lavorative          = 9, 17
giorni_weekend          = 6, 7

[pesi]
m_cert = 40
m_proto = 25

[m_fail]
soglia_z            = 3
r_min_operativo     = 0.30
min_flussi_corrente = 50
; i numeri di bit nDPI possono cambiare tra versioni
bit_ndpi_error_code = 43
```

Listing 5.1: Estratto di config.ini con le sezioni principali.

Questa struttura esterna rende il sistema adattabile a diversi contesti senza modificare il codice né intervenire sulla sonda: soglie, pesi, finestre temporali e persino i numeri dei bit nDPI, che possono variare tra versioni di ntopng e nDPI, si regolano tutti dalla configurazione. Per esempio, per analizzare una rete con indirizzi differenti, o un file di cattura (*pcap*) con indirizzi pubblici, è sufficiente aggiornare gli intervalli di rete nella configurazione, allineandoli alla corrispondente impostazione della sonda (`ntopng.conf`) e inserire il relativo *timestamp* per impostare l'analisi nella finestra di riferimento nell'istante desiderato. Nel modulo di configurazione sono presenti anche le due funzioni che traducono gli intervalli di rete in clausole SQL, riusate da tutte le metriche per filtrare gli indirizzi IP interni ed esterni in modo uniforme.

5.4 Struttura modulare del sistema

Ogni metrica è implementata come un modulo Python indipendente, che utilizza una funzione di calcolo che restituisce un risultato uniforme. Ogni modulo riceve la connessione al database e restituisce un dizionario che associa a ciascun host anomalo i dettagli della rilevazione. Ogni host restituito deve contenere almeno la chiave `penalita`, ovvero il punteggio da applicare; le altre chiavi sono specifiche della metrica e servono al report diagnostico. Il Listing 5.2 mostra il formato di ritorno, qui nel caso di M_{proto}:

```
#Ogni metrica restituisce: dict[host_ip -> dettagli]
risultati[host_ip] = {
    "M_proto":                m_proto,
    "hits_totali":            hits_totali,
    "combinazioni_distinte": combinazioni_distinte,
    "lista_mismatch":         list(lista_mismatch),
    "penalita":               PESO_M_PROTO,
    "timestamp":              datetime.now(timezone.utc).
   isoformat()
}
return risultati
```

Listing 5.2: Dizionario uniforme restituito da ogni metrica.

Questa uniformità è un punto centrale dell'architettura software, poiché consente al motore di scoring di trattare tutte le metriche allo stesso modo, senza conoscere il funzionamento interno. Le metriche da eseguire sono raccolte in un unico punto (Listing 5.3); infatti, per aggiungere una nuova metrica basta scrivere il relativo modulo e aggiungere una riga al registro `METRICHE` in `scoring.py`, senza modificare il resto del sistema.

```
# Registro delle metriche da eseguire.
```

```
# Lista di tuple (nome_chiave_dict, nome_metrica, funzione_calcolo).
METRICHE = [
    ("M_rep",   "Destination reputation",      calcola_m_rep),
    ("M_cert",  "TLS certificate anomalies",   calcola_m_cert),
    ("M_sni",   "SNI evasion",                 calcola_m_sni),
    ("M_srv",   "Server port detected",       calcola_m_srv),
    ("M_proto", "Non-standard port/protocol", calcola_m_proto),
    ("M_scan",  "Network discovery",           calcola_m_scan),
    ("M_vol",   "Asimmetria volumetrica",      calcola_m_vol),
    ("M_fail",  "Connection failure rate",     calcola_m_fail),
]
```

Listing 5.3: Registro delle metriche eseguite dal motore.

Come anticipato nel Capitolo 2, ntopng adotta un punteggio di rischio cumulativo e privo di limite superiore, in cui ogni evento somma un valore predefinito, portando facilmente un host compromesso a migliaia di punti. Il modello qui proposto usa una scala diversa, limitata a cento. Il motore quindi non recepisce il valore numerico assegnato da ntopng ma usa l'allarme nativo come semplice interruttore logico, così da applicare le penalità definite dal modello. In questo modo i pesi restano interamente sotto il controllo del sistema, coerenti con l'equazione del rischio definita nel Capitolo 3.

5.5 Realizzazione delle query

Le metriche deterministiche traducono in una query la lettura di un segnale già prodotto dalla sonda, sia esso un allarme nativo o uno o più bit di rischio. Il Listing 5.4 riporta la query di M_{proto}, in cui una prima *Common Table Expression* (CTE) filtra i flussi della finestra corrente in cui è acceso il bit del rischio cercato, applicando il filtro condiviso sul soggetto interno; una seconda fase aggrega per host. Il peso non compare nella query, perché viene applicato in Python secondo il principio dell'attivazione del bit descritto sopra, ovvero l'interrogazione si limita a stabilire se la metrica vada attivata per gli host analizzati.

```
    -- FASE 1 - CTE "dati_filtrati"
    -- Filtro temporale + bit 5 acceso (mismatch protocollo/porta)
WITH dati_filtrati AS (
        SELECT
            cli_ip AS host_ip,
            srv_port,
            JSONExtractString(json, 'alerts', '30', 'proto.ndpi') AS proto_ndpi
        FROM flow_alerts_view
```

```
9         WHERE
10            tstamp >= {rif} - INTERVAL {finestra_minuti}
    MINUTE
11            AND bitTest(flow_risk_bitmap, 5) = 1
12            AND {filtro_lan}
13     )
14
15     -- FASE 2 - Aggregazione finale per host
16     SELECT
17         host_ip,
18         count() AS hits_totali,
19         uniqExact(concat(proto_ndpi, ':', toString(srv_port)))
     AS combinazioni_distinte,
20         arraySlice(
21             groupUniqArray(concat(proto_ndpi, ' su porta ',
    toString(srv_port))),
22             1, 10
23         ) AS lista_mismatch,
24         1 AS M_proto
25
26     FROM dati_filtrati
27     GROUP BY host_ip
28     ORDER BY hits_totali DESC
```

Listing 5.4: Interrogazione di una metrica deterministica (estratto di M_{proto}).

Una seconda tipologia di metrica deterministica non legge un bit di flusso ma un allarme di host, che può portare con sé un dettaglio importante. La metrica M_{scan} interroga la tabella `host_alerts` filtrando sull'identificativo dell'allarme di scansione, estrae con `JSONExtract` l'array dei codici di sottotipo e, in un'unica `multiIf`, esclude la vittima (codice che ntopng marca come tale) e assegna la penalità graduata secondo la tecnica osservata. Una seconda fase aggrega per host conservando il peso massimo e scartando gli allarmi di sola vittima.

```
1  -- FASE 1 - Estrazione e classificazione per singolo allarme
2  WITH dati_filtrati_e_classificati AS (
3      SELECT
4          ip AS host_ip,
5          JSONExtract(json, 'alerts', 'Array(Int32)') AS
    codici_scan,
6          multiIf(
7              -- vittima: esclusa
8              has(codici_scan, 1), 0,
9              -- FIN/RST: evasione firewall
10             has(codici_scan, 3) OR
11             has(codici_scan, 4), 50,
12             -- SYN scan classico
13             has(codici_scan, 2), 40,
14             -- soli flussi incompleti
```

```
15              30
16          ) AS penalita_singola
17      FROM host_alerts
18      WHERE alert_id = {alert_id_scan}
19        AND tstamp >= {rif} - INTERVAL {finestra_minuti} MINUTE
20        AND {filtro_lan}
21  )
22  -- FASE 2 - Aggregazione per host : peso massimo
```

Listing 5.5: Interrogazione di una metrica deterministica (estratto di M_{scan}).

Le metriche statistiche realizzano invece in SQL l'intera catena del confronto con la *baseline*, organizzata come una sequenza di CTE concatenate. Il Listing 5.6 ne mostra lo scheletro, omettendo il codice del corpo di ciascuna fase. La prima determina la categoria temporale dell'ora corrente; la seconda aggrega i sette giorni precedenti in *bucket* orari della stessa categoria; la terza calcola la mediana per host e applica il filtro di *cold start* (almeno trenta *bucket*); la quarta calcola la MAD, che ClickHouse non fornisce nativamente, come mediana delle distanze assolute dalla mediana; la quinta somma il traffico dell'ora corrente. Infine la `SELECT` finale unisce il traffico corrente e la *baseline*, calcola la MAD effettiva e il *modified z-score*, e applica le condizioni di scatto descritte nel Capitolo 4.

```
1  -- categoria temporale
2  WITH categoria_corrente AS ( ... ),
3  -- 7 giorni in bucket orari, stessa categoria
4       baseline_grezza      AS ( ... ),
5  -- mediana per host
6       mediane              AS ( ...
7  -- filtro cold start
8     HAVING count() >= 30 ),
9  -- MAD = median(|v_bucket - v_mediana|)
10    statistiche_baseline AS ( ... ),
11 -- volume ora corrente
12      volume_corrente      AS ( ... )
13 SELECT host_ip
14 FROM volume_corrente JOIN statistiche_baseline USING host_ip
15 -- condizioni di scatto
16 WHERE z_robusto > 3 AND v_out > v_mediana AND v_out > 50000000
```

Listing 5.6: Scheletro della sequenza di CTE (estratto di M_{vol}).

Le due metriche statistiche condividono questa struttura e si differenziano solo per la grandezza osservata. In M_{vol} è un volume di byte, in M_{fail} è un tasso di fallimento, e questa differenza si riflette nel modo in cui viene contata la grandezza per ogni bucket. A differenza di M_{vol}, che opera sui soli flussi in uscita, M_{fail} interroga l'intera tabella `flows`, poiché per calcolare un tasso servono sia i flussi falliti sia quelli riusciti. Un flusso è considerato fallito quando il destinatario non ha mai

risposto, oppure quando è acceso almeno uno dei cinque bit di rischio nDPI che segnalano un fallimento (errore di protocollo, traffico unidirezionale, problemi TCP, hostname non risolto, tentativo di probing), riuniti in un'unica maschera verificata con `bitAnd`, come mostra il Listing 5.7. Il tasso così ottenuto alimenta poi la stessa catena di MAD e *modified z-score* descritta sopra.

```
-- Un flusso    "fallito" se il server non ha mai risposto
-- oppure se    acceso almeno uno dei bit nDPI di fallimento.
countIf(
    DST2SRC_PACKETS = 0
    OR bitAnd(FLOW_RISK, {maschera_fallimento}) != 0
) AS n_falliti,

-- Tasso del bucket: frazione di flussi falliti sul totale
toFloat64(n_falliti) / toFloat64(count()) AS rate_bucket
```

Listing 5.7: Definizione del flusso fallito (estratto di M_{fail}).

Il calcolo resta interamente lato database, dato che, in entrambi i casi, l'interrogazione restituisce al motore soltanto gli host anomali, già affiancati dai valori necessari per il report.

5.6 Il motore di scoring

Il motore di scoring gestisce l'esecuzione delle metriche e l'aggregazione dei risultati. Un suo ciclo si articola in una sequenza di passaggi.

All'avvio viene aperta una sola connessione a ClickHouse, condivisa da tutte le metriche. Oltre a evitare l'*overhead* di connessioni ripetute, questo garantisce che ogni metrica interroghi una visione coerente dei dati nello stesso istante. Le metriche vengono poi eseguite in sequenza, ciascuna isolata dalle altre; infatti, se una metrica fallisce, l'errore viene registrato ma non interrompe il ciclo, e il sistema prosegue con le rimanenti. Questa scelta evita che un singolo malfunzionamento, ad esempio una colonna assente o una variazione dello schema, comprometta l'intera analisi.

Conclusa l'esecuzione, i risultati vengono aggregati per host. Per ciascun host il motore somma le penalità delle metriche che sono scattate e applica il limite superiore previsto dal modello, ottenendo lo *score* $S(h) = \min(100, \sum_i p_i \cdot M_i(h))$ definito nel Capitolo 3. Il Listing 5.8 mostra il nucleo di questa logica.

```python
def aggrega_per_host(risultati_per_metrica: dict) -> dict:
    # Per ogni metrica, scorro gli host che ha flaggato
    for nome_metrica, risultati in risultati_per_metrica.items():
        for host_ip, dettagli in risultati.items():
            # Inizializza il record dell'host se e' il primo incontro
            if host_ip not in aggregato:
                aggregato[host_ip] = {
                    "score":           0,
                    "fascia":          "VERDE",
                    "metriche_attive": [],
                    "penalita_totale": 0,
                    "dettagli":        {},
                }

            # Aggiungo le info di questa metrica al record
            # Se per qualche motivo la metrica non ha una chiave "penalita", ritorna 0 invece di fallire.
            penalita = dettagli.get("penalita", 0)
            aggregato[host_ip]["penalita_totale"] += penalita
            aggregato[host_ip]["metriche_attive"].append(nome_metrica)
            aggregato[host_ip]["dettagli"][nome_metrica] = dettagli

    # Calcolo lo score limitato e la fascia per ciascun host
    for host_ip, rec in aggregato.items():
        rec["score"]  = min(SCORE_MAX, rec["penalita_totale"])
        rec["fascia"] = fascia_rischio(rec["score"])

    return aggregato
```

Listing 5.8: Aggregazione e saturazione del punteggio.

Ogni punteggio viene mappato nella corrispondente fascia di rischio secondo le soglie configurate. Sulla base dei punteggi individuali, il motore deriva infine lo stato complessivo della rete, combinando il punteggio di picco con la cardinalità dell'insieme degli host critici, secondo il principio dell'anello più debole già definito nel Capitolo 3.

L'esito non è un singolo valore numerico, ma un report destinato all'analista, che comprende un riepilogo dello stato della rete con la distribuzione degli host per fascia, una tabella di tutti gli host segnalati ordinata per punteggio decrescente, e un dettaglio espanso dei soli host critici, in cui per ogni nodo sono elencate le metriche che ne hanno determinato il punteggio. Gli host in fascia verde, tipicamente segnalati da una sola metrica a basso peso, non vengono espansi per non appesantire l'output, pur restando conteggiati nel riepilogo. Inoltre il motore restituisce lo stato

globale della rete tramite il proprio codice di uscita, così che lo *scheduler* che lo avvia possa innescare allarmi esterni senza dover interpretare il testo del report.

Il Listing 5.9 mostra un estratto reale dell'output prodotto su una delle catture di validazione (Capitolo 6). È in questa forma che l'interpretabilità del modello diventa concreta: accanto al punteggio, l'analista vede esattamente quali metriche lo hanno determinato e con quale contributo.

```
  SISTEMA DI RILEVAMENTO ANOMALIE - SCORING ORARIO
  Metriche attive: 8
======================================================
  [fase 3] STATO DELLA RETE
  Host osservati con almeno 1 metrica attiva : 10

  Distribuzione per fascia:
    Verde   (0-29)  :   0
    Giallo  (30-59) :   0
    Rosso   (60-100):  10

  Score massimo della rete : 65
  Fascia globale           : [!!] ROSSO

  >> RETE COMPROMESSA: almeno un host ha superato la soglia
     di intervento.    richiesta verifica immediata.

======================================================
  TABELLA HOST SOSPETTI
======================================================
  Host                Score  Fascia   Metriche attive
  ------------------ -----  -------  ----------------
  147.32.84.209          65  [!!] ROSSO  M_proto, M_scan

======================================================
  DETTAGLIO HOST CRITICI (fascia gialla e rossa)
======================================================

------------------------------------------------------
  [!!] HOST 147.32.84.209  ->  Score: 65/100  (ROSSO)
------------------------------------------------------
    > M_proto  (+25 punti)
         hits_totali           : 770
         combinazioni_distinte : 3
         lista_mismatch        : ['HTTP su porta 6667', ...]
    > M_scan   (+40 punti)
         hits_totali           : 1
         codici_rilevati       : [2]
         codici_descritti      : ['SYN Scan']
         categoria_dominante   : SYN SCAN CLASSICO
```

Listing 5.9: Estratto ridotto del report prodotto dal motore di scoring.

5.7 Modello di esecuzione

Il motore di scoring opera a **batch orario**, eseguito automaticamente a ogni ora tramite uno *scheduler* di sistema. La scelta del batch è coerente con il modo in cui operano le metriche statistiche, che confrontano l'ultima ora di traffico con la *baseline* dei sette giorni precedenti e quindi lavorano su intervalli orari. Le metriche deterministiche, pur basandosi su allarmi generati dalla sonda ntopng in tempo reale, vengono aggregate allo stesso modo, poiché a ogni esecuzione il motore considera gli allarmi accumulati e il traffico osservato nell'ultima ora, producendo un report coerente dello stato della rete.

La finestra di osservazione predefinita è di un'ora. Le metriche deterministiche consentono di modificare tale ampiezza, mentre le due metriche statistiche operano necessariamente su intervalli orari.

In un dispiegamento su rete reale questo batch orario è affidato a `systemd`, il gestore di servizi di sistema, tramite due *unit* distinte: una di servizio, che esegue il motore, e una di timer, che ne pianifica l'avvio. La unit di servizio (Listing 5.10) è di tipo *oneshot*, esegue cioè un singolo ciclo e termina, evitando esecuzioni sovrapposte, e viene avviata solo dopo che ClickHouse e la rete sono disponibili, così da non interrogare una sorgente dati non ancora pronta. Il motore gira inoltre come utente non privilegiato, e i codici di uscita 1 e 2, che nel modello codificano le fasce gialla e rossa, sono dichiarati come esiti di successo affinché `systemd` non li interpreti come errori.

```
[Unit]
Description=Calcolo dello score di rischio degli host
# Avvia lo scoring dopo che ClickHouse e la rete sono pronti
After=clickhouse-server.service network-online.target
Wants=network-online.target

[Service]
Type=oneshot # un solo ciclo, evita esecuzioni sovrapposte
WorkingDirectory=/home/diego/tesi/src
ExecStart=/home/diego/tesi/venv/bin/python scoring.py
User=diego
Group=diego
# gli exit code 1 e 2 codificano le fasce giallo/rosso
SuccessExitStatus=1 2
```

Listing 5.10: Unit di servizio systemd per il motore di scoring.

La *unit* di timer (Listing 5.11) fissa l'avvio all'inizio di ogni ora e si attiva automaticamente all'accensione della macchina. Due impostazioni riflettono l'attenzione all'uso reale: `Persistent=true` recupera un'esecuzione eventualmente saltata, per esempio se la macchina era spenta all'ora prevista, mentre la pianificazione può es-

sere spostata di qualche minuto oltre l'ora esatta per lasciare a ntopng il tempo di esportare i flussi dell'ora appena conclusa prima che il motore li interroghi. In questo modo il sistema, oltre a essere validato su catture storiche, è predisposto per operare in modo continuativo e automatico su una rete.

```
[Unit]
Description=Avvio orario dello scoring

[Timer]
# All'inizio di ogni ora
OnCalendar=hourly
# recupera un'esecuzione saltata
Persistent=true
AccuracySec=1min

[Install]
WantedBy=timers.target
```

Listing 5.11: Unit di timer systemd per l'avvio orario.

Capitolo 6

Validazione e risultati

Dopo aver descritto il modello di scoring nel Capitolo 3 e la sua implementazione nel Capitolo 5, questo capitolo ne verifica il comportamento su traffico reale di malware e botnet. L'obiettivo è misurare la capacità del sistema di classificare correttamente gli host interni compromessi senza segnalare quelli legittimi. I risultati sono riassunti in una matrice di confusione e nelle metriche di valutazione che ne derivano.

6.1 Metodologia di validazione

Il modello è progettato per operare principalmente su reti reali, in modalità batch orario, dove l'istante di riferimento delle metriche coincide con il momento dell'esecuzione. È tuttavia possibile validarlo su catture storiche (file pcap) senza modificarne la logica, agendo su un solo parametro di configurazione. Nella sezione `[validazione]` del file `config.ini`, il campo `epoch_riferimento` (Listing 6.1) fissa l'istante rispetto al quale tutte le finestre temporali vengono calcolate. Se lasciato vuoto oppure impostato a zero, il sistema usa `now()` e opera in tempo reale; se contiene un valore, questo è un *Unix epoch*, cioè un istante univoco indipendente dalla sua rappresentazione locale, che le query convertono con `toDateTime(epoch)` in modo coerente con il fuso in cui ntopng registra i timestamp dei flussi. Impostando quel valore su un'ora contenuta nel traffico catturato, tutte le finestre temporali si allineano al pcap e le metriche possono essere calcolate sul traffico registrato come se fosse in corso.

```
[validazione]
; Istante di riferimento per l'analisi
; Vuoto o 0 = real time (le metriche usano now()).
epoch_riferimento = 1568966400
```

Listing 6.1: Sezione di validazione del file di configurazione.

Le catture utilizzate provengono da dataset pubblici di riferimento:

- le serie CTU-13 e IoT-23 dello Stratosphere Laboratory [15, 14] con l'aggiunta di varie catture di dispositivi domestici commerciali per il controllo dei falsi positivi
- una cattura da CIC-IDS2017 [36]
- una cattura da malware-traffic-analysis

Distinto da questi, il dataset CICIoT2023 [28] è stato impiegato unicamente per la calibrazione dei pesi (Sezione 4.8) e non è incluso tra le catture di validazione, così da tenere separata la fase di taratura da quella di misura delle prestazioni.

Una proprietà comune ai dataset di validazione è che ogni cattura di malware è dedicata: contiene tipicamente un singolo host interno infetto insieme al suo traffico, e pochi o nessun altro host interno. Questa caratteristica ha una conseguenza diretta sui risultati, cioè i valori della matrice di confusione restano contenuti, perché il numero di host interni etichettabili per cattura è basso. Proprio per compensare questo, alla validazione sui positivi sono stati affiancati test complementari su catture di soli host normali, in modo da disporre di una classe negativa sufficiente per verificare l'assenza di falsi positivi.

Sono stati selezionati i test annotando gli IP degli host infetti e di quelli normali in modo da poter effettuare un confronto con i risultati; infatti, tutti gli IP possono essere trovati nella documentazione dei siti dove sono stati scaricati i pcap. Gli host di background, il cui stato non è etichettato con certezza, sono stati esclusi dal conteggio anziché considerati negativi, per non introdurre una stima distorta dei risultati. Solo gli host interni rientrano nell'analisi, coerentemente con il principio di direzionalità del modello, mentre attaccante e server di comando e controllo (C2), essendo esterni, sono fuori perimetro per costruzione.

La soglia di decisione adottata come riferimento è quella della fascia gialla, cioè un host è considerato positivo (segnalato) quando il suo punteggio $S(h)$ raggiunge o supera i 30 punti. Questa scelta riflette la struttura del sistema, per cui qualsiasi host in fascia gialla o rossa è ritenuto sospetto. Accanto ad essa viene riportata anche la soglia della fascia rossa ($S(h) \geq 60$, host compromesso), così da mostrare la sensibilità del sistema.

Una precisazione di configurazione riguarda M_{cert}. Tra le anomalie di certificato rilevabili tramite i bit di rischio nDPI, il controllo sul certificato scaduto è stato disattivato nella configurazione usata per la validazione: sulle catture storiche, infatti, certificati validi al momento della registrazione risultano oggi scaduti, generando falsi positivi non rappresentativi di un comportamento malevolo. La metrica resta quindi attiva sui due segnali più affidabili, il certificato autofirmato e quello con firma SHA-1.

6.2 Malware e botnet analizzati

Le famiglie sono state scelte per coprire comportamenti diversi, così che le rilevazioni provengano da metriche diverse e non da un unico segnale. Si riportano di seguito le famiglie, il comportamento osservato e le metriche attivate.

- **Neris** (botnet IRC): la cattura contiene dieci host interni infetti. Le rilevazioni derivano da M_{proto}, per l'uso del protocollo IRC su porte non standard, e da M_{scan}, per la scansione della rete. Tutti e dieci gli host superano la soglia rossa.
- **Dridex** (trojan bancario, C2): la comunicazione verso il server di comando (C2) avviene in TLS diretto verso indirizzi IP, senza indicazione del dominio. Si attivano M_{sni} (assenza di SNI), M_{proto} e M_{scan}, portando l'host in fascia rossa.
- **Trickbot** (trojan bancario, C2): il canale di comando presenta un certificato autofirmato e un protocollo su porta non standard. Si attivano M_{cert} (certificato autofirmato) e M_{proto} (TLS su porta 447).
- **Mirai** (botnet IoT, DDoS e scansione): oltre a M_{proto} e M_{scan}, la cattura ha attivato anche la metrica M_{fail} (si veda la Sezione 6.3).
- **Gafgyt** (botnet IoT, DDoS): il flood in uscita produce un picco volumetrico che ha permesso l'attivazione di M_{vol} (Sezione 6.3), in aggiunta a M_{scan}.
- **Hakai** (botnet IoT, C2): la cattura attiva la sola M_{scan}, collocando l'host in fascia gialla. La metrica M_{rep} resta inattiva, non per assenza di traffico verso il server di comando (C2), ma perché le blacklist correnti non contengono più gli IP blacklistati del 2018 (Sezione 6.7).
- **Trojan** (trojan IoT): analogamente, la sola M_{scan} si attiva e l'host risulta in fascia gialla.
- **IcedID** (trojan bancario): la cattura attiva solo la metrica M_{cert} per via del frequente utilizzo di certificati TLS autofirmati e non attiva la metrica M_{rep} per la ragione già indicata a proposito delle blacklist.
- **Torii** (due catture, C2 *low and slow*): il traffico non presenta certificati TLS irregolari, senza scansione né porte non standard, e gli indirizzi del server in blacklist ormai non sono più presenti. Il sistema non segnala l'host in nessuna delle due catture, producendo due falsi negativi. Questo limite è discusso nella Sezione 6.7.

Tabella 6.1: Catture di malware e botnet, host infetti e metriche attivate.

Cattura	Tipo di minaccia	Host infetti	Fascia	Metriche attivate
Neris	Botnet IRC	10	Rosso	M_{proto}, M_{scan}
Dridex	Trojan bancario (C2)	1	Rosso	M_{sni}, M_{proto}, M_{scan}
Trickbot	Trojan bancario (C2)	1	Rosso	M_{cert}, M_{proto}
Mirai	Botnet IoT (DDoS/scan)	1	Rosso	M_{proto}, M_{scan}, M_{fail}
Gafgyt	Botnet IoT (DDoS)	1	Rosso	M_{scan}, M_{vol}
Hakai	Botnet IoT (C2)	1	Giallo	M_{scan}
Trojan	Trojan IoT	1	Giallo	M_{scan}
IcedID	Trojan bancario (C2)	1	Giallo	M_{cert}
Torii	C2 cifrato low and slow	1	Verde	nessuna (FN)
Torii	C2 cifrato low and slow	1	Verde	nessuna (FN)

6.3 Metriche statistiche e baseline artificiale

Due metriche, l'asimmetria volumetrica M_{vol} e il tasso di fallimento M_{fail}, non si basano su una singola evidenza ma su uno scostamento statistico dal comportamento abituale dell'host, misurato tramite lo $Z_{modified}$ rispetto a una *baseline* storica. Questo tipo di metrica pone un problema specifico nella validazione su pcap dedicati: una cattura di malware, per via dei *dataset* scelti, contiene solo l'attività malevola in una finestra breve e non offre alcuno storico benigno dello stesso host su cui costruire la *baseline*. Senza *baseline*, per costruzione, la metrica non può attivarsi.

Per validare comunque queste due metriche, è stata iniettata una *baseline* a partire dal comportamento reale di un host normale, prelevato da una cattura benigna separata e riallineato con `editcap` nella finestra storica che precede l'ora di riferimento; la Figura 6.1 rappresenta l'operazione effettuata. Il segnale malevolo nella finestra corrente resta traffico reale e non modificato del pcap originale, ovvero solo la *baseline*, con cui viene confrontato, proviene da un comportamento benigno. Nel caso di M_{vol}, ad esempio, la baseline benigna descrive un carico orario in uscita dell'ordine di pochi megabyte, mentre il picco reale del flood del malware nella finestra corrente supera i tre gigabyte, producendo uno scostamento sufficientemente ampio da far scattare la metrica.

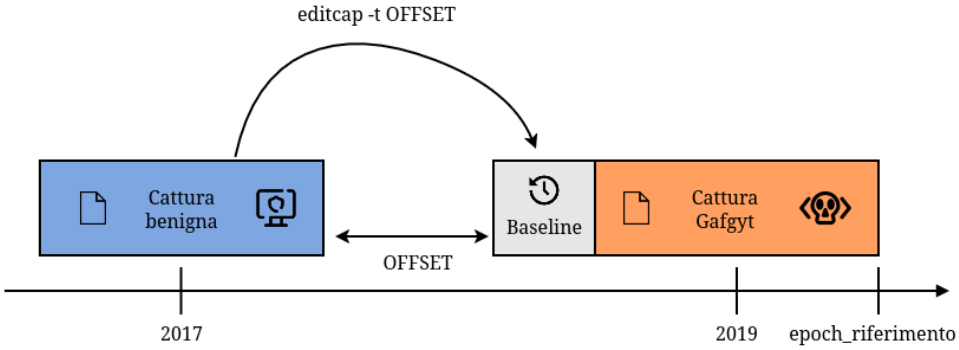

Figura 6.1: Allineamento temporale tramite editcap.

L'unione dei file benigni con la cattura malevola richiede due accorgimenti. Nel primo le metriche confrontano il traffico corrente e la *baseline* dello stesso host tramite una `join` sull'indirizzo IP, quindi la baseline iniettata deve essere attribuita allo stesso indirizzo dell'host infetto, altrimenti il confronto non produce risultati. Nelle catture di laboratorio dei dataset utilizzati questo è naturale, perché il dispositivo riprende spesso lo stesso indirizzo locale; in caso contrario, l'indirizzo sorgente del traffico benigno va riscritto su quello del dispositivo malevolo prima dell'analisi. Il secondo accorgimento riguarda la coerenza temporale: i bucket iniettati devono ricadere nella stessa categoria oraria (feriale lavorativo, feriale fuori orario, weekend) dell'ora di riferimento, e i timestamp vanno spostati in modo coerente.

Questa procedura è un'eccezione metodologica limitata alle due metriche statistiche. Non altera i risultati della matrice di confusione, perché la classificazione degli host come positivi regge anche escludendo il contributo delle metriche a *baseline* iniettata (Mirai resta in fascia rossa con M_{proto} e M_{scan}, Gafgyt resta rilevato in fascia gialla con la sola M_{scan}).

6.4 Host normali e controllo dei falsi positivi

Poiché le catture di malware contengono un solo host interno infetto e pochi host normali, la specificità del sistema, cioè la percentuale delle previsioni negative corrette (TN) sul totale delle istanze negative, non può essere misurata su di esse. Per questo motivo sono state incluse catture dedicate al comportamento di host legittimi, che costituiscono la classe negativa della validazione.

- **Amazon Echo** e **Somfy**: due dispositivi domestici commerciali, un assistente vocale e un attuatore per la casa connessa. Rappresentano lo scenario più severo per i falsi positivi, perché contattano numerosi endpoint cloud, a volte su porte non standard, proprio le condizioni in cui M_{proto}, M_{cert} o M_{sni} potrebbero sbagliare.

- **Catture normali CTU**: traffico di rete etichettato come benigno, privo di attività malevola.
- **Host normali nella cattura di Neris**: gli host legittimi presenti nella stessa rete degli infetti. Sono i negativi più significativi, perché condividono l'ambiente con gli host compromessi e mettono alla prova la capacità del sistema di distinguere gli uni dagli altri all'interno della stessa cattura.

Su tutti questi host il sistema non ha prodotto alcun falso positivo: nessun host normale ha raggiunto la fascia gialla. In particolare, Amazon Echo e Somfy, nonostante l'elevato numero di connessioni verso servizi cloud, sono rimasti in fascia verde, a conferma che le metriche relative al traffico TLS non si attivano sul normale comportamento di dispositivi legittimi.

6.5 Inclusione di CIC-IDS2017

Ai risultati è stata aggiunta la cattura CIC-IDS2017 relativa al traffico del venerdì, che contiene l'attività di una botnet (ARES) e scansioni di porta. Il sistema identifica correttamente tutti e cinque gli host interni compromessi (192.168.10.5, 192.168.10.8, 192.168.10.9, 192.168.10.14 e 192.168.10.15), collocandoli in fascia rossa. Le rilevazioni provengono da M_{cert} (certificati autofirmati), M_{proto} (DCERPC su porte alte non standard) e M_{scan} (SYN scan). L'indirizzo 172.16.0.1, che compare segnalato, non è stato conteggiato, poiché corrisponde all'interfaccia interna del firewall, dietro cui l'attaccante esterno (205.174.165.73) viene tradotto in NAT, quindi non è un host normale e la sua segnalazione non costituisce un falso positivo.

6.6 Matrice di confusione e metriche di performance

La Figura 6.2 riporta la matrice di confusione a soglia gialla, cioè considerando positivo ogni host con punteggio pari o superiore a 30. Il sistema riconosce 22 host infetti su 24, con 2 falsi negativi (i due host Torii) e nessun falso positivo su 13 host normali.

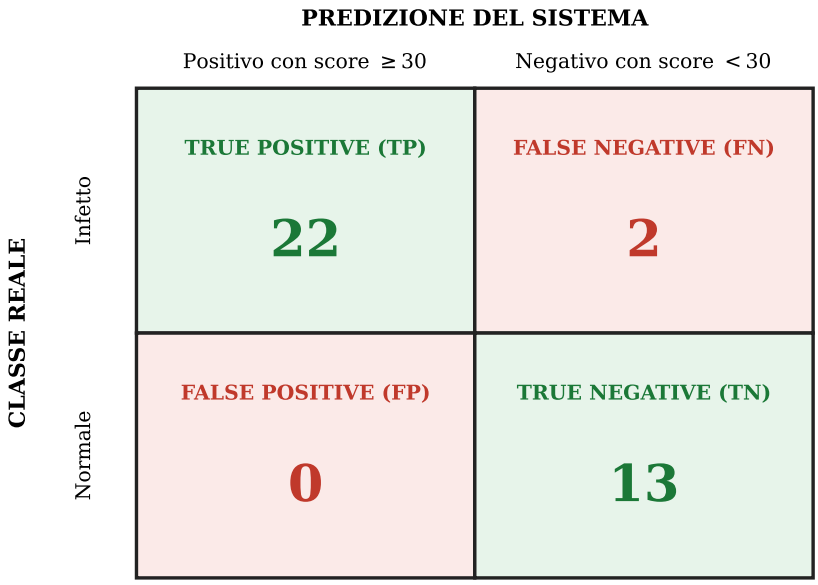

Figura 6.2: Matrice di confusione a soglia gialla (punteggio ≥ 30).

Le metriche di valutazione derivate dalla matrice sono riportate nella Tabella 6.2, calcolate sia alla soglia gialla sia a quella rossa. Per comprendere appieno l'analisi, è opportuno introdurre i quattro parametri della matrice di confusione: un vero positivo (TP) è un host infetto correttamente segnalato, un falso positivo (FP) un host normale erroneamente segnalato, un falso negativo (FN) un host infetto non rilevato (i due Torii), un vero negativo (TN) un host normale correttamente lasciato in fascia verde.

I risultati mostrano una specificità e una precisione del 100%, cioè l'assenza di falsi positivi su tutti i dati benigni, e una recall del 91,7% alla soglia gialla. Alzando la soglia alla fascia rossa la recall scende al 79,2% (Figura 6.3); il calo è dovuto agli host rilevati solo in fascia gialla (Hakai, Trojan e IcedID), che a quella soglia diventano falsi negativi, oltre ai due Torii che non sono stati segnalati.

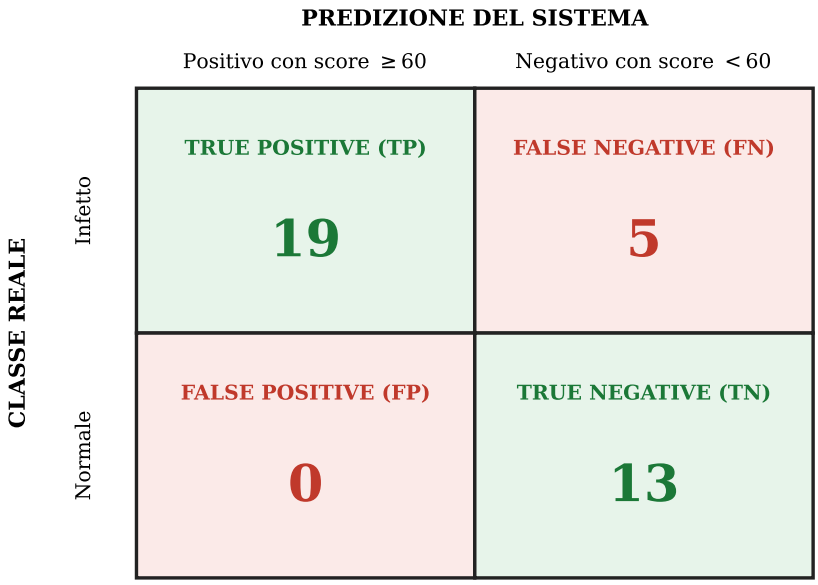

Figura 6.3: Matrice di confusione a soglia rossa (punteggio ≥ 60).

Tabella 6.2: Metriche di valutazione alle due soglie.

Metrica	Soglia Giallo (≥ 30)	Soglia Rosso (≥ 60)
Accuratezza	94,6%	86,5%
Tasso di errore	5,4%	13,5%
Precisione (PPV)	100,0%	100,0%
Recall / sensibilità (TPR)	91,7%	79,2%
Specificità (TNR)	100,0%	100,0%
False Positive Rate (FPR)	0,0%	0,0%
F1-score	95,7%	88,4%

6.7 Limiti osservati

I test hanno reso espliciti alcuni limiti del sistema.

- **C2 low and slow**: le due catture Torii non vengono rilevate. Un canale di comando cifrato, senza scansione né porte anomale né volumi anomali, non offre appiglio alle metriche deterministiche, e rappresenta il confine del modello.
- **Reputazione su dataset storici**: M_{rep} non si attiva sulle catture storiche perché valuta il traffico con le blacklist correnti, che non contengono più gli indirizzi di comando e controllo (C2) di diversi anni fa. Non è un difetto della metrica, ma una conseguenza dell'analisi effettuata su pcap degli anni passati.
- **Contatto con nuove porte server**: M_{srv} dipende da un periodo di apprendimento del profilo abituale dell'host da parte della sonda ntopng, apprendimento che un pcap breve non consente. La metrica resta quindi difficilmente esercitabile su registrazioni di breve durata.
- **Contaminazione della baseline**: le metriche statistiche (M_{vol} e M_{fail}) assumono che lo storico dei sette giorni precedenti rappresenti un comportamento legittimo. Se un host è già compromesso all'inizio dell'osservazione, l'attività malevola entra a far parte della sua baseline e viene appresa come normale, per cui uno scostamento persistente non produce più alcun segnale. Il modello rileva quindi la transizione verso un comportamento anomalo, non uno stato malevolo già stabile e continuativo. È lo stesso motivo per cui, nella validazione su pcap, la baseline benigna è stata iniettata da una cattura separata (Sezione 6.3).
- **Numerosità dei risultati**: la maggior parte delle catture di malware e botnet presenta un solo host infetto, il che limita la dimensione assoluta della matrice di confusione. È questa la ragione dei valori contenuti nei conteggi, compensata dai test complementari sugli host normali descritti nella Sezione 6.4.

6.8 Costo computazionale del modello

Oltre all'accuratezza, un sistema pensato per operare in modo continuativo deve avere un costo di esecuzione compatibile con la frequenza con cui viene lanciato. L'infrastruttura di monitoraggio passivo, basata sulla sonda ntopng e sul database ClickHouse, comporta un costo fisso comune a qualsiasi sistema di analisi del traffico e del tutto indipendente dal modello di scoring. Di conseguenza, l'analisi proposta in questa sezione si concentra esclusivamente sul costo aggiuntivo che il motore di scoring introduce rispetto a tale infrastruttura di base.

Le misure sono state raccolte su un portatile con processore Intel Core i7-1260P (con 16 *thread*), 16 GB di memoria e SSD da 512 GB. Il dataset di misura è quello con maggior numero di flussi tra i test effettuati: la tabella `flows` contiene 80 410 righe,

mentre `host_alerts` raccoglie i soli allarmi prodotti sul traffico analizzato. Poiché il costo delle query dipende dal numero di flussi e non dal volume in byte del traffico catturato, è questa la grandezza rilevante per il motore di scoring. Si tratta quindi di un carico ottenuto dal *replay* di catture su hardware da portatile, e i valori che seguono vanno letti come ordini di grandezza rappresentativi.

Il costo è stato osservato da tre punti di vista complementari. Il tempo di ogni metrica è stato misurato lato client, cronometrando la singola chiamata all'interno del motore. Il costo lato database è stato ricavato dalla tabella `system.query_log` di ClickHouse, che per ogni interrogazione registra durata, righe lette e memoria occupata; per isolare le sole query del motore da quelle prodotte in tempo reale dalla sonda, ogni interrogazione è stata marcata con un'etichetta dedicata (`log_comment`) aggiunta come parametro alla connessione con il database; i valori riportati sono le mediane su undici esecuzioni. Il costo dell'intero processo è stato infine misurato con `/usr/bin/time -v python scoring.py`, mentre la dimensione su disco proviene da `system.parts`.

Tabella 6.3: Tempo di esecuzione per metrica in un ciclo dello scoring.

Metrica	Tipo	Tempo (ms)
M_{rep}	deterministica	7
M_{cert}	deterministica	12
M_{sni}	deterministica	12
M_{srv}	deterministica	7
M_{proto}	deterministica	16
M_{scan}	deterministica	10
M_{vol}	statistica	35
M_{fail}	statistica	35

La Tabella 6.3 mostra che il costo non è distribuito uniformemente. Le sei metriche deterministiche, che leggono allarmi già prodotti dalla sonda, si esauriscono in pochi millisecondi ciascuna. Le due metriche statistiche, M_{vol} e M_{fail}, sono le più costose (35 ms ciascuna), perché ricostruiscono la *baseline* dell'host confrontando l'ora corrente con i sette giorni precedenti. È un risultato atteso, poiché sono le uniche due metriche che eseguono un'aggregazione sullo storico anziché una semplice lettura di allarmi. Lo stesso quadro emerge dal lato database. Le due query statistiche leggono circa 29 900 righe ciascuna con una durata mediana di 33 e 34 ms, mentre le altre sei restano sotto i 20 ms leggendo poche centinaia di righe.

Considerando l'intero ciclo, l'esecuzione completa di `scoring.py` (apertura della connessione, calcolo delle otto metriche, aggregazione per host e stampa del report) richiede 0,28 s di tempo totale, di cui circa 0,13 s di tempo di CPU, con un picco di memoria del processo Python di circa 31 MB. Questo valore riguarda il solo processo di scoring e va tenuto distinto dalla memoria occupata lato ClickHouse,

discussa sopra. Di seguito è possibile vedere in dettaglio la misurazione dei costi dello scoring.

```
/usr/bin/time -v python scoring.py

Command exited with non-zero status 2
Command being timed: "python scoring.py"
User time (seconds): 0.11
System time (seconds): 0.02
Percent of CPU this job got: 49%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.28
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 31524
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 5378
Voluntary context switches: 12
Involuntary context switches: 3
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 2
```

Listing 6.2: Misurazione costi del processo di scoring

Il codice di uscita 2 non indica un errore, ma corrisponde alla codifica della fascia globale rossa (0 verde, 1 giallo, 2 rosso) restituita da `scoring.py` al termine dell'esecuzione.

Considerando invece lo spazio sul disco, gli 80 410 flussi del dataset occupano solo 7,47 MB: questo conferma l'efficacia della compressione colonnare di ClickHouse nel contenere la crescita dello storico, che è la risorsa destinata ad accumularsi nel tempo.

Poiché il motore opera in modalità *batch* orario, un ciclo deve concludersi entro l'ora che lo separa dall'esecuzione successiva. Concludendosi in meno di un secondo su questo carico, il ciclo rispetta il vincolo operativo con oltre tre ordini di grandezza di margine.

Capitolo 7

Conclusioni

Questa tesi nasce da una domanda semplice da porre, ma a cui è difficile rispondere, cioè se una rete possa dirsi sicura in un contesto in cui il volume del traffico e la diffusione della cifratura rendono sempre meno leggibile ciò che accade al suo interno. A partire da questa esigenza è stato progettato e realizzato un sistema di scoring orientato al singolo host, configurato come un IDS *network-based* e *anomaly-based*, basato esclusivamente sulla cattura passiva del traffico, che assegna a ciascun host interno un punteggio di rischio sulla base di otto metriche indipendenti, deterministiche e statistico-comportamentali, riassunte in un unico valore $S(h)$ e in una classificazione a tre fasce.

La validazione condotta nel Capitolo 6 ha mostrato che il sistema riconosce host compromessi appartenenti a famiglie di minaccia molto diverse tra loro, dalle botnet IRC e IoT ai trojan bancari, attivando ogni volta metriche differenti e non un unico segnale. I risultati indicano una precisione del 100%, con l'assenza di falsi positivi anche su dispositivi domestici legittimi dal comportamento di rete complesso, e una recall del 91,7% alla soglia gialla. A questo si aggiunge un costo computazionale trascurabile, con un ciclo completo che si conclude in meno di un secondo e resta quindi ampiamente entro il vincolo dell'esecuzione oraria (Sezione 6.8).

Il contributo principale del lavoro non risiede però nella sola accuratezza, ma nella trasparenza del giudizio prodotto. Come osservato nell'introduzione, gli approcci basati su *machine learning* offrono una buona capacità di generalizzazione ma restituiscono allarmi difficili da giustificare, e il costo di un allarme senza causa comprensibile ricade interamente sull'analista [26]. Il modello proposto affronta questo limite alla radice, perché ogni punteggio è riconducibile a una causa precisa. L'informazione che il sistema fornisce non è soltanto che un host è sospetto, ma perché lo è.

Questa stessa natura deterministica costituisce anche il limite del sistema. Una comunicazione di comando e controllo (C2) di tipo *low and slow*, priva di scansione,

di porte anomale e di picchi volumetrici, non fa scattare nessuna delle metriche e sfugge alla rilevazione, come mostrato dalle due catture Torii rimaste in fascia verde. È un limite atteso e coerente con il sistema sviluppato, ed è proprio da esso che prendono le mosse gli sviluppi futuri descritti di seguito.

7.1 Sviluppi futuri

La prima direzione riguarda il punto cieco emerso in validazione, ovvero le minacce *low and slow*: comunicazioni a bassa intensità e diluite nel tempo per non superare le soglie. Comprendono il *beaconing* periodico verso un server di comando (C2) (come nelle catture Torii) e forme aperiodiche come l'esfiltrazione lenta. Le metriche attuali non osservano la struttura temporale del traffico: una metrica sensibile alla periodicità permetterebbe di riconoscere il *beaconing*, mentre le forme aperiodiche resterebbero un problema aperto.

In questa stessa prospettiva si colloca l'impronta *JA4* del client TLS [4], che caratterizza il modo in cui un host negozia la connessione cifrata a prescindere dal contenuto, e che sarebbe particolarmente utile proprio là dove i segnali basati sul contenuto vengono meno. La sua implementazione come metrica non è stata possibile in questo lavoro non per un ostacolo tecnico al calcolo dell'impronta, ma per la scarsità di riferimenti affidabili contro cui confrontarla, dato che il database pubblico di FoxIO contiene ancora poche firme JA4 di malware e botnet verificate con certezza, insufficienti a costruire una metrica solida. Con il progressivo consolidamento di questi riferimenti, la metrica potrà essere integrata nel sistema.

Un'ulteriore direzione riguarda la reputazione dei contatti. La metrica M_{rep} si è rivelata inefficace sulle catture storiche perché confronta il traffico con blacklist correnti, che non contengono più gli indirizzi di comando e controllo (C2) di anni fa. L'integrazione di feed di *threat intelligence* costantemente aggiornati, in un dispiegamento su rete reale, restituirebbe alla metrica la sua piena efficacia.

Sul piano della validazione, alcune metriche come M_{srv} dipendono da un periodo di apprendimento (*learning period*) del profilo abituale dell'host che una cattura di breve durata non consente di esercitare. Un dispiegamento prolungato su una rete di produzione permetterebbe di validare pienamente questi comportamenti e di osservare il sistema in condizioni operative reali, oltre che di misurarne il costo su volumi di traffico e numeri di host sensibilmente maggiori.

Infine, il confronto con il *machine learning* non va inteso soltanto come una contrapposizione. Un possibile sviluppo consiste in un approccio ibrido, in cui tecniche di apprendimento contribuiscano a calibrare i pesi delle metriche o a segnalare altre anomalie non visibili con questo approccio, mantenendo però la scomposizione interpretabile del punteggio come tratto distintivo del sistema. In questo modo si unirebbe la capacità di generalizzazione del *machine learning* alla trasparenza che ha guidato la progettazione di questo lavoro.

Appendice A

Codice sorgente e documentazione

Il codice sorgente del progetto, comprensivo di script, test e documentazione, è disponibile sul repository GitHub al seguente link:

https://github.com/dmilletti/tesi_milletti

Bibliografia

[1] Donald E. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, January 2011.

[2] Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, and Samuel Madden. The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases*, 5:197–280, 2013.

[3] Kalyan Acharjya, Madhur Grover, Avinash Samantra, Mithhil Arora, Muktha Eti, and Vino T. Application of artificial intelligence and machine learning techniques for network intrusion detection and prevention. In *2025 International Conference on Networks and Cryptology (NETCRYPT)*, pages 1829–1834, 2025.

[4] John Althouse. JA4+ Network Fingerprinting. https://blog.foxio.io/ja4+-network-fingerprinting, 2023. FoxIO.

[5] John B. Althouse, Jeff Atkinson, and Josh Atkins. JA3: A Method for Profiling SSL/TLS Clients. https://github.com/salesforce/ja3, 2017. Salesforce Engineering.

[6] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of tls (without decryption), 2016.

[7] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away traffic to bots: Detecting the rise of DGA-Based malware. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 491–506, Bellevue, WA, August 2012. USENIX Association.

[8] Richard Bejtlich. *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Addison-Wesley Professional, 2004.

[9] Luca Deri and Francesco Fusco. Using deep packet inspection in cybertraffic analysis. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 89–94, 2021.

[10] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless*

Communications and Mobile Computing Conference (IWCMC), pages 617–622, 2014.

[11] Luca Deri, Maurizio Martinelli, and Alfredo Cardigliano. Realtime high-speed network traffic monitoring using ntopng. In *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, LISA'14, page 69–79, USA, 2014. USENIX Association.

[12] Reham Taher El-Maghraby, Nada Mostafa Abd Elazim, and Ayman M. Bahaa-Eldin. A survey on deep packet inspection. In *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, pages 188–197, 2017.

[13] Falih Gozi Febrinanto, Kristen Moore, Chandra Thapa, Mujie Liu, Vidya Saikrishna, Jiangang Ma, and Feng Xia. Entropy causal graphs for multivariate time series anomaly detection. *ACM Transactions on Intelligent Systems and Technology*, 16(6):1–25, October 2025.

[14] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic, January 2020.

[15] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers Security*, 45:100–123, 2014.

[16] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, USA, 2007. USENIX Association.

[17] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.

[18] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[19] Sonam Juneja, Arshdeep, Souvik Maiti, Sneha Raweri, Bhoopesh Singh Bhati, and Harsh Sharma. Comprehensive evaluation of network performance monitoring solutions. In *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*, pages 1–6, 2024.

[20] Salman Khaliq, Zain Ul Abideen Tariq, and Ammar Masood. Role of user and entity behavior analytics in detecting insider attacks. In *2020 International Conference on Cyber Warfare and Security (ICCWS)*, pages 1–6, 2020.

[21] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):20, 2019.

[22] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks*, 109:127–141, 2016. Traffic and Performance in the Big Data Era.

[23] Petr Matoušek, Ondřej Ryšavý, and Ivana Burgetová. Experience report: Using ja4+ fingerprints for malware detection in encrypted traffic. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–5, 2024.

[24] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[25] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018.

[26] Vincent Zibi Mohale and Ibidun Christiana Obagbuwa. A systematic review on the integration of explainable artificial intelligence in intrusion detection systems to enhancing transparency and interpretability in cybersecurity. *Frontiers in Artificial Intelligence*, Volume 8 - 2025, 2025.

[27] Robert Moskowitz, Daniel Karrenberg, Yakov Rekhter, Eliot Lear, and Geert Jan de Groot. Address Allocation for Private Internets. RFC 1918, February 1996.

[28] Euclides Carlos Pinto Neto, Sajjad Dadkhah, Raphael Ferreira, Alireza Zohourian, Rongxing Lu, and Ali A. Ghorbani. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment. *Sensors*, 23(13), 2023.

[29] Subash Neupane, Jesse Ables, William Anderson, Sudip Mittal, Shahram Rahimi, Ioana Banicescu, and Maria Seale. Explainable intrusion detection systems (x-ids): A survey of current methods, challenges, and opportunities, 2022.

[30] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[31] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[32] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. RFC 9849, March 2026.

[33] Maria Andrea Romo-Chavero, Gustavo De Los Ríos Alatorre, Jose Antonio Cantoral-Ceballos, Jesús Arturo Pérez-Díaz, and Carlos Martinez-Cagnazzo. A hybrid model for bgp anomaly detection using median absolute deviation and machine learning. *IEEE Open Journal of the Communications Society*, 6:2102–2116, 2025.

[34] Bruce Schneier. *Beyond Fear: Thinking Sensibly About Security in an Uncertain World*. Copernicus Books (Springer-Verlag), New York, NY, 2003.

[35] Robert Schulze, Tom Schreiber, Ilya Yatsishin, Ryadh Dahimene, and Alexey Milovidov. Clickhouse - lightning fast analytics for everyone. *Proc. VLDB Endow.*, 17(12):3731–3744, August 2024.

[36] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*, 2018.

[37] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of IP flow-based intrusion detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[38] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. MITRE ATT&CK: Design and philosophy. Technical Report MP180360R1, The MITRE Corporation, McLean, VA, 2020.

[39] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70:238–254, 2017.

[40] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

[41] Akila Wickramasekara, M.P.P. Liyanage, and Udayagee Kumarasinghe. A comparative study between the capabilities of mysql and clickhouse in low-performance linux environment. In *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 276–277, 2020.

[42] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, page 199–208, New York, NY, USA, 2013. Association for Computing Machinery.

[43] Binhe Zhang, Hao Yu, and Ying Yan. Ntopng based traffic monitoring and modelling. In *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1305–1310, 2019.