UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITA' DEGLI STUDI DI MODENA E REGGIO EMILIA
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea Magistrale in Informatica

Adversarial Training per ML-NIDS in Problem Space: implementazione e valutazione di un framework di difesa

Relatore:
Prof. Mirco Marchetti

Corelatore:
Ing. Dimitri Galli

Tesi di Laurea di:
Matteo Manocchia

Anno Accademico 2025–2026

Indice

Indice delle figure

Indice delle tabelle

Capitolo 1

Introduzione

L'attività di un'organizzazione moderna, dal lavoro quotidiano dei dipendenti ai rapporti con clienti e fornitori, dai servizi erogati agli utenti finali alle comunicazioni interne fra sedi diverse, si svolge in larga parte attraverso le reti informatiche. Sulle stesse reti viaggiano informazioni di valore — dati personali, comunicazioni riservate, credenziali di accesso, proprietà intellettuale — che costituiscono un bersaglio costante per attori malevoli interessati a sottrarle, manipolarle o renderle indisponibili. Le reti non sono dunque soltanto un'infrastruttura di comunicazione, ma anche una superficie da sorvegliare in modo continuo, distinguendo, fra le molteplici comunicazioni legittime che le attraversano, quelle che indicano un attacco in corso. Questo compito è affidato ai *sistemi di rilevamento delle intrusioni di rete* (Network Intrusion Detection System, NIDS), che osservano il traffico in transito attraverso uno o più segmenti della rete e segnalano agli operatori di sicurezza le attività riconducibili a un attacco.

Per lungo tempo i NIDS hanno operato per riconoscimento di pattern noti, fondandosi su firme di attacco e regole scritte manualmente da analisti di sicurezza. Questo approccio ha progressivamente mostrato i propri limiti di fronte al volume e alla variabilità del traffico contemporaneo: il volume dei dati che attraversano le infrastrutture moderne è tale da rendere l'ispezione manuale sistematica impraticabile, e le tecniche di evasione adottate dagli attori malevoli evolvono con una rapidità che il lavoro di codifica manuale di nuove firme non riesce a sostenere.

Una soluzione a queste limitazioni è giunta dall'*apprendimento automatico* (machine learning, ML): su queste premesse si è sviluppata la categoria dei sistemi di rilevamento delle intrusioni di rete basati su machine learning, indicati con la sigla *ML-NIDS*, in grado di apprendere autonomamente dai dati osservati i pattern che distinguono il traffico legittimo da quello malevolo. I ML-NIDS rappresentano oggi una componente centrale delle architetture di sicurezza di rete.

L'integrazione di componenti basati su machine learning ha esteso in modo significativo le capacità di rilevamento di questi sistemi, ma ne ha aperto al tempo stesso una vulnerabilità di natura nuova rispetto ai NIDS tradizionali: gli *adversarial attack*. Si tratta

di attacchi in cui un avversario, modificando in modo mirato il traffico di rete che genera, induce il classificatore a una classificazione errata, ottenendo così che il proprio traffico malevolo venga etichettato come benigno e prosegua indisturbato all'interno della rete monitorata. La gravità di questa vulnerabilità non ha carattere puramente teorico: una linea di ricerca ormai consolidata ha documentato come perturbazioni anche minime del traffico, applicate con criteri sistematici, siano sufficienti a degradare in modo sostanziale la capacità di rilevamento di un ML-NIDS, anche di sistemi che su traffico ordinario presentano prestazioni considerate elevate. Le prestazioni di rilevamento di un ML-NIDS, in assenza di una valutazione esplicita in scenari adversariali, possono pertanto restituire un'immagine ottimistica della sua tenuta una volta esposto a un avversario reale.

Di fronte a questa vulnerabilità, la ricerca ha proposto diverse strategie difensive. Un'osservazione importante riguarda però il livello al quale tali strategie operano. La maggior parte degli approcci esistenti agisce su quello che la letteratura definisce *feature space*, ovvero sulla rappresentazione numerica interna che il rilevatore costruisce a partire dal traffico osservato: in questo spazio le perturbazioni sono analiticamente trattabili e le difese sono più facili da formulare. Rimane invece meno presidiata la prospettiva complementare, quella del *problem space*, in cui le difese vengano costruite a partire da manipolazioni del traffico di rete a livello di pacchetto, su un piano che riproduce le condizioni in cui un avversario reale effettivamente opera. La distinzione non è puramente tassonomica: un attaccante che voglia eludere un ML-NIDS in produzione agisce sui pacchetti che genera, non sulle rappresentazioni interne del rilevatore, e una difesa validata esclusivamente sul piano delle feature offre garanzie che non si trasferiscono in modo automatico al piano operativo.

È in questa zona della letteratura che si colloca il presente lavoro. La tesi propone un framework difensivo basato su *adversarial training* in *problem space*, una strategia che agisce sul processo di apprendimento del classificatore esponendolo, già in fase di addestramento, a esempi adversariali del tipo di quelli che potrebbe incontrare in fase operativa. La specificità dell'approccio adottato risiede nel modo in cui tali esempi vengono costruiti: anziché perturbare il vettore di feature, le perturbazioni vengono applicate direttamente al traffico di rete a livello di pacchetto, nel rispetto dei vincoli protocollari del dominio, e gli esempi adversariali così ottenuti vengono utilizzati per addestrare modelli più robusti.

L'obiettivo del lavoro è duplice: verificare se questa forma di adversarial training conferisca al rilevatore una robustezza misurabile contro traffico adversarial in fase di test, e accertare che tale robustezza non comporti una regressione delle prestazioni del modello su traffico ordinario. La validazione empirica poggia su un dataset di riferimento per la ricerca su botnet e ML-NIDS e su un classificatore di larga adozione in letteratura, con una metodologia di confronto sistematico fra modelli ordinari e modelli sottoposti ad adversarial training. Le scelte tecniche che danno corpo a questo schema, dalle specifiche

tipologie di perturbazione alle rappresentazioni del traffico considerate, sono oggetto dei capitoli successivi.

L'organizzazione della tesi riflette questo arco di lavoro. Il Capitolo 2 costruisce il quadro di riferimento teorico e bibliografico, descrivendo l'architettura dei ML-NIDS, la struttura degli adversarial attack rivolti contro di essi e le principali strategie difensive proposte in letteratura. Il Capitolo 3 formalizza il modello di minaccia adottato e introduce, a livello concettuale, il framework di adversarial training proposto. Il Capitolo 4 ne descrive l'implementazione, dalla costruzione del dataset di lavoro alle scelte sul classificatore e sulla generazione delle perturbazioni applicate al traffico. Il Capitolo 5 presenta gli scenari di valutazione, le metriche adottate e i risultati ottenuti, sia su traffico ordinario sia su traffico adversarial, articolati lungo il confronto fra modelli baseline e modelli sottoposti ad adversarial training. Il Capitolo 6, infine, ricapitola le indicazioni emerse dal lavoro, ne discute i limiti e delinea le direzioni di estensione.

Capitolo 2

Background e Stato dell'arte

L'obiettivo del seguente capitolo è quello di costruire il quadro di riferimento teorico e bibliografico su cui si fonda il lavoro. Il Capitolo 1 ha introdotto il problema e motivato l'approccio adottato; le sezioni successive approfondiscono i concetti e la letteratura necessari a comprendere le scelte metodologiche adottate.

Il capitolo è organizzato attorno a tre nuclei tematici, tra loro collegati. La Sezione 2.1 descrive l'architettura e la tassonomia dei sistemi di rilevamento delle intrusioni e le implicazioni che derivano dall'impiego di modelli di machine learning in questo contesto. La Sezione 2.2 introduce gli attacchi contro questi sistemi, con particolare attenzione agli *adversarial attack*; ne analizza la struttura e ne evidenzia i rischi operativi: si tratta del *problem statement* che motiva l'intera linea di ricerca. La Sezione 2.3 esamina le principali strategie difensive proposte in letteratura, con focus sull'adversarial training, distinguendo tra approcci in *feature space* e approcci in *problem space*, distinzione centrale per il lavoro descritto nei capitoli successivi.

2.1 NIDS & ML-NIDS

2.1.1 Sistemi di rilevamento delle intrusioni di rete

La sicurezza di un sistema informatico si articola tradizionalmente attorno a tre fasi distinte: prevenzione, rilevamento e reazione [9]. La prevenzione completa di qualsiasi minaccia è riconosciuta come un obiettivo irraggiungibile nella pratica, e la reazione presuppone per definizione che il danno si sia già verificato. Il rilevamento occupa pertanto un ruolo strategico, poiché è la fase in cui le anomalie vengono identificate prima che le loro conseguenze diventino irreversibili.

A svolgere questo compito nelle reti informatiche sono i *sistemi di rilevamento delle intrusioni* (Intrusion Detection System, IDS). Per comprendere concretamente a cosa servono, è utile un parallelismo: così come un sistema di allarme e videosorveglianza presidia un edificio monitorando sensori e telecamere per rilevare comportamenti anomali,

come una porta forzata o un movimento sospetto, un IDS presidia una rete informatica monitorando il traffico e le attività dei sistemi per individuare comportamenti che possano indicare una compromissione.

All'interno della famiglia degli IDS si distinguono due grandi categorie, in base al perimetro di osservazione. Un *Host Intrusion Detection System* (HIDS) monitora l'attività di un singolo host, tra cui processi in esecuzione, file di sistema e log applicativi, ed è pertanto limitato a ciò che avviene sulla macchina su cui è installato. Un *Network Intrusion Detection System* (NIDS), invece, osserva il traffico che transita attraverso uno o più segmenti di rete, analizzando le comunicazioni tra host in modo indipendente dai sistemi endpoint [8, 9]. Un NIDS può essere collocato in diversi punti dell'infrastruttura, come il perimetro, segmenti interni o zone a ridosso di asset critici, e la sua copertura dipende dalla posizione e dalla quantità di traffico che riesce a osservare. Il presente lavoro si colloca interamente nell'ambito dei NIDS, in quanto la strategia di difesa proposta opera su traffico di rete e non su eventi locali di sistema.

Metodi di rilevamento: misuse-based e anomaly-based. Il modo in cui un NIDS decide se un'attività è lecita o malevola dipende dall'approccio di rilevamento adottato [9, 8]. Sono due i paradigmi principali, ciascuno con caratteristiche, vantaggi e limitazioni proprie.

I sistemi *misuse-based*, detti anche *signature-based* o basati su regole, operano per riconoscimento di pattern noti. L'idea di fondo è semplice: ogni attacco informatico ha delle determinate caratteristiche e lascia delle tracce, e queste possono essere codificate come *firme* (signature). Il sistema confronta continuamente il traffico osservato con le firme disponibili: se viene trovata una corrispondenza, viene emesso un alert. Questo approccio ha radici consolidate: i NIDS basati su firme sono stati i primi sistemi commerciali adottati su larga scala, e garantisce precisione elevata e quasi assenza di falsi positivi per gli attacchi già noti [3]. Il limite fondamentale è però altrettanto evidente: una firma può essere scritta solo per un attacco già osservato e catalogato. Varianti inedite di minacce esistenti, nuovi vettori di attacco, o tecniche specificamente progettate per aggirare le firme note sono, per definizione, invisibili a questi sistemi. Il processo di aggiornamento manuale delle signature richiede competenze specialistiche, è soggetto a ritardi e introduce un inevitabile intervallo di vulnerabilità tra la comparsa di una nuova minaccia e la disponibilità della firma corrispondente [9].

I sistemi *anomaly-based* adottano una filosofia complementare. Anziché cercare corrispondenze con pattern di attacco noti, costruiscono prima un modello del comportamento normale, detto *baseline*, della rete che monitorano. Ogni deviazione significativa dalla baseline viene trattata come potenzialmente sospetta e segnalata. In un contesto pratico, ad esempio in una rete aziendale, il comportamento normale è caratterizzato da volumi di traffico prevedibili, protocolli ricorrenti, fasce orarie di attività tipiche, profili di comu-

nicazione stabili tra host: una botnet che inizia a comunicare con server di comando e controllo remoti, inondando la rete di traffico anomalo o stabilendo connessioni inusuali, introduce pattern estranei alla baseline e diventa visibile al rilevatore [8, 3]. Il vantaggio principale dell'approccio anomaly-based è la capacità potenziale di rilevare attacchi non ancora catalogati: se il comportamento è anomalo rispetto alla baseline, il sistema può segnalarlo anche in assenza di una firma corrispondente. Il prezzo da pagare è un tasso di falsi positivi strutturalmente più elevato, poiché non ogni deviazione dalla baseline corrisponde a un'intrusione reale. Infatti, picchi di traffico legittimi, nuovi servizi introdotti in rete, variazioni stagionali del comportamento degli utenti possono tutti produrre alert ingiustificati [9].

I due approcci non si escludono reciprocamente: soluzioni moderne tendono a combinarli, sfruttando la precisione dei sistemi basati su firma per le minacce note e la flessibilità dell'approccio anomaly-based per quelle inedite [8].

Dall'analisi manuale all'apprendimento automatico. Il limite condiviso da entrambi gli approcci tradizionali è la dipendenza dal lavoro manuale da parte di un operatore. Scrivere firme accurate, costruire modelli di baseline affidabili, aggiornare le knowledge base al ritmo con cui le minacce evolvono, richiede un impegno operativo che scala rapidamente, rendendo l'approccio sempre meno sostenibile man mano che gli attacchi si evolvono e che le infrastrutture crescono. A questo si aggiunge la crescita esponenziale del volume e della complessità del traffico nelle reti moderne: le infrastrutture attuali generano quantità di dati che rendono l'ispezione manuale sistematica semplicemente impraticabile [7, 9].

La risposta a queste sfide è arrivata dall'apprendimento automatico (machine learning, ML). Anziché utilizzare la codifica esplicita di regole da parte di esperti, i metodi di ML apprendono automaticamente i pattern rilevanti direttamente dai dati, costruendo modelli in grado di generalizzare su istanze non viste in fase di addestramento. Nella pratica, questo significa che un sistema basato su ML può apprendere sia le caratteristiche del traffico malevolo noto (paradigma misuse-based) sia le caratteristiche del traffico benigno, segnalando le deviazioni (paradigma anomaly-based), senza che un analista debba definire manualmente ogni regola o soglia [9, 8]. I vantaggi rispetto agli approcci tradizionali sono molteplici: la riduzione del carico operativo sugli analisti, la capacità di identificare segnali deboli non percepibili all'occhio umano, la possibilità di aggiornare i modelli riaddestrando il sistema su nuovi dati piuttosto che riscrivendo regole, e, in alcuni scenari, performance di rilevamento superiori rispetto ai sistemi basati su firma [7].

È attorno a queste premesse che si è sviluppata la categoria dei *ML-NIDS*, oggetto del nostro lavoro.

2.1.2 I ML-NIDS: architettura e funzionamento

Un *ML-NIDS* è un sistema di rilevamento delle intrusioni di rete che include, tra i propri componenti, almeno un modello di machine learning [8]. La definizione è volutamente ampia. Essa non prescrive quale algoritmo di ML venga utilizzato, né su quale tipo di dati il sistema operi. Ciò che li caratterizza è la presenza di un componente che ha *appreso* il proprio comportamento dai dati, piuttosto che averlo ricevuto sotto forma di regole scritte a mano da un analista. Per capire concretamente come un ML-NIDS funziona, è utile seguire il percorso che compie il traffico di rete dal momento in cui transita attraverso la rete fino al momento in cui il sistema produce una decisione. Questo percorso si articola in una sequenza di trasformazioni, spesso descritta come *pipeline*, che porta il dato grezzo a una forma che il modello di ML è in grado di elaborare.

Dal traffico grezzo alle feature. Il punto di partenza è il traffico di rete nella sua forma più elementare, ovvero sequenze di pacchetti che viaggiano tra host, ciascuno dei quali contiene informazioni sulla destinazione della comunicazione e sul suo contenuto. Analizzare il traffico a questo livello di granularità, ispezionando ogni singolo pacchetto e in particolare il suo contenuto, è stato per lungo tempo l'approccio dominante nei sistemi di rilevamento basati su firma.

Si tratta però di un metodo che presenta due criticità importanti. La prima è di natura computazionale, poiché il volume di pacchetti che attraversa una rete moderna è tale che l'ispezione sistematica di ciascuno di essi richiede risorse significative. La seconda è strutturale, poiché con la diffusione massiccia della cifratura del traffico Internet, il contenuto della maggior parte delle comunicazioni è opaco, e un NIDS che lavori sul payload non può semplicemente aprirlo e leggerlo [3, 8].

La risposta a queste limitazioni è stato lo spostamento verso un livello di astrazione superiore, quello dei *flussi di rete*. Anziché analizzare ogni singolo pacchetto, un sistema basato su flussi raggruppa le comunicazioni in unità logiche più sintetiche, chiamate appunto flussi o *NetFlow*. Un NetFlow non contiene il payload della comunicazione, ma ne riassume le caratteristiche statistiche, tra cui il numero di pacchetti scambiati, i byte trasmessi in ciascuna direzione, la durata della connessione, le porte e i protocolli utilizzati e la distribuzione temporale dei pacchetti. È una sorta di sommario della comunicazione, che cattura il *comportamento* dello scambio senza esporne il contenuto. Questo approccio è molto più economico da calcolare, non soffre del problema della cifratura, e, come ha dimostrato in modo sistematico la letteratura, contiene informazioni sufficienti a distinguere traffico benigno da traffico malevolo con elevata accuratezza [8, 3].

Per queste ragioni, l'analisi dei NetFlow è oggi la modalità prevalente nei ML-NIDS, sia nella ricerca che nelle implementazioni pratiche [8], ed è l'approccio adottato nel presente lavoro.

Il ruolo del modello di machine learning. Una volta che il traffico è stato trasformato in una rappresentazione numerica, ovvero un vettore di valori che descrivono le caratteristiche di ciascun flusso, il modello di ML entra in gioco.

Il principio fondamentale su cui si basa qualsiasi modello di ML è che esso non riceve regole esplicite, ma le *apprende* a partire da esempi. Questo processo di apprendimento, chiamato *addestramento*, consiste nell'esporre il modello a una collezione di esempi, nel caso di un ML-NIDS flussi di rete, e nel fargli ricavare autonomamente quali caratteristiche distinguono le due categorie [9, 8]. Il risultato di questo processo è un classificatore che, dato un nuovo flusso, mai visto in fase di allenamento, è in grado di produrre una predizione sulla sua natura.

In fase operativa, il classificatore viene applicato in modo continuo al traffico che transita nella rete. Per ogni nuovo flusso osservato, il sistema estrae le feature, costruisce il vettore numerico corrispondente, e lo sottopone al modello. Se la predizione è "malevolo", viene generato un alert che richiede l'attenzione dell'operatore di sicurezza. L'intera pipeline, dalla cattura del traffico all'estrazione delle feature fino alla classificazione e alla generazione dell'alert, costituisce l'architettura operativa di un ML-NIDS [8].

Il presente lavoro si colloca in questo schema. Il sistema analizzato opera su flussi NetFlow estratti da traffico reale, e il componente di classificazione è un modello di ML addestrato su specifici dati. I dettagli implementativi di questa scelta sono descritti nei capitoli successivi.

2.1.3 Tassonomia dei ML-NIDS

Non tutti i ML-NIDS sono uguali. Pur condividendo l'architettura di base descritta nella sezione precedente, che comprende traffico di rete, estrazione di feature e modello di classificazione, i sistemi esistenti si differenziano in modo sostanziale per il modo in cui il modello apprende come distinguere traffico benigno da traffico malevolo.

La distinzione fondamentale, riconosciuta trasversalmente in letteratura, è quella tra approcci *supervisionati* e approcci *non supervisionati* [9, 8].

Apprendimento supervisionato. Nell'apprendimento supervisionato, il modello viene addestrato su un insieme di esempi per cui è già nota la risposta corretta. Nel contesto di un ML-NIDS, questo significa disporre di una collezione di flussi di rete ciascuno dei quali è già stato etichettato da un operatore o da una fonte autorevole come benigno o malevolo. Il modello apprende, a partire da questi esempi etichettati, quali caratteristiche del flusso sono associate a ciascuna delle due categorie, costruendo una superficie di separazione tra di esse. Una volta completato l'addestramento, il classificatore è in grado di assegnare un'etichetta a qualunque nuovo flusso, basandosi su ciò che ha appreso dagli esempi passati [9, 8].

Il vantaggio principale di questo approccio è la precisione: sapendo esattamente quali esempi appartengono a quale classe, il modello può imparare in modo mirato e tendenzialmente produce classificatori accurati su dati simili a quelli su cui è stato addestrato.

Il limite speculare è la dipendenza dalla disponibilità di dati etichettati. Produrre etichette accurate per il traffico di rete richiede competenze specialistiche e, in molti contesti operativi reali, non è un'operazione banale, poiché capire se un determinato flusso di rete osservato in produzione sia effettivamente malevolo o benigno può richiedere un'analisi approfondita da parte di un esperto di sicurezza [8]. Nonostante questa limitazione, l'apprendimento supervisionato è la modalità prevalente nella letteratura sui ML-NIDS, in quanto i dataset pubblici disponibili per la ricerca forniscono già etichette affidabili per ciascun flusso, rendendo l'approccio supervisionato direttamente applicabile [8, 3].

Apprendimento non supervisionato. L'apprendimento non supervisionato non richiede che i dati di addestramento siano etichettati. Anziché imparare la differenza tra due classi a partire da esempi noti, il modello apprende la struttura interna dei dati e identifica raggruppamenti o pattern ricorrenti senza che nessun supervisore gli dica in anticipo cosa cercare. Nel contesto dei ML-NIDS, questo si traduce tipicamente in approcci di tipo anomaly-based: il modello costruisce una rappresentazione del comportamento normale del traffico, apprendendo i pattern di un flusso benigno senza etichettarlo esplicitamente come tale, e segnala come sospetto tutto ciò che si discosta in modo significativo da quella rappresentazione [9, 8].

Il vantaggio di questo approccio è evidente: non richiedendo etichette, può essere applicato anche in contesti in cui i dati di training non sono stati annotati, e in linea di principio è in grado di rilevare attacchi mai visti in precedenza, purché si manifestino come anomalie rispetto al comportamento normale della rete.

Tuttavia, come discusso nella sezione 2.1.1, la stessa caratteristica che rende gli approcci anomaly-based potenzialmente capaci di rilevare attacchi inediti, li rende anche soggetti a un tasso di falsi positivi più elevato, poiché non tutto ciò che si discosta dalla normalità è un attacco, e distinguere le due cose senza etichette di riferimento è strutturalmente più difficile [9, 8].

Supervised e unsupervised: una distinzione ortogonale al tipo di rilevamento. Vale la pena chiarire un punto che nella letteratura genera talvolta confusione. La distinzione tra apprendimento supervisionato e non supervisionato non coincide con quella tra approcci misuse-based e anomaly-based introdotta nella sezione 2.1.1. Le due dimensioni possono essere sovrapposte: un modello supervisionato può essere addestrato sia per riconoscere pattern di attacco noti (misuse-based), sia per apprendere la nozione di normalità a partire da esempi etichettati come benigni (anomaly-based supervisionato); analogamente, un modello non supervisionato può tentare di individuare cluster corrisponden-

ti ad attacchi noti, oppure modellare la normalità senza alcun riferimento esplicito alle classi [9, 8].

La distinzione rilevante ai fini del presente lavoro è quella tra supervisionato e non supervisionato, poiché determina il tipo di dati necessari all'addestramento e le garanzie che il sistema può offrire. Nel presente lavoro si adotta un approccio *supervisionato*: il classificatore viene addestrato su flussi NetFlow etichettati, provenienti dal dataset CTU-13, che fornisce annotazioni affidabili per il traffico sia benigno sia malevolo. La scelta è coerente con la letteratura prevalente nel dominio dei ML-NIDS [8, 3] e permette un confronto diretto con i lavori correlati che adottano lo stesso paradigma. I dettagli relativi al dataset, alla costruzione del training set e alla configurazione del classificatore sono rinviati ai successivi capitoli.

2.1.4 Limiti e sfide dei ML-NIDS

Le sezioni precedenti hanno delineato il funzionamento dei ML-NIDS e le ragioni per cui l'adozione di modelli di machine learning rappresenta un passo avanti rispetto agli approcci tradizionali. Tuttavia, sarebbe scorretto presentare i ML-NIDS come una soluzione priva di criticità. La letteratura ha documentato in modo sistematico un insieme di limiti che, se ignorati, possono portare a sopravvalutare la robustezza reale di questi sistemi una volta che vengono deployati in un contesto operativo [8, 7].

Il primo limite riguarda il processo con cui i ML-NIDS vengono tipicamente sviluppati e valutati nella ricerca. Lo schema standard prevede di raccogliere un dataset di traffico etichettato, suddividerlo in una parte usata per addestrare il modello e una parte usata per misurarne le prestazioni, e riportare le metriche ottenute su quest'ultima. Questo approccio è metodologicamente corretto, ma presenta una limitazione intrinseca: le prestazioni misurate in laboratorio, su un dataset fisso e controllato, non garantiscono che il modello si comporti allo stesso modo una volta esposto al traffico reale di una rete in produzione. Il traffico di rete varia nel tempo, cambia con l'introduzione di nuovi servizi e dispositivi, e dipende in modo sostanziale dalle caratteristiche specifiche dell'ambiente in cui il sistema viene deployato. Un modello addestrato su un dataset raccolto in un determinato contesto può perdere significativa capacità discriminativa quando applicato a un contesto diverso, anche se le prestazioni in laboratorio erano state eccellenti [8]. Questo fenomeno, noto come *gap tra ricerca e pratica*, è uno dei problemi aperti più discussi nella comunità scientifica che si occupa di ML-NIDS.

Il secondo limite, strettamente connesso al primo, riguarda la valutazione della robustezza. La grande maggioranza dei lavori che propongono nuovi ML-NIDS li valuta esclusivamente su traffico non perturbato: il modello viene addestrato su flussi benigni e malevoli nella loro forma naturale, e testato sugli stessi. Questo tipo di valutazione non dà però alcuna indicazione su come il sistema si comporterebbe di fronte a un avversario

che modifica intenzionalmente il proprio traffico per sfuggire al rilevamento. Un modello che ottiene metriche elevate su traffico pulito non è necessariamente in grado di mantenere le stesse prestazioni quando il traffico malevolo viene alterato in modo mirato [8, 3]. La questione non è di secondaria importanza: un sistema di rilevamento delle intrusioni opera precisamente in un contesto in cui esiste un avversario motivato a eluderlo, e valutarlo ignorando questa possibilità equivale a misurarne le prestazioni in condizioni che difficilmente si verificheranno nella realtà operativa.

È proprio questa seconda categoria di limiti che costituisce il *problem statement* del presente lavoro. La domanda a cui si intende rispondere non è se un ML-NIDS possa raggiungere prestazioni elevate su traffico pulito, questione a cui la letteratura ha già risposto affermativamente, ma se e come sia possibile costruire un sistema che mantenga le proprie capacità di rilevamento anche in presenza di traffico malevolo deliberatamente modificato per ingannarlo. Questa vulnerabilità strutturale dei ML-NIDS agli attacchi adversarial, e le strategie per contrastarla, sono l'oggetto delle sezioni successive.

2.2 Adversarial attack contro i ML-NIDS

2.2.1 Definizione e contestualizzazione

Come illustrato nel paragrafo precedente, i ML-NIDS comportano vantaggi significativi rispetto agli approcci tradizionali, poiché automatizzano l'analisi del traffico, generalizzano su minacce non esplicitamente conosciute e possono essere riaddestrati con un flusso di lavoro relativamente semplice [7, 9]. Questi vantaggi hanno reso i ML-NIDS una componente sempre più presente nelle infrastrutture di sicurezza moderne. Tuttavia, l'adozione di modelli di machine learning espone i sistemi di rilevamento a una classe di minacce che i NIDS tradizionali non devono affrontare, gli *adversarial attack*.

Un **adversarial attack** è un attacco in cui un avversario introduce perturbazioni intenzionali nell'input di un modello di machine learning, al fine di indurre una classificazione errata. Con *perturbazione* si intende una modifica dell'input, tipicamente piccola e mirata, che non altera la natura sostanziale del dato dal punto di vista di un osservatore esterno, ma che è sufficiente a far cambiare la predizione del modello. Nel contesto dei ML-NIDS, il risultato è che traffico malevolo, opportunamente modificato, viene classificato come benigno, consentendo all'attaccante di operare indisturbato all'interno della rete monitorata [3, 6].

Per comprendere perché questa minaccia sia concreta, è utile considerare la struttura di fondo di un ML-NIDS. Il sistema osserva il traffico di rete e ne estrae una rappresentazione numerica, a partire dalla quale il modello ha imparato, durante la fase di addestramento, a separare il traffico benigno da quello malevolo. Un attaccante che voglia eludere il rilevamento, non deve necessariamente compromettere il sistema o conoscerne

i dettagli interni, gli è sufficiente modificare il traffico malevolo da lui generato in modo da renderlo sufficientemente simile, secondo la metrica appresa dal modello, al traffico benigno. È questo il principio alla base degli adversarial attack contro i ML-NIDS, e la sua efficacia è stata documentata in modo sistematico dalla letteratura [3, 4].

Una distinzione fondamentale, trasversale all'intera letteratura sugli attacchi avversariali contro i ML-NIDS, riguarda il *livello* al quale le perturbazioni vengono introdotte, e ha implicazioni dirette sulla realizzabilità concreta dell'attacco. Pertanto distinguiamo attacchi in **feature space** ed attacchi in **problem space**.

Gli **attacchi in feature space** operano direttamente sulla rappresentazione interna che il ML-NIDS costruisce a partire dal traffico osservato, dove ogni flusso di rete viene tradotto in un insieme di valori numerici che descrivono le caratteristiche della comunicazione, ed è su questi valori che l'attaccante interviene, modificandoli in modo da far classificare erroneamente il sample al modello. Questo approccio è stato ampiamente adottato nella letteratura su altri domini, in particolare nella visione artificiale, dove tecniche come il Fast Gradient Sign Method(FGSM) [13] hanno mostrato come perturbazioni minime sui pixel di un'immagine possano ingannare classificatori sofisticati, e ha il vantaggio di essere analiticamente trattabile, poiché le feature sono direttamente accessibili e manipolabili. Nel dominio dei ML-NIDS, tuttavia, questo approccio risulta di scarsa rilevanza per la valutazione della robustezza reale di un sistema, poiché presuppone che l'attaccante possa intervenire direttamente sulle feature interne del rilevatore, ovvero che conosca quali feature vengono estratte, come vengono calcolate e con quale parametrizzazione. In uno scenario operativo reale, questa conoscenza tipicamente non è disponibile, poiché un attaccante che genera traffico malevolo sulla rete agisce nel mondo fisico dei pacchetti, non nello spazio astratto delle feature interne del rilevatore [3, 10]. Un modello valutato esclusivamente contro attacchi in feature space offre pertanto garanzie di robustezza che difficilmente si traducono in protezione reale una volta che il sistema viene deployato.

Gli **attacchi in problem space** superano questo limite operando direttamente sul traffico di rete, a livello di pacchetto. L'attaccante modifica il traffico che genera, ad esempio alterando campi di intestazione, aggiungendo dati al payload, o modificando la temporizzazione dei pacchetti, in modo da alterare la rappresentazione numerica che il sistema estrarrà e, di conseguenza, la classificazione che produrrà.

Le perturbazioni devono tuttavia rispettare i vincoli del dominio, per cui il traffico risultante deve essere *protocol-compliant*, deve preservare la funzionalità malevola della comunicazione originale, e deve risultare indistinguibile da traffico legittimo agli occhi del rilevatore, costituendo a tutti gli effetti un oggetto valido nel problem space [3, 11, 10]. Va notato che questo vincolo non è uniforme tra contesti applicativi diversi; ad esempio, i protocolli dei sistemi industriali e OT presentano requisiti strutturalmente più stringenti rispetto ai protocolli IT tradizionali. Nel caso specifico del Modbus TCP, la

conformità richiede che l'ordine richiesta/risposta tra client e server venga preservato e che i messaggi rispettino i vincoli dimensionali imposti dalla specifica del protocollo [12]. Perturbazioni che non rispettano questi vincoli possono produrre pacchetti malformati che, pur modificando le feature estratte dal classificatore, sarebbero rigettati dai dispositivi della rete industriale o immediatamente rilevabili tramite ispezione del traffico.

Indipendentemente dal contesto applicativo, la distinzione tra feature space e problem space non è dunque meramente tassonomica, poiché determina se le vulnerabilità identificate in fase di analisi corrispondono a minacce che il sistema potrà effettivamente incontrare nel proprio ciclo di vita operativo. Prima di esaminare le evidenze empiriche di questa vulnerabilità, è opportuno inquadrare più precisamente la tipologia degli attacchi considerati, attraverso una tassonomia dei principali criteri di classificazione.

2.2.2 Tassonomia degli attacchi adversarial

Gli adversarial attack non costituiscono una categoria monolitica, poiché esistono scenari profondamente diversi tra loro, che si distinguono per le modalità con cui l'attaccante interviene sul sistema e per le risorse e le conoscenze di cui dispone. Comprendere queste distinzioni è importante non solo per classificare gli attacchi descritti in letteratura, ma soprattutto per valutare correttamente la rilevanza pratica di ciascuno scenario. Un attacco che richiede condizioni difficilmente verificabili nella realtà offre indicazioni limitate su quanto un sistema sia robusto nel suo effettivo contesto di deployment. In questa sezione introduciamo i due criteri di classificazione secondo i quali gli attacchi adversarial contro i ML-NIDS vengono tipicamente inquadrati [6, 3].

Fase dell'attacco: poisoning ed evasion. Il primo criterio riguarda il *momento* in cui l'attaccante interviene rispetto al ciclo di vita del modello. Un ML-NIDS attraversa due fasi distinte, ovvero una fase di *addestramento*, in cui il modello apprende come distinguere traffico benigno da traffico malevolo a partire dai dati di training, e una fase di *inferenza*, in cui il modello addestrato viene utilizzato per classificare nuovo traffico in tempo reale. Un attaccante può, in linea di principio, tentare di interferire con entrambe.

Gli attacchi di tipo *poisoning* prendono di mira la fase di addestramento. L'obiettivo è alterare i dati su cui il modello viene addestrato, introducendo campioni malevoli etichettati come benigni o modificando campioni esistenti, in modo da compromettere il comportamento del modello prima ancora che venga messo in produzione. Un attacco di poisoning riuscito può rendere il modello sistematicamente incapace di rilevare determinate categorie di minacce, o addirittura condizionarlo a rispondere in modo prevedibile a stimoli specifici [6]. Questo tipo di attacco richiede però che l'attaccante abbia accesso alla pipeline di addestramento del sistema, ossia ai dati di training, al processo di etichet-

tatura, o al meccanismo di aggiornamento del modello, una condizione che presuppone un livello di compromissione dell'infrastruttura del difensore non sempre realistico.

Gli attacchi di tipo *evasion* operano invece nella fase di inferenza, quando il modello è già addestrato e in produzione. L'attaccante non altera il modello né i suoi dati di training, ma agisce esclusivamente sul traffico che genera, modificandolo in modo da essere classificato erroneamente dal sistema di rilevamento già in esercizio. Il modello rimane invariato, ed è l'input che viene manipolato [6, 3].

Nel contesto del presente lavoro, l'attacco rilevante è l'evasion. L'ipotesi di partenza è che il ML-NIDS sia già addestrato e operativo, e che l'attaccante non abbia mai avuto, né abbia, accesso al training set o alla pipeline di addestramento. L'obiettivo è comprendere se, e in che misura, un avversario che opera esclusivamente sul traffico che genera sia in grado di eludere il rilevamento.

Conoscenza dell'attaccante: white-box, gray-box e black-box. Il secondo criterio di classificazione riguarda il grado di conoscenza che l'attaccante possiede riguardo al sistema che intende ingannare. Questa dimensione è trasversale alla precedente, nel senso che sia gli attacchi di poisoning sia quelli di evasion possono essere condotti con livelli di conoscenza molto diversi, e la quantità di informazioni disponibili all'attaccante influenza direttamente la difficoltà dell'attacco e la sua rilevanza pratica [3].

Nello scenario *white-box*, l'attaccante ha accesso completo al modello, di cui conosce l'architettura, i parametri interni, il feature set utilizzato e, in alcuni casi, anche i dati su cui è stato addestrato. Questa conoscenza gli permette di costruire attacchi ottimali, calcolati con precisione per massimizzare la probabilità di evasione. È lo scenario più studiato nella letteratura su adversarial machine learning, perché rende il problema matematicamente trattabile, in quanto conoscendo il modello è possibile calcolare analiticamente le perturbazioni più efficaci. Tuttavia, è anche lo scenario meno realistico in un contesto operativo, poiché un attaccante esterno alla rete monitorata non ha, in condizioni normali, accesso alle componenti interne del sistema di rilevamento [3, 8]. Valutare un ML-NIDS esclusivamente in uno scenario white-box rischia pertanto di fornire una misura di robustezza che sovrastima la difficoltà reale dell'evasione dal punto di vista dell'attaccante, o, detto altrimenti, che sottostima la protezione effettiva del sistema in condizioni operative reali.

Nello scenario *gray-box*, l'attaccante dispone di una conoscenza parziale del sistema. Un esempio tipico è il caso in cui l'attaccante conosca il tipo di feature utilizzate dal rilevatore, ad esempio che il sistema lavora su statistiche aggregate di flusso, senza però avere accesso al modello interno, ai suoi parametri o ai dati di training. Questa è una situazione più realistica, poiché un attaccante sufficientemente esperto potrebbe ragionevolmente inferire che un ML-NIDS basato su NetFlow utilizzi feature legate a durata, volume di traffico e distribuzione dei pacchetti, senza conoscere i dettagli del classificatore [3].

Nello scenario *black-box*, infine, l'attaccante non dispone di alcuna conoscenza interna del sistema, non conosce le feature, non conosce il modello, e può al più osservare gli output del sistema, qualora questi siano accessibili, per inferire informazioni sul suo comportamento. È lo scenario più realistico dal punto di vista operativo, ma anche il più vincolante per l'attaccante. Nonostante questo, la letteratura ha documentato attacchi black-box efficaci contro i ML-NIDS, sfruttando la trasferibilità degli adversarial example, per cui un attacco sviluppato contro un modello surrogato costruito dall'attaccante autonomamente può risultare efficace anche contro il modello bersaglio reale, pur senza che l'attaccante ne conosca i dettagli [3, 6].

La scelta dello scenario di conoscenza non è quindi solo una questione di classificazione formale, poiché determina quali attacchi sono realisticamente eseguibili e, di conseguenza, quali valutazioni di robustezza hanno effettiva rilevanza pratica per un sistema in produzione. Stabilita questa tassonomia, è possibile esaminare in che misura gli attacchi adversarial si siano dimostrati concretamente efficaci contro i ML-NIDS nella pratica, e quali implicazioni questo comporti per la loro affidabilità come strumenti di sicurezza.

2.2.3 Impatto degli adversarial attack sui ML-NIDS: evidenze empiriche

Le sezioni precedenti hanno mostrato cosa sono gli adversarial attack e secondo quali criteri è possibile classificarli. Resta però aperta la domanda più concreta: questi attacchi funzionano davvero? Il fatto che esistano vulnerabilità teoriche in un modello di machine learning non implica necessariamente che siano sfruttabili in pratica, potrebbe trattarsi di debolezze che richiedono condizioni così particolari da risultare irrilevanti in uno scenario reale.

La letteratura scientifica ha affrontato questa domanda in modo sistematico, e la risposta che emerge è inequivocabile, e dice che i ML-NIDS sono vulnerabili agli adversarial attack in modo concreto e riproducibile, indipendentemente dalle specifiche scelte implementative del sistema considerato [4, 14, 3]. Ciò che colpisce, leggendo i lavori che hanno studiato questo problema, non è tanto l'esistenza della vulnerabilità in sé, che potrebbe essere attesa dato che i modelli di machine learning sono per natura sensibili all'input, quanto la sua entità e la sua sistematicità. Il pattern che emerge trasversalmente alla letteratura è sempre lo stesso e cioè, che modelli che su traffico non perturbato raggiungono prestazioni di rilevamento elevate, subiscono un degrado significativo non appena il traffico malevolo viene modificato in modo mirato. E la quantità di modifica necessaria per ottenere questo effetto è spesso sorprendentemente contenuta. Non si tratta di stravolgere il traffico malevolo al punto da renderlo irriconoscibile o disfunzionale, poiché perturbazioni circoscritte, che lasciano intatta la funzionalità dell'attacco sottostante, sono sufficienti a far sì che il modello non lo riconosca più come tale [4, 3]. Questo fenomeno è stato

documentato in scenari molto diversi tra loro, con modelli di tipo diverso, su dataset differenti, e con livelli di conoscenza dell'attaccante che vanno dal white-box al black-box. La convergenza di questi risultati è un segnale importante poiché suggerisce che la vulnerabilità non sia un artefatto di scelte implementative specifiche come un modello mal configurato, un dataset poco rappresentativo, ma una caratteristica strutturale dei ML-NIDS come categoria. In altre parole, non si tratta di un problema che si risolve semplicemente scegliendo un algoritmo diverso o raccogliendo più dati di training [14, 3, 8].

Vale la pena soffermarsi su un aspetto che distingue questo problema da quello analogo in altri domini applicativi del machine learning. Quando un classificatore di immagini viene ingannato da un adversarial example, scambia un animale per uno di categoria completamente diversa a causa di una perturbazione impercettibile sui pixel, e le conseguenze sono limitate al singolo errore di classificazione. Nel contesto di un ML-NIDS, le conseguenze sono strutturalmente diverse poiché un falso negativo non è solo un errore statistico, ma significa che un attaccante sta operando all'interno della rete monitorata senza essere rilevato, per tutto il tempo che la sua attività rimane sotto la soglia di rilevamento del sistema. La posta in gioco è quindi asimmetrica rispetto ad altri domini, la differenza tra un modello robusto e uno vulnerabile non si misura solo in punti percentuali di accuracy, ma in termini di **esposizione** reale dell'infrastruttura a minacce attive [8]. Questa asimmetria ha una conseguenza diretta per come si valuta e si progetta un ML-NIDS. Un sistema che viene testato esclusivamente su traffico non perturbato, anche con risultati eccellenti, offre garanzie di robustezza incomplete, poiché le sue prestazioni in condizioni reali, dove un attaccante motivato ha ogni incentivo a cercare modi per eluderlo, potrebbero essere significativamente inferiori a quelle osservate in laboratorio. La valutazione della robustezza avversariale non è dunque un esercizio opzionale, riservato a chi vuole approfondire ulteriormente un sistema già validato, ma è una componente necessaria di qualsiasi processo di validazione serio di un ML-NIDS [8, 3].

Presa consapevolezza della vulnerabilità e della sua rilevanza operativa, la domanda che si apre naturalmente è come affrontarla. Quali strategie sono state proposte in letteratura per rendere i ML-NIDS più robusti agli adversarial attack,e quali sono i loro limiti? È a questa domanda che la sezione successiva risponde.

2.3 Difese verso gli adversarial attack

La sezione precedente ha illustrato come gli adversarial attack rappresentino una minaccia concreta e documentata per i ML-NIDS: perturbazioni anche minime del traffico di rete, applicate in modo mirato, sono sufficienti a far classificare campioni malevoli come benigni, compromettendo l'affidabilità del rilevatore nel suo contesto operativo. Chiarita la portata del problema, è opportuno spostare il punto di osservazione dal lato del difensore e chiedersi come si possa costruire un rilevatore che mantenga le proprie capacità di

detection anche in presenza di traffico adversarial. Le risposte proposte dalla letteratura non sono univoche. Esistono famiglie di approcci che intervengono in momenti diversi e con obiettivi parzialmente distinti, ciascuna con vantaggi e limitazioni proprie. Il presente lavoro si colloca nell'ambito dell'*adversarial training* (AT), una strategia difensiva su cui ci si sofferma in dettaglio nelle sezioni seguenti e che costituisce il nucleo metodologico del contributo proposto.

2.3.1 Panoramica delle strategie difensive

Le difese contro gli adversarial attack sui ML-NIDS possono essere classificate in base alla fase del processo di addestramento e deployment in cui intervengono. Una prima distinzione utile separa le difese che agiscono prima o indipendentemente dal training (modificando la rappresentazione dei dati o la struttura del modello), da quelle che invece intervengono durante il training stesso, alterando il processo di apprendimento per produrre un classificatore intrinsecamente più robusto.

Nella prima categoria rientrano due famiglie principali. La prima è quella del **preprocessing e feature engineering difensivo**: l'idea di fondo è che alcune feature estratte dai flussi di rete siano strutturalmente più facili da manipolare da parte di un attaccante rispetto ad altre, e che rimuoverle o trasformarle possa ridurre la superficie di attacco disponibile. Han et al. [14] propongono una difesa di questo tipo nel contesto dei ML-NIDS, identificando le feature più vulnerabili alle perturbazioni in *problem space*[1] e rimuovendole dalla rappresentazione utilizzata dal classificatore, ottenendo una riduzione significativa del tasso di evasione. Questo approccio ha il vantaggio di essere applicabile a modelli già addestrati senza richiederne il riaddestramento, ma presenta una limitazione strutturale: la selezione delle feature da rimuovere presuppone una conoscenza a priori delle strategie perturbative dell'attaccante, e un avversario che adotti tattiche diverse da quelle considerate in fase di progettazione potrebbe comunque trovare feature manipolabili tra quelle rimaste.

La seconda famiglia è quella delle **modifiche alla struttura del modello**. Un esempio rilevante nel contesto dei ML-NIDS è il lavoro di Apruzzese et al. [1], che adatta la tecnica della *defensive distillation* a classificatori basati su Random Forest. In sintesi, l'approccio prevede che il modello venga addestrato non su etichette binarie rigide, ovvero ogni campione è malevolo o benigno, senza sfumature, ma su rappresentazioni più sfumate della confidenza di classificazione, prodotte da un primo classificatore ausiliario. L'obiettivo è rendere la superficie decisionale del modello meno sensibile a piccole variazioni dell'input, che è precisamente ciò che le perturbazioni adversariali sfruttano. Si tratta di una difesa che opera interamente in **feature space**: la robustezza ottenuta è definita rispetto a

[1] Han et al. utilizzano nel loro lavoro il termine *traffic-space*; ai fini della presente trattazione lo si considera equivalente a *problem space*.

perturbazioni del vettore di feature estratto, non rispetto a manipolazioni del traffico reale a livello di pacchetto.

La terza famiglia, su cui si concentra il presente lavoro, è quella dell'**adversarial training**. A differenza degli approcci precedenti, l'AT non modifica la struttura del modello né seleziona o rimuove feature: interviene sul processo di apprendimento, ampliando il training set con esempi adversariali affinché il modello impari direttamente a classificare correttamente anche gli input perturbati. Questa strategia è descritta in dettaglio nella sezione successiva. Vale la pena osservare che la maggior parte degli approcci difensivi presenti in letteratura, incluse le due famiglie appena descritte, opera in **feature space**: le perturbazioni contro cui si difendono sono perturbazioni del vettore numerico estratto dal traffico, non manipolazioni del traffico reale. Come discusso nella sezione precedente, questa scelta implica che le garanzie di robustezza offerte da tali difese siano valide rispetto a un modello di attaccante che può intervenire sulle feature interne del rilevatore, condizione che in uno scenario operativo reale è raramente verificabile.

2.3.2 L'adversarial training: fondamenti

Tra le strategie difensive discusse nella sezione precedente, l'adversarial training occupa una posizione particolare. Non interviene sulla rappresentazione dei dati né sulla struttura del classificatore, ma agisce sul processo di apprendimento stesso. L'intuizione di fondo è che un modello addestrato su traffico reale, incluso traffico malevolo reale, impara a separare le due classi nelle loro forme naturali, ma non è necessariamente in grado di riconoscere traffico malevolo che è stato deliberatamente modificato per sembrare benigno agli occhi del classificatore. Durante il training, il modello non ha mai incontrato campioni costruiti appositamente per ingannarlo, ha visto attacchi reali, non attacchi ottimizzati per eluderlo. Quando questi ultimi arrivano in fase operativa, il modello li affronta come situazioni per cui non è stato preparato.

L'adversarial training risponde a questo problema in modo pragmatico, esponendo il modello, già durante l'addestramento, a campioni costruiti per ingannarlo, affinché impari a riconoscerli e classificarli correttamente insieme al traffico ordinario. In termini operativi, questo si traduce in una forma di *data augmentation* strutturata. Il training set originale viene ampliato con esempi adversariali, ovvero campioni malevoli opportunamente perturbati, e il modello viene addestrato su questo insieme esteso. L'obiettivo non è semplicemente memorizzare gli esempi adversariali visti durante il training, ma indurre nel classificatore una maggiore robustezza strutturale, tale da generalizzare anche su perturbazioni non incontrate esplicitamente. In questo senso l'adversarial training non è un rimedio puntuale contro un attacco specifico, ma una strategia di irrobustimento generale del processo di apprendimento.

Rispetto alle altre famiglie di difese, l'AT presenta alcuni vantaggi concettuali rilevanti. Non dipende da una lista chiusa di attacchi noti: qualunque tipologia di perturbazione utilizzata per generare gli esempi di training contribuisce, in misura variabile, ad aumentare la robustezza del modello. Non richiede modifiche all'architettura del classificatore, il che lo rende applicabile a modelli già progettati e validati senza interventi strutturali. Agisce inoltre sulla capacità discriminativa del modello, non sulla sua superficie di input, poiché un modello addestrato avversarialmente impara a costruire rappresentazioni interne più robuste, non semplicemente a filtrare certi tipi di input a monte.

Accanto a questi vantaggi esistono però alcune limitazioni. Una limitazione ben documentata è il potenziale *trade-off* tra robustezza e performance su traffico pulito. Un modello esposto a esempi adversariali durante il training potrebbe, in linea di principio, perdere parte della propria accuratezza sul traffico non perturbato, poiché il processo di apprendimento deve ora soddisfare vincoli aggiuntivi rispetto al caso standard. Questa tensione è osservata in letteratura in diversi contesti, ed è uno degli aspetti che il presente lavoro verifica sperimentalmente nel dominio dei ML-NIDS. Accertare che i modelli sottoposti ad adversarial training non subiscano una regressione significativa sulle performance su traffico pulito è parte integrante della valutazione proposta, e non una verifica accessoria.

2.3.3 Lavori correlati e posizionamento del contributo

L'adversarial training è una strategia difensiva che conta numerosi esempi in letteratura, ma la sua applicazione nel dominio dei ML-NIDS, e in particolare in *problem space*, rimane un territorio ancora poco esplorato rispetto a quanto fatto in *feature space* o in altri domini applicativi della sicurezza informatica. Per comprendere dove si collochi il presente lavoro rispetto a quanto già proposto, è utile esaminare nel dettaglio i contributi più rilevanti, evidenziandone le scelte metodologiche e le differenze rispetto all'approccio qui adottato.

Adversarial retraining in feature space per ML-NIDS. Apruzzese et al. [6] affrontano esplicitamente il problema delle difese contro gli adversarial attack nei sistemi di sicurezza basati su machine learning, con esperimenti condotti su traffico di rete reale del dataset CTU-13 e su più classificatori supervisionati. La strategia difensiva proposta è un *adversarial retraining* in **feature space**, nel quale gli esempi adversariali vengono costruiti perturbando direttamente i valori numerici del vettore di feature già estratto dal traffico, senza alcun intervento sul traffico originale. Il retraining con questi campioni riduce la severità degli attacchi evasivi, ma gli stessi autori osservano che la protezione ottenuta non si estende a tipologie di perturbazione diverse da quelle impiegate durante il retraining, evidenziando un limite intrinseco di generalizzazione. Rispetto al presen-

te lavoro, la differenza fondamentale risiede nello spazio operativo. Le perturbazioni in *feature space* presuppongono che l'attaccante sia in grado di intervenire sulla rappresentazione interna del classificatore, condizione che in scenari operativi reali è difficilmente verificabile. Il presente lavoro costruisce invece gli esempi adversariali in *problem space*, a partire da manipolazioni del traffico reale prima dell'estrazione delle feature.

Defensive distillation per Random Forest su CTU-13. Apruzzese et al. [1] propongono una strategia di irrobustimento basata sull'adattamento della *defensive distillation* a classificatori Random Forest, operante interamente in **feature space**. Il lavoro è condotto su un setup sperimentale molto vicino a quello del presente contributo, poiché utilizza lo stesso classificatore, lo stesso dataset e un set di feature comparabile. L'idea alla base della distillazione è che il modello venga addestrato non sulle etichette binarie originali, ma su una rappresentazione più sfumata della confidenza di classificazione, prodotta da un classificatore ausiliario, al fine di rendere la superficie decisionale meno reattiva a piccole variazioni dell'input. I risultati mostrano che il modello distillato mantiene performance elevate su traffico non perturbato e ottiene una robustezza in scenari adversariali superiore sia alla baseline non difesa sia all'adversarial retraining tradizionale. La differenza rispetto al presente lavoro è duplice. Sul piano della strategia difensiva, la distillazione interviene sul modo in cui il modello apprende le etichette, mentre l'adversarial training interviene sul contenuto del training set, aggiungendo esempi perturbati. Sul piano dello spazio operativo, la distillazione è interamente in *feature space*, e la robustezza che produce è definita rispetto a perturbazioni del vettore numerico, non a manipolazioni del traffico reale.

Adversarial training con Deep Reinforcement Learning per botnet detection. Apruzzese et al. [2] propongono un framework basato su Deep Reinforcement Learning (DRL) per la generazione automatica di campioni adversariali e il successivo irrobustimento di rilevatori di botnet. Il DRL è una tecnica di apprendimento automatico in cui un agente software apprende, attraverso un processo iterativo di tentativi e ricompense, quale sequenza di modifiche applicare a un campione malevolo affinché il classificatore bersaglio lo classifichi erroneamente come benigno. Il lavoro opera su traffico NetFlow estratto dai dataset CTU-13 e BOTNET, con classificatori Random Forest e un modello basato su deep learning. Gli agenti DRL apprendono a generare campioni evasivi attraverso modifiche incrementali e controllate di un sottoinsieme di feature del flusso di rete, guidate dal feedback del classificatore target. I rilevatori vengono poi irrobustiti iniettando nel training set i campioni così generati. I risultati mostrano che i modelli hardened non subiscono degradazione delle performance su traffico non perturbato e migliorano significativamente la capacità di rilevamento in scenari avversariali, sia rispetto alla baseline sia rispetto a strategie di retraining basate su campioni costruiti manualmente. Si tratta del

contributo più vicino al presente lavoro per dominio applicativo, classificatore e dataset. La differenza sostanziale è che l'intero processo, sia la generazione degli esempi adversariali sia la valutazione della difesa, avviene in **feature space**. Le perturbazioni sono guidate dall'output del classificatore, il che implica un modello di attaccante con accesso almeno parziale al sistema di rilevamento. Nel presente lavoro, le perturbazioni sono invece applicate in *problem space* ai pacchetti di rete prima della conversione in flussi, e sono stocastiche, ovvero non guidate da alcun feedback del classificatore.

Feature removal come difesa per ML-NIDS. Han et al. [14] propongono un framework che combina attacco e difesa per la valutazione della robustezza di ML-NIDS. Sul lato offensivo, il loro attacco opera in **problem space** (indicato nel loro lavoro come *traffic-space*), modificando il traffico di rete reale per alterare le caratteristiche dei flussi malevoli. Sul lato difensivo, la contromisura proposta consiste nell'identificazione e nella rimozione delle feature più vulnerabili alle perturbazioni, con una riduzione significativa del tasso di evasione. L'approccio presenta dunque una struttura ibrida, in cui l'attacco è realistico e opera sul traffico, ma la difesa interviene sulla rappresentazione interna dei dati, non sul processo di apprendimento. Rispetto al presente lavoro, la differenza è sia nella tipologia di difesa adottata, feature removal contro adversarial training, sia nel fatto che la rimozione di feature presuppone la conoscenza a priori delle strategie perturbative dell'attaccante, requisito che ne limita la generalizzabilità a scenari in cui l'avversario adotti tattiche impreviste.

Perturbazioni in problem space su traffico di rete senza adversarial training. Apruzzese, Fass e Pierazzi [5] si avvicinano maggiormente all'approccio del presente lavoro sul piano delle perturbazioni. Il loro studio analizza l'effetto di perturbazioni in **problem space** applicate a traffico reale di rete, operando allo stesso livello, quello del pacchetto, su cui si basa la strategia adottata in questa tesi. Il contributo è tuttavia focalizzato sull'analisi dell'interazione tra perturbazioni adversariali e *concept drift*. Con questo termine si indica il fenomeno per cui la distribuzione statistica del traffico di rete cambia nel tempo, ad esempio per l'introduzione di nuovi servizi, la variazione dei comportamenti degli utenti o l'evoluzione delle minacce, con la conseguenza che un modello addestrato su dati di un certo periodo può perdere progressivamente efficacia su dati raccolti in un periodo successivo. Lo studio in questione analizza come questo fenomeno interagisca con le manipolazioni introdotte dall'attaccante, e non propone né implementa un meccanismo di adversarial training. Il presente lavoro si differenzia dunque non tanto nella tecnica perturbativa, che è concettualmente affine, quanto nell'uso che viene fatto dei campioni perturbati, che qui sono impiegati per ampliare il training set e costruire modelli hardened, un passaggio che quel lavoro non compie.

Adversarial training in problem space per classificatori di malware binario. Al di fuori del dominio dei ML-NIDS, il lavoro di Lucas et al. [15] rappresenta uno dei pochi esempi di adversarial training condotto interamente in **problem space**. Gli autori operano su classificatori di malware addestrati direttamente sui byte grezzi di file eseguibili, e le perturbazioni consistono in trasformazioni che modificano la struttura del binario preservandone la funzionalità. Un risultato particolarmente rilevante per il confronto con il presente lavoro riguarda l'inefficacia della semplice *data augmentation* casuale come strategia difensiva, a fronte dell'efficacia dell'adversarial training condotto con versioni anche semplificate degli stessi attacchi reali, con una riduzione del tasso di successo degli attacchi dal 90% al 5% nel caso migliore. Questa evidenza è coerente con la scelta adottata nel presente lavoro, dove i campioni adversariali di training sono costruiti a partire da perturbazioni realistiche del traffico e non da rumore privo di struttura. La differenza rispetto al presente contributo è nel dominio applicativo, poiché Lucas et al. operano su malware binario e non su traffico di rete, e nel tipo di dati e classificatori impiegati.

Il quadro che emerge da questa rassegna è sintetizzato nella Tabella 2.1, che mette a confronto i lavori discussi lungo le dimensioni che definiscono il posizionamento del presente contributo. La lettura per colonne rende visibile un pattern ricorrente. Nella colonna dello spazio delle perturbazioni, i lavori che propongono una qualche forma di adversarial training nel dominio dei ML-NIDS [6, 2] operano tutti in *feature space*, costruendo gli esempi adversariali a partire dal vettore numerico già estratto dal traffico. I lavori che operano in *problem space* su traffico reale di rete [14, 5] non adottano l'adversarial training come strategia difensiva, proponendo alternative come la rimozione delle feature vulnerabili o limitandosi ad analizzare l'effetto delle perturbazioni senza intervenire sul processo di addestramento. L'unico esempio di adversarial training in *problem space* [15] opera su un dominio diverso, quello dei classificatori di malware binario, con dati, perturbazioni e modelli non direttamente trasferibili al contesto dei ML-NIDS. La colonna relativa alla generazione degli esempi adversariali evidenzia una seconda distinzione rilevante. Tutti i lavori che impiegano l'adversarial training generano i propri campioni attraverso processi guidati dal feedback del classificatore bersaglio, il che presuppone un modello di attaccante con accesso almeno parziale al sistema di rilevamento.

Il presente lavoro adotta invece perturbazioni stocastiche, che non richiedono alcuna conoscenza del classificatore e risultano quindi coerenti con lo scenario black-box formalizzato nel threat model. Infine, la colonna relativa alla verifica su traffico non perturbato mostra che non tutti i lavori valutano esplicitamente se la difesa proposta introduca una regressione delle prestazioni in assenza di attacchi, aspetto che il presente lavoro considera parte integrante della valutazione.

Il presente contributo si colloca dunque nello spazio lasciato aperto da questa rassegna, combinando l'adversarial training come strategia difensiva con la costruzione degli esempi adversariali in *problem space* attraverso manipolazioni stocastiche del traffico a

livello di pacchetto, e verificando che questa strategia non comprometta le performance del rilevatore su traffico non perturbato.

Tabella 2.1: Confronto tra letteratura e corrente lavoro

Lavoro	**Dominio**	**Spazio perturbazioni**	**Strategia difensiva**	**Generazione esempi adv.**	**Verifica su traffico clean**
Addressing Adv. Attacks Against Security Systems [6]	NIDS	Feature sp.	AT	Guidata	No
Hardening Random Forest Cyber Detectors [1]	NIDS	Feature sp.	Distillation	N.A.	Sì
Deep Reinforcement Adv. Learning Against Botnet [2]	NIDS	Feature sp.	AT	Guidata	Sì
Evaluating and Improving Adv. Robustness of ML-NIDS [14]	NIDS	Ibrido	Feature rem.	N.A.	No
When Adv. Perturbations Meet Concept Drift [5]	NIDS	Problem sp.	Nessuna	N.A.	No
Adversarial Training for Raw-Binary Malware [15]	Malware	Problem sp.	AT	Guidata	No
Presente lavoro	NIDS	Problem sp.	AT	Stocastica	Sì

Capitolo 3

Contributo e approccio

Il Capitolo 2 ha costruito il quadro di riferimento teorico necessario a comprendere le scelte adottate. Ha descritto l'architettura e le vulnerabilità dei ML-NIDS, ha inquadrato gli adversarial ML attack come minaccia concreta a questi sistemi e ha esaminato le principali strategie difensive presenti in letteratura, con particolare attenzione all'adversarial training e alla distinzione tra approcci in *feature space* e approcci in *problem space*.

Ha inoltre evidenziato come le strategie difensive proposte in letteratura operino prevalentemente in *feature space*, spazio in cui la generazione di esempi adversariali è analiticamente più trattabile, ma che non riflette le operazioni concrete di un attaccante reale, il quale agisce a livello di traffico di rete e non ha accesso diretto alla rappresentazione interna del rilevatore. È proprio in questa direzione, non esplorata dalla letteratura esaminata, che si colloca il presente lavoro: adotta l'adversarial training come strategia difensiva e costruisce gli esempi adversariali in *traffic space*, ossia a partire da perturbazioni stocastiche applicate direttamente ai pacchetti di rete malevoli.[1]

A partire da questo quadro, il presente capitolo introduce il contributo metodologico originale della tesi: la progettazione e la validazione di un framework di adversarial training operante interamente in *traffic space*, in cui gli esempi adversariali sono costruiti a partire da perturbazioni stocastiche applicate al traffico reale a livello di pacchetto e utilizzati per addestrare modelli ML-NIDS più robusti. Vengono inoltre descritte, ad alto livello, le fasi del framework proposto e viene formalizzato lo scenario operativo in cui esso si colloca. I dettagli tecnici della pipeline sono trattati nel capitolo successivo, dedicato all'implementazione. I risultati sperimentali e la valutazione delle prestazioni trovano invece spazio nel Capitolo 5.

Il capitolo è organizzato in due sezioni. La Sezione 3.1 formalizza il threat model adottato, specificando le conoscenze e le capacità attribuite all'attaccante, l'obiettivo del difensore e le caratteristiche generali dello scenario sperimentale. La Sezione 3.2 presenta

[1] Con il termine *traffic space* si intende qui l'istanziazione del più generale concetto di *problem space* [20] al dominio specifico del traffico di rete, in cui le perturbazioni operano a livello di pacchetto e sono soggette ai vincoli protocollari del dominio.

il framework di lavoro e le fasi che lo compongono, illustrando la logica complessiva dell'approccio, dalla generazione degli esempi adversariali all'addestramento dei modelli hardened, senza anticipare i dettagli tecnici.

3.1 Threat Model

Prima di descrivere le scelte metodologiche adottate nel presente lavoro, è necessario formalizzare il *threat model*, ovvero il modello di minaccia che definisce lo scenario entro cui il problema viene studiato.

Un threat model specifica chi sono gli attori coinvolti — attaccante e difensore — quali conoscenze e capacità ciascuno possiede, e quale obiettivo ciascuno persegue. Senza questa formalizzazione, le affermazioni sulla robustezza o sulla vulnerabilità di un sistema risultano prive di un riferimento preciso.

La questione non è puramente formale. La letteratura ha documentato come molti lavori sugli adversarial attack contro i ML-NIDS adottino threat model che, pur essendo analiticamente comodi, presuppongono capacità dell'attaccante difficilmente verificabili in scenari operativi reali [3, 8]. Ad esempio, un attaccante white-box, che conosce il modello nel dettaglio e può calcolare perturbazioni ottimali, è un caso limite utile per stabilire un lower bound sulla robustezza del sistema, ma non rappresenta la minaccia più probabile che un ML-NIDS in produzione si trova ad affrontare.

Per questo motivo, il presente lavoro adotta deliberatamente un threat model realistico e conservativo, in cui le capacità dell'attaccante sono vincolate a ciò che è concretamente realizzabile in un contesto operativo. Come si argomenta nelle sezioni seguenti, questo non indebolisce la rilevanza dei risultati: al contrario, dimostrare che anche un attaccante fortemente vincolato riesce a degradare significativamente le prestazioni di un ML-NIDS rafforza la tesi sulla vulnerabilità strutturale di questi sistemi.

3.1.1 Sistema target

Il sistema oggetto di analisi è un ML-NIDS dedicato al monitoraggio del traffico di una rete aziendale di medie dimensioni [3]; esso non è collocato direttamente sul perimetro della rete, bensì è collegato al border router tramite SPAN port[2], in modo da ricevere una copia del traffico e analizzarlo, senza intaccare il flusso delle comunicazioni.

Il compito del ML-NIDS è classificare il traffico di rete osservato come benigno o malevolo, generando un alert ogni volta che viene rilevata un'attività sospetta. Il traffico di rete è nella sua forma elementare composto da pacchetti, ovvero unità di dati scambiate tra host contenenti informazioni sul mittente, sul destinatario e sul contenuto della

[2]La SPAN port (Switched Port Analyzer) è una funzionalità degli switch/router che consente di inviare una copia del traffico in transito verso una porta dedicata, senza interferire con il flusso originale.

comunicazione. Anziché operare su questi pacchetti nella loro forma grezza, il sistema lavora su rappresentazioni aggregate del traffico in formato *NetFlow*: ogni flusso viene descritto da un insieme di feature statistiche, tra cui durata, numero di pacchetti, byte trasmessi in ciascuna direzione, porte, flag TCP e altre caratteristiche comportamentali della comunicazione, che costituiscono il vettore di input su cui il classificatore produce la propria predizione [3]. Questa modalità operativa è la più diffusa nella letteratura sui ML-NIDS, in quanto bilancia efficacia discriminativa, scalabilità computazionale e compatibilità con l'infrastruttura di rete esistente e consente inoltre di operare indipendentemente dalla cifratura del traffico, poiché le feature estratte dai flussi non richiedono l'ispezione del payload [8].

Il classificatore è stato addestrato su un dataset di traffico etichettato, contenente esempi di traffico benigno e di traffico malevolo appartenente a una specifica famiglia di botnet, e ha dimostrato, prima del deployment, prestazioni elevate su traffico non perturbato. Gli amministratori del sistema considerano pertanto l'ML-NIDS affidabile per identificare le minacce note, e lo impiegano come componente centrale della propria strategia di rilevamento. Ciò che gli amministratori non sanno, e non possono sapere senza una valutazione adversariale esplicita, è se il sistema manterrà le proprie capacità di rilevamento nel momento in cui un attaccante modifichi intenzionalmente il proprio traffico per sfuggire alla classificazione. È proprio questa incertezza che rende necessario formalizzare le caratteristiche dell'avversario che si intende modellare: conoscenze, capacità operative e obiettivo perseguito. La sezione seguente è dedicata a questa formalizzazione.

3.1.2 Attaccante

Posizione. Lo scenario considerato presuppone che l'attaccante abbia già ottenuto il controllo di uno o più host all'interno della rete monitorata, ad esempio tramite phishing, sfruttamento di vulnerabilità note o social engineering. Gli host compromessi sono sotto il pieno controllo dell'attaccante, che può eseguire codice arbitrario e manipolare il traffico di rete generato da tali macchine.

Obiettivo. L'obiettivo dell'attaccante non è l'intrusione in sé, già avvenuta, ma il mantenimento della propria presenza all'interno della rete senza essere rilevato, in modo da poter perseguire obiettivi di livello superiore quali l'esfiltrazione di dati, il movimento laterale verso altri sistemi o il mantenimento della persistenza nel lungo periodo.

Questo obiettivo si traduce nel mantenere il proprio traffico al di sotto della soglia di attenzione del sistema difensivo, qualunque esso sia, alterandone le caratteristiche in modo da renderlo indistinguibile dal traffico legittimo agli occhi di chi monitora la rete.

Conoscenza. L'attaccante opera in uno scenario *black-box* rispetto all'ML-NIDS: il sistema di rilevamento è, dal suo punto di vista, invisibile [5]. L'unica assunzione che gli si attribuisce è di carattere operativo generale, ossia sa di operare in una rete monitorata e ritiene plausibile che il proprio traffico possa essere soggetto ad analisi, ma non dispone di alcuna informazione sul sistema che lo analizza: non ne conosce la natura (non sa se sia basato su regole, su machine learning o su altro), non conosce le feature estratte dal traffico, il dataset di addestramento, l'algoritmo adottato né i suoi parametri interni [5]. Non è inoltre in grado di sottoporre query al sistema per osservarne le risposte e inferirne il comportamento, come avverrebbe in uno scenario *query-based* [3].

Questa caratterizzazione corrisponde a un attaccante realistico e, rispetto agli scenari più comuni nella letteratura sugli adversarial attack contro i ML-NIDS, deliberatamente debole [8]. La grande maggioranza dei lavori esistenti assume scenari white-box o gray-box, in cui l'attaccante dispone di conoscenza parziale o totale del modello bersaglio, o può almeno interrogarlo per inferirne il comportamento [6, 3]. Tali scenari permettono di costruire perturbazioni ottimizzate e rappresentano un caso limite utile per stabilire un lower bound sulla robustezza del sistema, ma difficilmente corrispondono alle condizioni in cui un avversario reale si trova a operare: le componenti interne di un ML-NIDS in produzione sono tipicamente ben protette e non accessibili dall'esterno [8]. La scelta di un attaccante black-box è dunque motivata non da una semplificazione analitica, ma dalla volontà di valutare la robustezza del sistema in condizioni operative plausibili [5].

Capacità. L'attaccante può modificare il traffico di rete generato dagli host sotto il proprio controllo, e unicamente quello. In pratica, può intervenire sui pacchetti prima che lascino la macchina compromessa, purché il risultato sia un pacchetto valido dal punto di vista protocollare. Non può modificare il traffico generato da altri host, non può intercettare o alterare comunicazioni che non transitino attraverso le macchine compromesse, e non può in alcun modo accedere al ML-NIDS o interferire con il suo funzionamento.

Questa distinzione richiama direttamente la differenza tra attacchi in *traffic space* e attacchi in *feature space* introdotta nel Capitolo 2. Un attacco in feature space richiederebbe che l'attaccante possa modificare direttamente il vettore di feature estratto dal traffico, operazione che presuppone accesso alle componenti interne del sistema di rilevamento e che non è realizzabile da un host compromesso. Le manipolazioni disponibili all'attaccante nel presente scenario si collocano invece interamente in traffic space: agiscono sul traffico reale a livello di pacchetto, rispettano i vincoli protocollari e preservano la funzionalità malevola delle comunicazioni [3, 5].

Strategia. Date la conoscenza nulla del sistema bersaglio e la capacità di intervenire esclusivamente sul traffico degli host compromessi, la strategia disponibile all'attaccante è l'introduzione di perturbazioni *blind* al traffico malevolo, ovvero modifiche applicate

senza possibilità di osservare o anticipare l'effetto che avranno sulla classificazione [5]. L'attaccante agisce a livello di pacchetto, alterando le caratteristiche del traffico generato dagli host sotto il proprio controllo in modo da modificare le statistiche di flusso su cui il rilevatore opera, pur mantenendo la validità protocollare dei pacchetti e la funzionalità delle comunicazioni malevole. Questo aspetto è centrale per la lettura dei risultati sperimentali: se anche perturbazioni non ottimizzate, applicate senza alcuna conoscenza del modello bersaglio, sono sufficienti a degradare significativamente le prestazioni di un ML-NIDS, la vulnerabilità osservata non è un artefatto di condizioni favorevoli all'attaccante, ma una caratteristica strutturale di questi sistemi. I dettagli tecnici delle perturbazioni adottate sono descritti nel Capitolo 4.

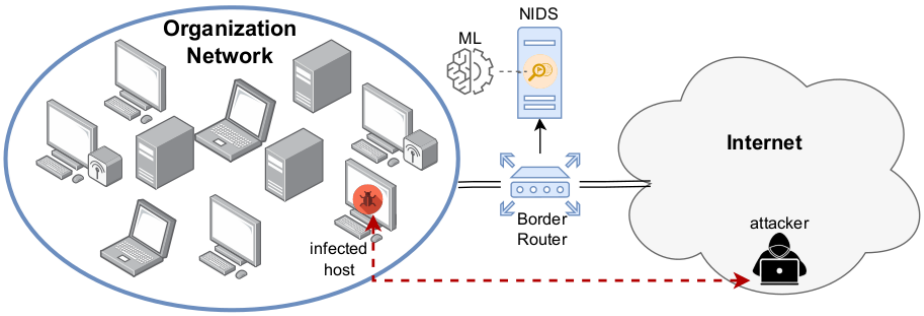

Figura 3.1: L'attaccante, esterno alla rete organizzativa, controlla uno o più host interni tramite un canale C&C. Le perturbazioni al traffico malevolo vengono applicate sull'host compromesso (*traffic space*). L'attaccante non ha accesso al ML-NIDS e non ne conosce le componenti interne (scenario *black-box*). Figura tratta da [5].

3.1.3 Difensore

Posizione. Il difensore è il team tecnico interno all'organizzazione, responsabile del deployment e della manutenzione dell'ML-NIDS [3, 5]. Opera dall'interno del sistema che gestisce, con piena visibilità sulle sue componenti e con la capacità di intervenire su di esse. Questa collocazione privilegiata rispetto al sistema è uno degli elementi che rendono il threat model realistico: nella pratica operativa, il difensore conosce il proprio sistema nel dettaglio, mentre l'attaccante vi si confronta dall'esterno e senza visibilità diretta.

Obiettivo. Il difensore persegue due obiettivi simultanei, che nel presente lavoro vengono verificati sperimentalmente in modo distinto.

Il primo obiettivo è la **robustezza**: il modello deve mantenere una *recall* elevata anche quando il traffico malevolo viene perturbato dall'attaccante. Con recall si intende la proporzione di campioni malevoli correttamente classificati come tali sul totale dei campioni

malevoli presenti; la definizione formale e la motivazione della scelta di questa metrica come indicatore primario sono discusse nel Capitolo 5.

Il secondo obiettivo è la **non-regressione**: il rafforzamento del modello contro le perturbazioni non deve compromettere le prestazioni su traffico non perturbato. Un modello che acquisisce robustezza avversariale al prezzo di una perdita significativa di recall sul traffico clean non è utile in produzione, poiché il traffico non perturbato costituisce la componente largamente dominante di ciò che il sistema osserva nel suo normale ciclo operativo. La verifica di questo secondo obiettivo è parte integrante della valutazione proposta, e non una considerazione accessoria: i risultati presentati nel Capitolo 5 includono esplicitamente la valutazione dei modelli hardened su traffico clean, a fianco della valutazione su traffico adversarial.

Questi due obiettivi non sono indipendenti: qualsiasi strategia difensiva adottata deve soddisfarli entrambi per essere considerata efficace in un contesto operativo reale. È proprio a partire da questo vincolo che si motiva la scelta dell'adversarial training in traffic space come approccio difensivo, su cui ci si sofferma nelle sezioni successive.

Conoscenza. Il difensore conosce tutte le componenti del sistema che amministra: l'algoritmo di classificazione adottato, le feature estratte dal traffico, il dataset di addestramento, gli iperparametri del modello e le metriche di performance osservate prima del deployment [3, 5]. Non dispone però di visibilità sulle azioni dell'attaccante: sa che perturbazioni realistiche del traffico malevolo sono possibili e conosce la categoria generale delle manipolazioni plausibili in traffic space, ma non può osservare in anticipo la forma concreta che tali perturbazioni assumeranno né con quale intensità verranno applicate.

Capacità. Il difensore può intervenire attivamente su tutte le componenti del sistema che amministra, dalla fase di costruzione del dataset fino al deployment del modello. Non può invece agire sul traffico in transito nella rete né interferire con il comportamento dell'attaccante o degli host compromessi.

Strategia. Di fronte a un attaccante che opera in modo cieco e che non può essere osservato in anticipo, la risposta del difensore non può basarsi sulla conoscenza delle perturbazioni specifiche che verranno adottate. La strategia adottata nel presente lavoro è l'adversarial training in *traffic space*: il modello viene addestrato su un insieme di dati arricchito con esempi adversariali costruiti a partire da perturbazioni realistiche del traffico malevolo, in modo da esporlo durante il training a manipolazioni dello stesso tipo di quelle che potrebbe incontrare in produzione. L'approccio è descritto in dettaglio nella Sezione 3.2.

3.2 Framework di lavoro e fasi

3.2.1 Panoramica e articolazione del lavoro

La sezione precedente ha formalizzato il modello di minaccia adottato: un attaccante black-box che opera in *traffic space*, modificando il traffico generato dagli host compromessi attraverso perturbazioni cieche e valide dal punto di vista protocollare, con l'obiettivo di eludere il rilevatore senza disporre di alcuna conoscenza interna del sistema. Di fronte a questo scenario, il difensore si trova nella posizione di dover rafforzare il proprio ML-NIDS contro una categoria di perturbazioni che non può osservare in anticipo nella loro forma concreta, ma di cui può anticipare la natura generale: manipolazioni realistiche del traffico di rete, applicate a livello di pacchetto, che alterano le caratteristiche dei flussi su cui il classificatore opera.

È a partire da questa asimmetria informativa tra attaccante e difensore che prende forma la domanda centrale del presente lavoro: è possibile addestrare un ML-NIDS in modo che sia robusto nei confronti di perturbazioni in *traffic space*, non osservate nella medesima forma durante la fase di addestramento, senza che tale rafforzamento comprometta le sue prestazioni sul traffico non perturbato?

La risposta che si propone è l'applicazione dell'adversarial training in *traffic space*, già introdotto come strategia difensiva nel Capitolo 2, con la differenza sostanziale che gli esempi adversariali vengono costruiti direttamente a livello di pacchetto, nel medesimo spazio in cui opera l'attaccante [5].

A tal fine, il lavoro si articola attorno a due momenti principali. Il primo è la costruzione di un modello di rilevamento addestrato su traffico non perturbato, che costituisce la *baseline* rispetto alla quale tutte le valutazioni successive vengono condotte. Questo modello rappresenta il sistema del difensore nella sua configurazione ordinaria, prima di qualsiasi intervento di rafforzamento.

Il secondo momento è l'adversarial training vero e proprio, che costituisce il contributo centrale del presente lavoro. A partire dalla stessa infrastruttura di dati e dallo stesso classificatore della baseline, si costruisce un modello *hardened*, addestrato su un insieme di dati ampliato con esempi adversariali generati in *traffic space*. La generazione di questi esempi segue una logica che rispecchia fedelmente le operazioni di un attaccante reale: si prende il traffico malevolo nella sua forma grezza a livello di pacchetto, lo si sottopone a perturbazioni stocastiche che rispettano i vincoli protocollari del dominio di rete, e si converte il traffico così modificato in flussi, ovvero nella stessa rappresentazione su cui il classificatore opera. Il risultato sono esempi adversariali che non sono costrutti artificiali, ma rappresentazioni fedeli di come traffico intenzionalmente alterato si presenterebbe a un rilevatore reale. È su questi esempi, affiancati al traffico ordinario, che il modello hardened viene addestrato.

Le sezioni seguenti descrivono le fasi del framework, dalla costruzione dei dati di lavoro alla generazione degli esempi adversariali, fino all'addestramento e alla valutazione dei modelli hardened.

3.2.2 Dati e modelli baseline

Prima di descrivere la costruzione della baseline, è opportuno chiarire la forma in cui i dati vengono utilizzati nel framework. Poiché il presente lavoro adotta un approccio in *traffic space*, le perturbazioni vengono applicate direttamente a livello di pacchetto, nello stesso spazio in cui opera l'attaccante: ne consegue naturalmente che il traffico malevolo debba essere disponibile nella sua forma grezza, in modo che su di esso possano essere operate le modifiche descritte nella sezione successiva. Il traffico benigno, non essendo oggetto di perturbazione, può essere utilizzato direttamente nella rappresentazione a flussi. Come si vedrà nel Capitolo 4, queste considerazioni si traducono in scelte operative precise; in questa sede interessa la loro giustificazione metodologica, che discende direttamente dalla scelta di operare in *traffic space*.

Organizzazione dei dati di lavoro. Il framework opera su due categorie di traffico distinte: traffico benigno e traffico malevolo. Il traffico malevolo, viene suddiviso in tre insiemi disgiunti, destinati rispettivamente all'addestramento del modello, alla validazione e alla verifica finale delle prestazioni. Il traffico benigno viene invece mantenuto come pool indifferenziato, da cui attingere nelle diverse fasi del framework. Questa organizzazione è il presupposto strutturale su cui si fondano tutte le fasi descritte nelle sezioni seguenti: il training set alimenta l'addestramento dei modelli, il validation set viene utilizzato per la generazione degli esempi adversariali nella fase di adversarial training, e il test set viene riservato esclusivamente alla valutazione finale, senza mai essere visto dai modelli durante l'addestramento.

Preprocessing e rappresentazione delle feature. Il traffico malevolo grezzo, una volta organizzato nei tre insiemi descritti, viene convertito in una rappresentazione a flussi, nella stessa forma in cui il traffico benigno è già disponibile. L'insieme risultante, che comprende sia i flussi malevoli sia quelli benigni, viene quindi sottoposto a un'operazione di preprocessing che li trasforma in una rappresentazione numerica uniforme su cui il classificatore può operare. Questa operazione è necessaria perché i flussi grezzi contengono informazioni in formati eterogenei che non sono tutti direttamente utilizzabili da un classificatore nella forma in cui si presentano: alcune richiedono trasformazioni specifiche affinché il modello possa estrarne informazione utile senza introdurre artefatti nella rappresentazione.

Nel presente lavoro vengono considerate due rappresentazioni di feature distinte, denominate rispettivamente rappresentazione base e rappresentazione estesa. Entrambe derivano dallo stesso traffico di partenza e seguono la stessa pipeline di preprocessing; la differenza è che la rappresentazione estesa aggiunge, a partire dalle caratteristiche estratte direttamente dai flussi, un insieme di feature derivate calcolate combinandole tra loro, che catturano aspetti del comportamento del flusso non immediatamente leggibili nelle statistiche grezze. La motivazione per considerare entrambe le rappresentazioni è verificare se e in quale misura la scelta delle feature influenza non solo le prestazioni del modello su traffico ordinario, ma anche la sua vulnerabilità alle perturbazioni e la sua risposta all'adversarial training. La costruzione delle due rappresentazioni è descritta nel dettaglio nel Capitolo 4; i loro effetti comparati sulle prestazioni sono analizzati nel Capitolo 5.

I modelli baseline. Il training set risultante dal preprocessing, che combina i flussi malevoli destinati all'addestramento con un sottoinsieme dei flussi benigni, costituisce la base su cui viene addestrato il classificatore. Durante la fase di addestramento, il classificatore apprende come distinguere traffico malevolo da traffico benigno, a partire dai campioni presenti nel training set. Il modello così ottenuto prende il nome di *clean* o *baseline*.

La costruzione del modello baseline non è un passo meramente preparatorio, ma svolge una funzione logica duplice nel framework. Da un lato, le sue prestazioni su traffico non perturbato stabiliscono la misura di riferimento rispetto alla quale tutti i risultati successivi vengono interpretati: verificare che l'adversarial training non degradi questa misura è uno degli obiettivi espliciti del lavoro. Dall'altro, le prestazioni del modello baseline su traffico perturbato rendono osservabile e quantificabile il problema che il framework mira a risolvere: senza un modello che mostri vulnerabilità alle perturbazioni in *traffic space*, non vi sarebbe motivazione per l'adversarial training. In questo senso, il modello baseline è lo strumento attraverso cui si rende concreta e misurabile la domanda che motiva l'intero lavoro: un ML-NIDS addestrato su traffico ordinario è davvero vulnerabile a perturbazioni realistiche in *traffic space*?

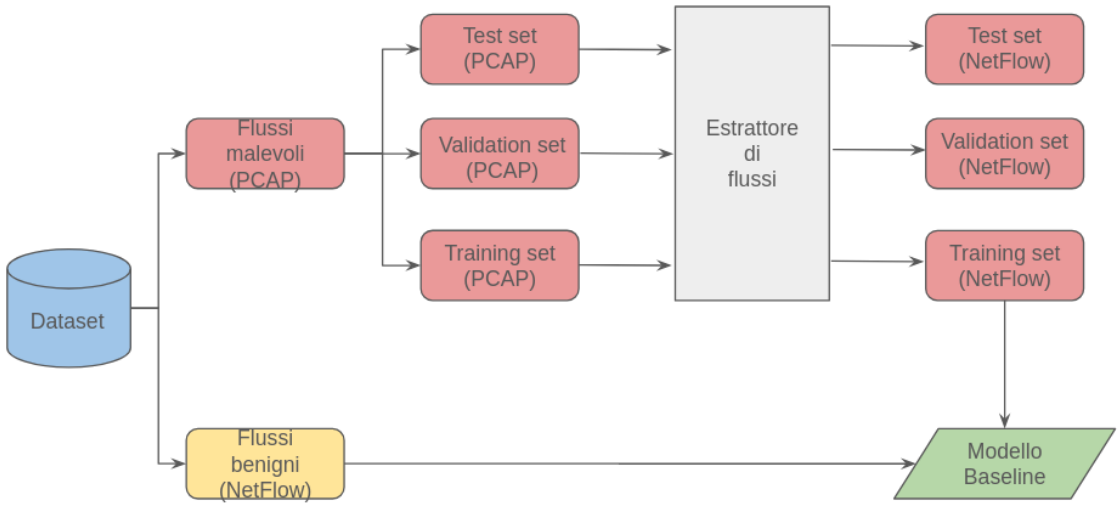

Figura 3.2: Il traffico malevolo viene suddiviso in tre insiemi disgiunti (training, validation e test set) e convertito in flussi. Il traffico benigno, già disponibile in forma di flussi, viene mantenuto come pool indifferenziato. Il modello baseline viene addestrato combinando il training set malevolo e i flussi benigni. Il validation set e il test set vengono riservati alle fasi successive del framework.

3.2.3 Generazione degli esempi adversariali in traffic space

La costruzione dei modelli baseline descritta nella sezione precedente produce un sistema di rilevamento che, nella sua configurazione ordinaria, non ha mai incontrato traffico intenzionalmente modificato per sfuggire alla classificazione.

Per costruire modelli più resistenti attraverso l'adversarial training, è necessario disporre di esempi di traffico malevolo perturbato: traffico che mantenga la propria natura malevola dal punto di vista funzionale, ma che sia stato modificato in modo da alterare le caratteristiche statistiche dei flussi su cui il classificatore opera.

La generazione di questi esempi adversariali non è un'operazione arbitraria: ogni scelta compiuta in questa fase discende direttamente dal threat model formalizzato nella Sezione 3.1, e in particolare dalla caratterizzazione dell'attaccante come soggetto che opera in *traffic space*, senza accesso alle componenti interne del rilevatore e senza la possibilità di costruire perturbazioni ottimizzate in funzione della struttura del modello.

La scelta del punto di applicazione delle perturbazioni. Il primo aspetto da chiarire riguarda il livello a cui le perturbazioni vengono introdotte. Come stabilito nel threat model, l'attaccante può intervenire esclusivamente sui pacchetti generati dagli host sotto il proprio controllo, e il risultato di questo intervento deve essere traffico valido dal punto di vista protocollare.

Questo vincolo determina una scelta che ha conseguenze profonde sulla struttura dell'intero esperimento: le perturbazioni vengono applicate direttamente sul traffico grezzo a livello di pacchetto (PCAP), non sulla rappresentazione a flussi (NetFlow). La distinzione è sostanziale. Se si modificassero i valori numerici già presenti nei NetFlow, si opererebbe in *feature space*, manipolando una rappresentazione astratta del traffico che non esiste come tale nella realtà della rete. Operare sui PCAP significa invece intervenire sul traffico reale a livello di pacchetto, e lasciare che i NetFlow vengano ricalcolati a partire dal traffico modificato, esattamente come accadrebbe se l'attaccante generasse traffico alterato sulla rete e il rilevatore lo osservasse e lo convertisse in flussi nel proprio normale ciclo operativo. È questa fedeltà al processo reale che garantisce che le perturbazioni si propaghino in modo autentico alle caratteristiche statistiche dei flussi, senza aggirare il processo di estrazione delle feature che il rilevatore effettua sul traffico osservato. Chiarito il livello a cui si interviene, occorre definire la natura concreta delle perturbazioni adottate.

La natura delle perturbazioni. Una volta chiarito che le perturbazioni vengono introdotte a livello di pacchetto, occorre definire di quale tipo esse siano e perché questa scelta sia coerente con il modello di minaccia adottato.

La questione non è banale: esistono molte forme in cui un attaccante potrebbe intervenire sul traffico generato dagli host che controlla, e non tutte sono ugualmente adatte allo scenario considerato. Per essere coerente con il threat model, una perturbazione deve soddisfare simultaneamente tre condizioni. La prima è che sia realizzabile da un attaccante black-box, ovvero che non presupponga alcuna conoscenza del sistema di rilevamento: non la struttura del modello, non l'algoritmo di classificazione adottato, non le feature estratte dal traffico, né alcun'altra componente interna del sistema difensivo.

La seconda condizione è che la perturbazione rispetti i vincoli protocollari del dominio, in modo che il traffico risultante sia valido a livello di rete e raggiunga effettivamente il rilevatore senza essere scartato dall'infrastruttura sottostante.

La terza condizione è che la perturbazione alteri le caratteristiche statistiche dei flussi in misura sufficiente da influenzare il comportamento del classificatore, senza però compromettere la funzionalità malevola della comunicazione che l'attaccante intende preservare: un'evasione che interrompe il canale di comando e controllo sarebbe controproducente per l'attaccante stesso.

Queste tre condizioni orientano la scelta verso perturbazioni che intervengono sulla dimensione dei pacchetti che trasportano dati applicativi, aggiungendo contenuto privo di significato alle comunicazioni malevole. Questa tipologia di manipolazione è applicabile senza alcuna conoscenza del sistema bersaglio, produce pacchetti validi dal punto di vista protocollare, altera le caratteristiche statistiche dei flussi, in misura rilevante per un classificatore basato su traffico aggregato, senza interferire con il corretto svolgimento

delle connessioni sottostanti. La *protocol-compliance* delle perturbazioni risultanti non è quindi un requisito accessorio, ma la condizione che rende l'intero approccio realistically deployabile in uno scenario operativo reale.

Le varianti di perturbazione e la loro motivazione. All'interno della tipologia di perturbazione descritta, il presente lavoro considera due varianti distinte, che differiscono per il modo in cui viene determinata l'entità della modifica applicata ai singoli pacchetti. La distinzione non è meramente tecnica, ma riflette due modelli di comportamento dell'attaccante concettualmente diversi, entrambi compatibili con il threat model black-box adottato.

La prima variante applica una modifica di entità costante a ciascun pacchetto perturbabile, indipendentemente dalle sue caratteristiche individuali. Questa variante modella uno scenario in cui l'attaccante adotta una strategia uniforme e parametrizzata: la modifica è sempre la stessa, replicabile sistematicamente su ogni pacchetto perturbato, senza che l'attaccante debba adattare il proprio comportamento al contesto.

La seconda variante introduce invece una componente stocastica: l'entità della modifica applicata a ciascun pacchetto viene determinata in modo casuale all'interno di un intervallo predefinito, in modo che pacchetti diversi ricevano modifiche di ampiezza variabile anche all'interno dello stesso flusso. In questo caso viene modellato uno scenario in cui l'attaccante introduce deliberata irregolarità nel traffico perturbato, riducendo la regolarità statistica della sequenza di pacchetti e rendendo il risultato meno riconoscibile come frutto di una manipolazione sistematica.

Entrambe le varianti vengono considerate in più istanze corrispondenti a intensità di modifica crescenti, in modo da esplorare un ventaglio ampio di possibili configurazioni di attacco e valutare come la robustezza del modello vari al variare dell'intensità della perturbazione. I parametri specifici sono presentati nel Capitolo 4.

La scelta di considerare entrambe le varianti risponde a una domanda con implicazioni pratiche dirette per il difensore: un modello hardened addestrato contro perturbazioni di una certa natura acquisisce robustezza anche nei confronti di perturbazioni strutturalmente diverse, o la sua resistenza è circoscritta alla specifica variante vista durante l'addestramento? La risposta a questa domanda, che emerge dall'analisi condotta nel Capitolo 5, ha ricadute rilevanti sulla praticabilità dell'adversarial training come strategia difensiva in scenari operativi reali, dove il difensore non può sapere in anticipo quale forma assumeranno le perturbazioni dell'attaccante.

Gli insiemi perturbati e il loro ruolo nel framework. Le perturbazioni descritte nelle sezioni precedenti vengono applicate a due dei tre insiemi in cui il traffico malevolo è stato suddiviso nella fase di costruzione della baseline: l'insieme di validazione e l'insieme di

test. La decisione di non perturbare l'insieme di addestramento è deliberata e risponde a una logica precisa che vale la pena esplicitare.

Preservare l'insieme di addestramento nella sua forma originale consente di mantenere una baseline pulita e confrontabile tra i modelli clean e i modelli hardened: l'unica differenza tra i due processi di addestramento è la presenza o l'assenza di esempi adversariali, il che rende il confronto diretto e interpretabile. Perturbare direttamente l'insieme di addestramento avrebbe invece mescolato l'effetto dell'adversarial training con variazioni nella distribuzione dei dati, rendendo più difficile isolare il contributo specifico della strategia difensiva rispetto ad altri fattori.

I due insiemi perturbati svolgono però ruoli completamente distinti nel framework, e questa distinzione è fondamentale per garantire la validità della valutazione. L'insieme di validazione perturbato costituisce il materiale con cui viene costruito il modello hardened: i flussi ottenuti dal traffico perturbato dell'insieme di validazione vengono integrati nel training set ampliato, in modo che il modello possa osservare esempi di traffico adversariale già durante la fase di apprendimento. In questo senso, l'insieme di validazione perturbato rappresenta il ponte tra la fase di generazione delle perturbazioni e la fase di adversarial training.

L'insieme di test perturbato svolge invece una funzione radicalmente diversa: viene riservato esclusivamente alla valutazione finale e non entra in alcun modo nel processo di addestramento. Questa separazione è il presupposto che rende la valutazione significativa: se il modello hardened venisse testato sugli stessi esempi adversariali su cui è stato addestrato, non si misurerebbe la sua capacità di generalizzare a perturbazioni nuove, ma semplicemente la sua capacità di memorizzare quelle già viste. Riservare il test set perturbato alla sola fase di valutazione garantisce invece che la robustezza misurata rifletta effettivamente la capacità del modello di classificare correttamente traffico adversariale che non ha mai incontrato.

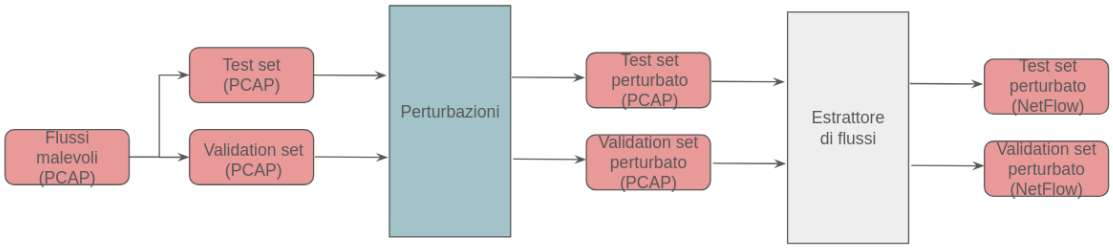

Figura 3.3: Pipeline di generazione degli esempi adversariali in *traffic space*. Il traffico malevolo grezzo appartenente all'insieme di validazione e all'insieme di test viene sottoposto a perturbazioni applicate a livello di pacchetto. Il traffico perturbato risultante viene convertito nella rappresentazione a flussi, producendo rispettivamente il validation set perturbato, utilizzato nella fase di adversarial training, e il test set perturbato, riservato esclusivamente alla valutazione finale dei modelli hardened. Il training set non viene perturbato, in modo da preservare una baseline pulita e garantire la confrontabilità tra modelli baseline e modelli hardened.

3.2.4 Adversarial training e costruzione dei modelli hardened

Le sezioni precedenti hanno descritto come vengono costruiti i modelli baseline e come viene generato il traffico adversariale. A questo punto del framework si dispone di tutti gli elementi necessari per affrontare la fase centrale del lavoro: l'adversarial training, ovvero il processo attraverso cui i modelli vengono rafforzati per resistere alle perturbazioni descritte nella sezione precedente. Prima di entrare nel merito di come questo processo viene realizzato, è utile richiamare brevemente le osservazioni che lo motivano.

Come anticipato nella Sezione 3.2.1, i modelli baseline vengono valutati sia su traffico non perturbato sia su traffico malevolo perturbato.

La prima valutazione, condotta su un insieme che include sia campioni malevoli sia campioni benigni, conferma che i modelli baseline raggiungono prestazioni elevate in condizioni ordinarie, come atteso da un sistema addestrato correttamente.

La seconda valutazione, condotta sottoponendo ai modelli il traffico malevolo perturbato, permette di osservare in quale misura le prestazioni si modificano rispetto al caso non perturbato. È a partire da questa osservazione che l'adversarial training trova la propria giustificazione empirica: se il modello baseline mostrasse già una robustezza naturale alle perturbazioni in *traffic space*, non vi sarebbe ragione di intervenire sulla procedura di addestramento.

La logica dell'adversarial training. L'idea alla base dell'adversarial training è concettualmente semplice: se un modello fatica a classificare correttamente traffico che non ha mai incontrato durante l'addestramento, la soluzione più diretta è esporlo a quel tipo di

traffico già durante la fase di apprendimento, in modo che impari a riconoscerlo. Nella sua formulazione generale, questa strategia consiste nell'ampliare il training set con esempi adversariali, affiancandoli agli esempi originali, e nell'addestrare il modello sull'insieme risultante [13, 16].

Il modello così ottenuto, che nel presente lavoro viene denominato *hardened*, ha avuto modo di osservare durante l'addestramento sia traffico nella sua forma ordinaria sia traffico perturbato, e dovrebbe quindi essere in grado di classificare correttamente entrambe le tipologie in fase operativa.

Nel contesto specifico di questo lavoro, l'adversarial training presenta una caratteristica che lo distingue dagli approcci più comuni in letteratura: poiché gli esempi adversariali vengono generati in *traffic space* secondo la logica descritta nella sezione precedente, i NetFlow che entrano nel training set del modello hardened sono ottenuti da un processo di estrazione delle feature identico a quello utilizzato per il traffico ordinario.

La costruzione del training set ampliato. Per costruire il training set del modello hardened, si combinano tre categorie di campioni: i campioni malevoli clean già utilizzati per addestrare il modello baseline, gli esempi adversariali nella loro rappresentazione NetFlow derivati dall'insieme di validazione perturbato, e i campioni benigni. L'insieme risultante viene sottoposto congiuntamente alla stessa pipeline di preprocessing utilizzata nella fase baseline, in modo da garantire la coerenza delle trasformazioni sull'intero corpus.

Al termine del preprocessing, il numero di campioni benigni da includere nel training set finale viene determinato in modo da mantenere la stessa proporzione realistica tra le due classi adottata nella fase baseline. Questa scelta assicura che il modello hardened non venga addestrato su una distribuzione artificialmente sbilanciata rispetto a quella del modello baseline, preservando la confrontabilità tra i due.

Il modello hardened viene quindi addestrato su questo training set ampliato, utilizzando gli stessi iperparametri del modello baseline. Mantenere invariati gli iperparametri è una scelta deliberata: garantisce che l'adversarial training sia l'unica variabile che differenzia il modello hardened dal modello baseline, rendendo il confronto tra i due diretto e interpretabile.

Vale la pena sottolineare che questo processo viene ripetuto separatamente per ciascuna combinazione di rappresentazione di feature e tipologia di perturbazione considerata nel lavoro, in modo da ottenere un insieme di modelli hardened specializzati, ciascuno addestrato contro una specifica variante di perturbazione. Non si costruisce quindi un unico modello hardened generico, ma tanti modelli distinti quante sono le tipologie di perturbazione investigate. Questa granularità è funzionale a una delle domande centrali del lavoro, ovvero se e in quale misura un modello hardened contro una specifica perturbazione acquisisca robustezza anche nei confronti di perturbazioni diverse da quelle

viste durante l'addestramento. I criteri adottati per selezionare gli scenari inclusi nella valutazione sono descritti nel Capitolo 4.

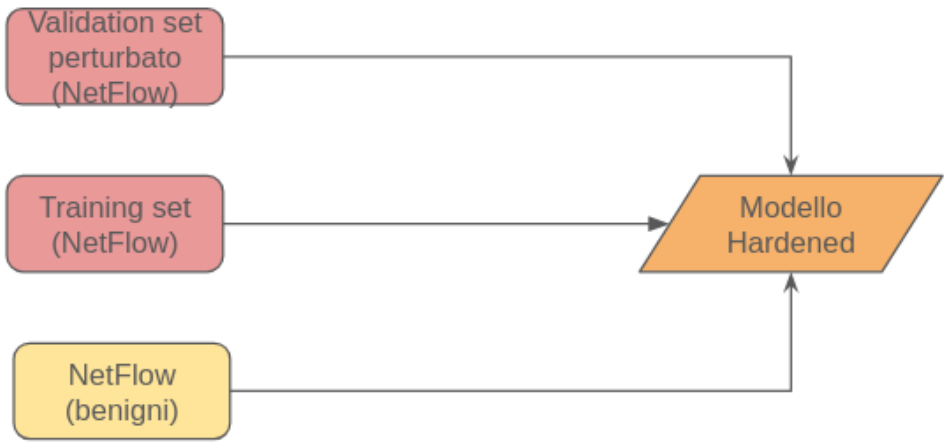

Figura 3.4: Pipeline di costruzione del modello hardened. Il training set ampliato viene composto combinando il training set malevolo clean, le rappresentazioni a flussi degli esempi adversariali derivati dall'insieme di validazione perturbato e i campioni benigni. Il modello hardened viene addestrato su questo insieme esteso, mantenendo invariati gli iperparametri del modello baseline.

La valutazione dei modelli hardened. Una volta addestrati i modelli hardened, la fase conclusiva del framework consiste nel valutarne le prestazioni e nel confrontarle con quelle dei modelli baseline. Come stabilito nel threat model, il difensore persegue due obiettivi simultanei; la valutazione è strutturata per verificarli entrambi in modo distinto e quantitativo.

Il primo obiettivo è verificare la robustezza acquisita: il modello hardened riesce a classificare correttamente il traffico malevolo perturbato meglio di quanto faccia il modello baseline? A tal fine, il modello hardened viene valutato sui test set perturbati, gli stessi che erano stati riservati esclusivamente alla valutazione finale e che nessun modello ha mai incontrato durante l'addestramento. Il confronto diretto tra le prestazioni del modello baseline e quelle del modello hardened sugli stessi test set perturbati fornisce una misura quantitativa dell'efficacia dell'adversarial training come strategia difensiva.

Il secondo obiettivo è verificare la non-regressione: il rafforzamento del modello non ha compromesso le sue prestazioni su traffico non perturbato? A tal fine, il modello hardened viene valutato anche sui test set clean, ovvero sugli stessi insiemi di traffico ordinario utilizzati per valutare il modello baseline. Un modello che acquisisse robustezza avversariale al prezzo di una perdita significativa di capacità di rilevamento sul traffico ordinario non sarebbe utile in un contesto operativo reale, dove il traffico non perturbato costituisce la componente largamente dominante di ciò che il sistema osserva. La verifica di questo

secondo obiettivo è quindi parte integrante della valutazione, e non una considerazione accessoria. I risultati di entrambe le valutazioni sono presentati e discussi nel Capitolo 5.

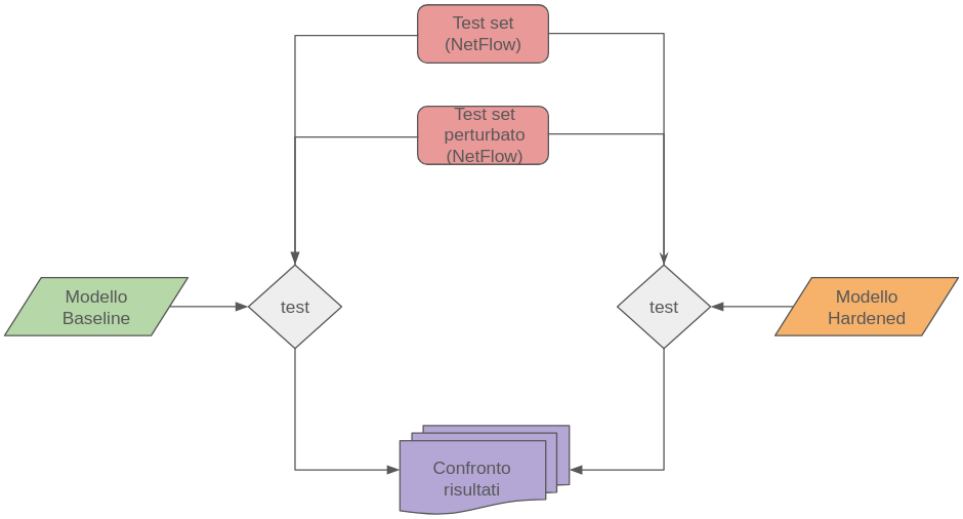

Figura 3.5: Schema di valutazione e confronto tra modello baseline e modello hardened. Entrambi i modelli vengono valutati sul test set clean, per verificare la non-regressione delle prestazioni su traffico ordinario, e sul test set perturbato, per misurare la robustezza acquisita dall'adversarial training. I risultati dei quattro confronti confluiscono in un'analisi comparativa discussa nel Capitolo 5.

3.2.5 Proprietà dell'approccio: semplicità, realismo, scalabilità

Le sezioni precedenti hanno descritto nel dettaglio le fasi del framework adottato, dalla costruzione dei dati alla generazione degli esempi adversariali fino all'addestramento e alla valutazione dei modelli hardened.

Prima di procedere con la descrizione implementativa nel capitolo successivo, vale la pena fare un passo indietro e ragionare sull'approccio nel suo complesso, non più dal punto di vista di cosa fa, ma dal punto di vista di cosa è: quali caratteristiche lo contraddistinguono, perché queste caratteristiche sono rilevanti in un contesto operativo reale, e in che modo si posiziona rispetto alla letteratura esistente.

La prima caratteristica che vale la pena sottolineare è la **semplicità**. Le perturbazioni adottate, non richiedono accesso al modello bersaglio, non presuppongono conoscenza delle feature estratte dal traffico, e non richiedono alcun processo di ottimizzazione iterativa. Si tratta di manipolazioni che un attaccante reale potrebbe applicare con strumenti comuni, senza competenze specializzate. Questa semplicità potrebbe sembrare una limitazione, come se il lavoro stesse affrontando solo la versione più elementare del problema. È invece una scelta metodologica consapevole: l'obiettivo non è costruire l'attacco più sofisticato possibile, ma verificare se perturbazioni semplici e non ottimizzate siano già sufficienti a degradare le prestazioni di un ML-NIDS, e se l'adversarial training sia

in grado di recuperare quella robustezza. Un risultato positivo in queste condizioni ha una rilevanza pratica immediata, poiché le perturbazioni considerate sono concretamente realizzabili da un attaccante reale senza competenze specializzate. Se il sistema difensivo proposto funziona contro perturbazioni semplici, la sua utilità pratica è immediata; se funzionasse solo contro perturbazioni elaborate, la sua applicabilità sarebbe limitata agli scenari in cui tale sofisticazione è effettivamente presente.

La seconda caratteristica è il **realismo**. Le perturbazioni operano in *traffic space*, agiscono sul traffico reale a livello di pacchetto, rispettano i vincoli protocollari e preservano la funzionalità malevola delle comunicazioni. Questo significa che le perturbazioni considerate nel presente lavoro non sono costrutti teorici, ma manipolazioni che un attaccante reale potrebbe introdurre nel traffico che genera. Il realismo non riguarda solo la natura delle perturbazioni, ma anche il modo in cui l'adversarial training viene condotto: gli esempi adversariali utilizzati durante l'addestramento del modello hardened sono NetFlow ottenuti convertendo traffico realmente perturbato a livello di pacchetto, e non vettori di feature modificati artificialmente. Questa coerenza tra il modo in cui le perturbazioni vengono generate e il modo in cui si presenterebbero in un sistema reale è uno degli elementi che distingue il presente lavoro dalla maggior parte della letteratura esistente sulle difese contro gli adversarial attack per ML-NIDS, che opera prevalentemente in *feature space* [3, 6, 4].

La terza caratteristica è la **scalabilità**. L'approccio adottato non dipende dall'architettura interna del classificatore: la pipeline di generazione degli esempi adversariali opera interamente a livello di traffico di rete, e il modello viene semplicemente addestrato su un training set ampliato con gli esempi così ottenuti. Questo significa che, in linea di principio, la stessa strategia difensiva potrebbe essere applicata a classificatori diversi da quello considerato in questo lavoro, senza modifiche alla pipeline di generazione delle perturbazioni. Analogamente, l'approccio è estendibile a scenari di traffico malevolo diversi da quello considerato in questo lavoro, e a tipologie di perturbazione diverse da quelle adottate, purché esse operino in *traffic space* e rispettino i vincoli di realizzabilità descritti nel threat model. Questa flessibilità strutturale suggerisce che il framework proposto non sia un sistema chiuso e specifico per un singolo scenario, ma un approccio generale che può essere adattato a contesti operativi diversi da quello sperimentale considerato in questo lavoro.

Queste tre proprietà, considerate congiuntamente, definiscono il posizionamento del contributo rispetto alla letteratura esistente. Non si tratta di proporre la difesa più sofisticata possibile, né di ottimizzare le prestazioni del classificatore al massimo livello raggiungibile. Si tratta piuttosto di dimostrare che un approccio semplice, realistico e scalabile in *traffic space* è praticabile, e che produce risultati concreti in termini di robustezza del rilevatore.

È in questo senso che il presente lavoro si propone come un primo passo pratico

nella direzione dell'adversarial training in *traffic space* per ML-NIDS, differenziandosi dagli approcci difensivi discussi nel Capitolo 2 che operano prevalentemente in *feature space* [6, 1].

Capitolo 4

Implementazione

Il Capitolo 3 ha presentato il framework proposto sul piano metodologico: il threat model, la struttura del processo di adversarial training e la logica delle perturbazioni in *traffic space*, senza entrare nei dettagli tecnici dell'esecuzione.

Il presente capitolo porta quella descrizione al livello implementativo, documentando le scelte concrete che hanno reso operativo l'approccio. Il punto di partenza è l'ambiente di esecuzione, descritto nella Sezione 4.1, che raccoglie le informazioni necessarie alla riproducibilità degli esperimenti, tra cui la macchina utilizzata, gli strumenti di sistema e le versioni delle librerie Python impiegate. Si passa poi alla Sezione 4.2, che descrive il dataset CTU-13 nella sua struttura tecnica e le due rappresentazioni di feature costruite a partire dai dati grezzi, motivando la scelta di valutarle in parallelo. Proseguendo, la Sezione 4.3 documenta il modello scelto e gli iperparametri impostati. A partire da questi elementi, la Sezione 4.4 illustra come l'intero approccio è stato messo in pratica, descrivendo gli aspetti tecnici delle scelte e delle implementazioni adottate.

4.1 Ambiente di esecuzione

4.1.1 Hardware e sistema operativo

Tutti gli esperimenti descritti in questo lavoro sono stati condotti su una singola macchina con processore Intel Core i7-6500U a 2,50 GHz, 8 GB di memoria RAM e uno storage complessivo di 500 GB. Si tratta di una configurazione *consumer-grade*, non specializzata per il calcolo scientifico ad alte prestazioni: nessuna GPU è stata impiegata, né si è fatto ricorso a infrastrutture di calcolo distribuito o cluster. Questa scelta riflette un obiettivo preciso del lavoro, ossia dimostrare che l'approccio proposto è praticabile anche in assenza di risorse computazionali dedicate, rendendo i risultati riproducibili in ambienti ordinari. Il sistema operativo adottato è Debian GNU/Linux 12 (*bookworm*), distribuzione *stable*. La scelta di una distribuzione Linux stabile è motivata principalmente dalla natura della pipeline sperimentale, che si articola, oltre che di script di codice, anche di operazio-

ni a riga di comando e richiede l'uso di strumenti di sistema, come `tcpdump` e Argus, che trovano nel mondo Unix il loro ambiente naturale. Una distribuzione in modalità *stable*, inoltre, garantisce la riproducibilità degli esperimenti nel tempo, eliminando il rischio che aggiornamenti intermedi del sistema alterino il comportamento degli strumenti impiegati.

4.1.2 Strumenti di sistema

La pipeline sperimentale si appoggia a quattro strumenti di sistema che operano direttamente sui file PCAP in fasi distinte del processo di elaborazione.

Il primo è `mergecap`, un'utility inclusa nella suite Wireshark che consente di unire più file PCAP in un unico file di output. Nel presente lavoro viene utilizzato per aggregare i PCAP dei tredici scenari del CTU-13 in sette file distinti, uno per ciascuna famiglia di botnet.

Il secondo è `pcapfix`, uno strumento a riga di comando che individua e corregge anomalie strutturali nei file PCAP, come header malformati o record troncati. Il suo impiego garantisce l'integrità dei file PCAP prima che vengano elaborati dagli strumenti successivi della pipeline.

Il terzo è `tcpdump`, utilizzato nella fase iniziale di preparazione dei dati per filtrare il traffico TCP dai PCAP aggregati per famiglia di botnet, scartando tutti i pacchetti appartenenti ad altri protocolli. Poiché l'intero lavoro si concentra esclusivamente sul traffico TCP, applicare questo filtraggio a monte consente di ridurre fin da subito le dimensioni dei file su cui operano gli step successivi della pipeline.

Il quarto strumento è Argus, nella versione 3.0.8.2, impiegato in una fase successiva per convertire i file PCAP in record NetFlow in formato CSV. Argus è una suite composta da due componenti distinti che vengono invocati in sequenza: il primo, `argus`, legge un PCAP grezzo e produce un file binario intermedio nel formato proprietario Argus; il secondo, `ra`, legge quel file binario e lo converte in un CSV con i campi NetFlow di interesse. La versione 3.0.8.2 è la stessa adottata in [5], lavoro dal quale questa tesi riprende, tra le altre cose, la pipeline di conversione PCAP-NetFlow. Mantenere la stessa versione garantisce che la componente di estrazione dei flussi si comporti in modo allineato con quella del riferimento metodologico, riducendo le differenze di configurazione che potrebbero influenzare i valori delle feature estratte e rendere meno agevole qualsiasi confronto con i risultati di quel lavoro."

4.1.3 Librerie Python

Il codice sviluppato in questo lavoro è scritto interamente in Python, nella versione 3.11, e si appoggia a un insieme di librerie che coprono le diverse fasi della pipeline sperimentale. La scelta di Python come linguaggio principale è motivata dalla disponibilità di un ecosistema maturo e consolidato per il trattamento di dati tabulari e per l'implementazione di

modelli di machine learning, oltre che da librerie che consentono di lavorare direttamente con pacchetti di rete.

Tra queste ultime, un ruolo centrale è ricoperto da Scapy, nella versione 2.7.0. Scapy è una libreria Python che permette di leggere, costruire e modificare pacchetti di rete a livello granulare, operando direttamente sui campi degli header dei vari protocolli dello stack TCP/IP. Nel contesto di questo lavoro, Scapy viene impiegata nella fase di perturbazione dei PCAP in modo tale da generare nuove tracce di traffico che contengano al loro interno traffico perturbato, che potrà essere successivamente convertito in NetFlow e utilizzato nelle fasi di valutazione e di addestramento avversariale.

Il trattamento dei dati tabulari prodotti dalla conversione Argus è invece affidato a Pandas, nella versione 3.0.0. Pandas è una libreria orientata alla manipolazione di dati strutturati in forma tabellare e rappresenta lo strumento di riferimento per tutte le operazioni di preprocessing sui CSV NetFlow: dalla lettura e pulizia dei dati grezzi, alla trasformazione delle feature categoriche, fino alla costruzione dei dataset finali utilizzati per l'addestramento e la valutazione dei modelli.

L'addestramento e la valutazione del classificatore sono realizzati tramite Scikit-learn, nella versione 1.8.0. Scikit-learn è la libreria Python più diffusa per il machine learning su dati tabulari e fornisce implementazioni efficienti di un'ampia gamma di algoritmi, tra cui il Random Forest adottato in questo lavoro. Oltre all'addestramento, Scikit-learn viene utilizzato per il calcolo delle metriche di valutazione sui set di test. La serializzazione dei modelli addestrati, necessaria per poterli riutilizzare nelle fasi successive della pipeline senza doverli riaddestrare, è gestita tramite joblib, nella versione 1.5.3, una libreria specializzata nel salvataggio e nel caricamento efficiente di oggetti Python di grandi dimensioni, come appunto i modelli di machine learning.

4.2 Dataset e rappresentazioni di feature

4.2.1 Il dataset CTU-13

Il dataset CTU-13 [18] è una raccolta di traffico di rete reale prodotta dal gruppo Stratosphere IPS presso la Czech Technical University di Praga nel 2013. Nel corso degli anni, il CTU-13 è diventato uno dei dataset di riferimento più utilizzati in questo ambito, e la sua adozione è consolidata tanto nella letteratura sugli ML-NIDS quanto in quella sull'adversarial machine learning applicato al traffico di rete, come testimoniato dai principali lavori analizzati nel Capitolo 2.

Il dataset è organizzato in tredici scenari distinti, appartenenti a sette famiglie di botnet. Ogni scenario documenta una campagna di infezione da parte di una specifica famiglia, catturata in condizioni controllate ma su traffico di rete reale, e ciascuna famiglia può essere presente in uno o più scenari. Le sette famiglie considerate nel presente lavoro

sono Neris, Rbot, Virut, Murlo, NSIS, Menti e Sogou. Si tratta di famiglie con comportamenti di rete eterogenei: alcune si caratterizzano per attività di spam e phishing, altre per comunicazioni strutturate con server di comando e controllo (C&C) tramite protocolli IRC o HTTP, altre ancora per attività di click fraud o propagazione tramite sfruttamento di vulnerabilità. Questa varietà rende il CTU-13 un banco di prova rappresentativo di una gamma ampia di minacce, e non limitato a un singolo pattern comportamentale. Ogni scenario include sia traffico malevolo, generato dagli host compromessi, sia traffico benigno, prodotto dagli altri host della rete durante la stessa finestra temporale di cattura. Questa coesistenza di traffico malevolo e benigno all'interno degli stessi scenari è una delle caratteristiche che rendono il CTU-13 particolarmente adatto per lo studio degli ML-NIDS: i dati riflettono le condizioni operative reali in cui un rilevatore si trova a operare, dove il traffico malevolo è sempre immerso in un contesto di comunicazioni legittime largamente dominanti.

Il CTU-13 distribuisce i propri dati in due formati distinti, che nel presente lavoro vengono impiegati per scopi diversi. Il traffico benigno è disponibile nei file *binetflow*, un formato tabulare in cui ogni riga rappresenta un flusso di rete bidirezionale descritto da un insieme di attributi statistici estratti tramite Argus. Il traffico malevolo è invece disponibile in formato PCAP (*Packet Capture*), il formato standard per la registrazione e l'archiviazione del traffico di rete a livello di pacchetto, che preserva ciascun pacchetto nella sua forma originale, inclusi header e payload. Questa distinzione di formato non è un dettaglio secondario: poiché l'approccio proposto in questo lavoro opera in *traffic space*, le perturbazioni vengono applicate direttamente sui pacchetti di rete, e ciò richiede necessariamente che il traffico malevolo sia disponibile nella forma grezza del PCAP. Il traffico benigno, non essendo oggetto di alcuna perturbazione, può essere utilizzato direttamente nella rappresentazione NetFlow in cui è distribuito.

La scelta del CTU-13 come dataset di riferimento per questo lavoro è motivata da un insieme di considerazioni di natura tecnica, etica e metodologica. In primo luogo, la disponibilità dei PCAP relativi al solo traffico malevolo è un requisito imprescindibile per qualsiasi approccio in *traffic space*: senza accesso ai pacchetti grezzi, non sarebbe possibile applicare perturbazioni a livello di rete nel rispetto dei vincoli di realizzabilità che caratterizzano il lavoro, e questa caratteristica non è comune a tutti i dataset di traffico disponibili in letteratura, molti dei quali distribuiscono i dati esclusivamente in rappresentazioni tabulari pre-elaborate. In secondo luogo, il dataset è pubblico e privo di dati personali identificabili, condizione necessaria per un lavoro di ricerca di questo tipo. In terzo luogo, la sua diffusione nella letteratura di riferimento [5, 19] consente un confronto, anche indiretto, con i risultati di lavori che hanno adottato lo stesso banco di prova sperimentale.

4.2.2 Rappresentazione base

I dati grezzi estratti dal traffico di rete non possono essere forniti direttamente in input a un classificatore che utilizza un algoritmo di machine learning nella forma in cui vengono prodotti da Argus. Valori come indirizzi IP, numeri di porta e flag di stato TCP, nella loro rappresentazione originale, non si prestano a essere trattati come variabili numeriche ordinarie: un indirizzo IP come `147.32.84.165` non ha una relazione ordinale con un altro indirizzo, e trattarlo come un intero introdurrebbe nel modello una struttura artificiale che non corrisponde ad alcuna proprietà reale del traffico.

Per questo motivo, prima di poter addestrare il classificatore, è necessario trasformare i campi grezzi in una rappresentazione strutturata di feature che catturi le proprietà rilevanti del flusso in una forma numerica coerente.

La rappresentazione base, denominata nel presente lavoro NO_FD *(senza feature derivate)*, è composta da **36 feature** ed è ispirata all'approccio di preprocessing proposto nel lavoro DReLAB [19], che adotta una logica analoga per la costruzione del vettore di input dei propri classificatori. Le feature sono ottenute trasformando i 16 campi grezzi estratti da Argus tramite una serie di operazioni di encoding e binarizzazione, la cui procedura tecnica è presentata nella Sezione 4.4.

Il protocollo di trasporto viene codificato come variabile intera, con il valore `0` assegnato a TCP, che è l'unico protocollo considerato nel presente lavoro per le ragioni discusse nella Sezione 4.4.

Gli indirizzi IP sorgente e destinazione vengono invece trasformati in due flag binari, `IPSrcType` e `IPDstType`, che indicano rispettivamente se l'indirizzo sorgente e quello destinazione appartengono alla sottorete interna della rete CTU-13 (`147.32.0.0/16`) oppure a un indirizzo esterno. Questa scelta, mutuata direttamente da DReLAB [19], evita che il classificatore apprenda a distinguere il traffico malevolo da quello benigno sulla base di indirizzi IP specifici, un comportamento che porterebbe a un overfitting stretto sul dataset di training e comprometterebbe la generalizzabilità del modello su traffico proveniente da reti diverse.

Un ragionamento analogo si applica alle porte. I numeri di porta sorgente e destinazione vengono ciascuno trasformato in tre flag binari mutuamente esclusivi, seguendo la tassonomia IANA: `WellKnown` per le porte nel range 0–1023, tipicamente associate a servizi di sistema standardizzati; `Registered` per il range 1024–49151, associato a servizi applicativi noti; `Private` per il range 49152–65535, tipicamente utilizzato per connessioni dinamiche o effimere. In questo modo si preserva l'informazione semantica sul tipo di porta coinvolta nella comunicazione senza introdurre la dipendenza dal valore numerico specifico, che varierebbe da sessione a sessione e da rete a rete.

La direzione del flusso, originariamente codificata in Argus come stringa simbolica, viene espansa tramite one-hot encoding in quattro colonne binarie, corrispondenti alle

quattro modalità possibili: `->` (flusso unidirezionale in uscita), `?>` (direzione incerta in uscita), `<?` (direzione incerta in ingresso) e `<?>` (direzione bidirezionale non determinata).

Lo stato della connessione TCP, che Argus rappresenta come una stringa compatta di flag per la direzione sorgente e per quella destinazione, viene scomposto in sedici feature binarie, otto per ciascuna direzione, una per ciascun flag TCP possibile: A (*ACK*, conferma della ricezione di dati), C (*CWR*, segnalazione di riduzione della finestra di congestione), E (*ECE*, notifica esplicita di congestione nella rete), F (*FIN*, richiesta di chiusura ordinata della connessione), P (*PSH*, indicazione che i dati devono essere consegnati immediatamente all'applicazione senza attendere il riempimento del buffer), R (*RST*, reset e chiusura forzata della connessione), S (*SYN*, avvio del processo di handshake per l'apertura di una nuova connessione) e U (*URG*, presenza di dati urgenti da processare con priorità). Ciascuna feature vale `1` se il flag corrispondente è presente nello stato della connessione, `0` altrimenti.

Le restanti feature della rappresentazione base sono numeriche e vengono mantenute nella loro forma originale: il tipo di servizio sorgente e destinazione (`sTos`, `dTos`), la durata del flusso (`Dur`), il numero totale di pacchetti (`TotPkts`), il numero totale di byte scambiati (`TotBytes`), i byte trasmessi dalla sorgente (`SrcBytes`) e i byte ricevuti dalla destinazione (`DstBytes`).

4.2.3 Rappresentazione estesa

La rappresentazione base descritta nella sezione precedente cattura le proprietà strutturali e protocollari di ciascun flusso, ma non include alcuna misura di intensità o di efficienza della comunicazione. Sapere quanti byte sono stati scambiati in un flusso, ad esempio, fornisce un'informazione utile, ma non dice nulla su quanto velocemente quei byte siano stati trasmessi, né su quanto siano distribuiti asimmetricamente tra le due direzioni della comunicazione. Queste grandezze derivate possono rivelarsi discriminanti importanti per distinguere traffico malevolo da traffico benigno, poiché le botnet tendono a produrre pattern di comunicazione caratteristici in termini di throughput e asimmetria del traffico.

Per questa ragione, accanto alla rappresentazione base viene costruita una seconda rappresentazione, denominata FD (con feature derivate), che estende le 36 feature di NO_FD con l'aggiunta di quattro feature calcolate a partire da quelle già presenti. Questa scelta è ispirata all'approccio proposto in DReLAB [19], che introduce un insieme analogo di feature derivate con la medesima motivazione.

Le feature derivate adottate nel presente lavoro sono le stesse proposte in quel lavoro, ma la pipeline di preprocessing in cui vengono integrate presenta alcune differenze rispetto all'originale, descritte nella Sezione 4.4. La rappresentazione FD conta pertanto **40 feature** in totale.

Le quattro feature aggiunte sono le seguenti. La prima è **BytesPerPkt**, definita come il rapporto tra il numero totale di byte scambiati e il numero totale di pacchetti del flusso:

$$\texttt{BytesPerPkt} = \frac{\texttt{TotBytes}}{\texttt{TotPkts}} \tag{4.1}$$

Questa feature misura la dimensione media dei pacchetti nel flusso e può riflettere la natura del traffico: flussi con pacchetti di dimensione costante e ridotta sono tipici di comunicazioni C&C, mentre flussi con pacchetti grandi sono più frequenti in trasferimenti di dati legittimi.

La seconda è **BytesPerSec**, definita come il rapporto tra il numero totale di byte e la durata del flusso:

$$\texttt{BytesPerSec} = \frac{\texttt{TotBytes}}{\texttt{Dur}} \tag{4.2}$$

Questa feature misura il throughput medio del flusso, ovvero la quantità di dati trasmessa nell'unità di tempo. Flussi con throughput molto elevato possono indicare trasferimenti massivi di dati, mentre flussi con throughput basso e costante sono caratteristici di comunicazioni periodiche a bassa intensità, come quelle tipiche di un bot che esegue il polling del proprio server C&C.

La terza è **PktsPerSec**, definita come il rapporto tra il numero totale di pacchetti e la durata:

$$\texttt{PktsPerSec} = \frac{\texttt{TotPkts}}{\texttt{Dur}} \tag{4.3}$$

Questa feature misura la frequenza di trasmissione dei pacchetti nell'unità di tempo. A differenza di `BytesPerSec`, che riflette il volume di dati, `PktsPerSec` cattura il ritmo della comunicazione indipendentemente dalla dimensione dei singoli pacchetti: due flussi con lo stesso throughput possono avere frequenze di pacchetto molto diverse se i pacchetti hanno dimensioni differenti, e questa distinzione può essere rilevante per distinguere tipologie di traffico con pattern temporali caratteristici.

La quarta è **RatioOutIn**, definita come il rapporto tra i byte trasmessi dalla sorgente e i byte ricevuti dalla destinazione:

$$\texttt{RatioOutIn} = \frac{\texttt{SrcBytes}}{\texttt{DstBytes}} \tag{4.4}$$

Questa feature misura l'asimmetria direzionale del flusso. Un valore elevato indica che la sorgente trasmette molto più di quanto riceva, comportamento tipico di alcune categorie di traffico malevolo come l'esfiltrazione di dati o la comunicazione con server C&C che rispondono con messaggi brevi a comandi più lunghi.

Le tre feature `BytesPerSec`, `PktsPerSec` e `RatioOutIn` possono assumere valore infinito quando il denominatore è zero, situazione che si verifica per flussi di durata nulla o privi di traffico in direzione destinazione. Seguendo la pratica documentata in DRe-

LAB [19], i valori infiniti vengono sostituiti con il massimo valore finito della feature calcolato sull'insieme dei dati disponibili. I dettagli tecnici di questa operazione, così come i filtri applicati per rimuovere campioni con valori anomali nelle feature numeriche, sono descritti nella Sezione 4.4.

La scelta di valutare entrambe le rappresentazioni in parallelo, anziché adottarne una sola, risponde a una precisa esigenza sperimentale e segue la logica già adottata in DRe-LAB [19]. Le feature derivate arricchiscono il profilo statistico del flusso con informazioni che la rappresentazione base non contiene, e l'obiettivo è verificare se questa maggiore ricchezza informativa si traduca in un beneficio misurabile: sia in termini di capacità discriminativa del classificatore su traffico non perturbato, sia in termini di robustezza dopo l'adversarial training. Confrontare i risultati delle due rappresentazioni consente di rispondere a questa domanda in modo diretto, e di trarre conclusioni più articolate sull'efficacia dell'approccio proposto.

4.3 Classificatore

4.3.1 Il Random Forest come scelta per ML-NIDS

Il classificatore adottato nel presente lavoro è il *Random Forest* (RF), un modello di apprendimento supervisionato appartenente alla famiglia degli *ensemble methods*, ovvero quei metodi che costruiscono le proprie predizioni combinando i risultati di un insieme di modelli più semplici. Nel caso del Random Forest, questi modelli più semplici sono alberi decisionali: strutture che apprendono come classificare i dati ponendo una sequenza di domande sulle feature di ingresso e producendo ciascuna una predizione finale. Le predizioni dei singoli alberi vengono poi aggregate per voto di maggioranza, in modo che la classificazione risultante rifletta il consenso dell'intero insieme.

L'idea alla base del Random Forest è che un insieme numeroso di alberi, ciascuno addestrato su una porzione diversa dei dati e con accesso a un sottoinsieme casuale delle feature disponibili, produca collettivamente predizioni più stabili e generalizzabili di quelle che un singolo albero potrebbe offrire. Un albero decisionale addestrato senza vincoli tende ad adattarsi eccessivamente ai dati di training, memorizzando pattern specifici del campione osservato e perdendo capacità di generalizzazione su dati nuovi: questo fenomeno è noto come *overfitting*. Il meccanismo di aggregazione del Random Forest mitiga questo comportamento, poiché gli errori individuali dei singoli alberi tendono a compensarsi nel voto di maggioranza, e il risultato complessivo risulta più robusto.

La scelta del Random Forest come classificatore di riferimento per questo lavoro non è arbitraria, ma riflette la consolidata presenza di questo modello nella letteratura sui sistemi di rilevamento di intrusioni basati su machine learning [19, 5]. Il Random Forest è infatti impiegato con continuità in lavori che affrontano la rilevazione di botnet e la robustezza

adversariale di questi sistemi, e questa frequenza è motivata da alcune sue caratteristiche fondamentali.

Una prima caratteristica è la capacità di operare direttamente su dati tabulari eterogenei senza richiedere normalizzazione delle feature. I dati prodotti dall'estrazione di feature da flussi di rete combinano feature numeriche continue, feature numeriche intere e feature categoriali codificate, spesso senza una scala comune tra loro. Il Random Forest non richiede che le feature siano normalizzate o scalate, poiché le divisioni negli alberi sono effettuate sulla base di soglie calcolate localmente su ciascuna feature, indipendentemente dalla sua scala. Questo lo distingue da modelli come le reti neurali o le macchine a vettori di supporto, che risultano invece sensibili alla distribuzione e alla scala delle feature in ingresso e richiedono pertanto un preprocessing più accurato prima dell'addestramento.

Una seconda caratteristica è la robustezza alla presenza di feature ridondanti o scarsamente informative. La randomizzazione nella selezione delle feature a ogni nodo fa sì che nessuna singola feature possa dominare sistematicamente il processo di apprendimento, distribuendo il contributo discriminativo sull'intero insieme disponibile. In un contesto come quello dei NetFlow, dove alcune feature possono risultare più direttamente manipolabili da un attaccante rispetto ad altre, questa proprietà contribuisce a rendere il modello intrinsecamente meno esposto a perturbazioni che agiscono su un sottoinsieme ristretto delle feature.

Una terza caratteristica, di natura più pratica, è la relativa semplicità di configurazione e addestramento rispetto ad architetture più complesse. Il Random Forest non richiede la definizione di architetture a strati, funzioni di attivazione o procedure di ottimizzazione iterativa basate su gradienti, e il suo comportamento è interamente controllato da un insieme limitato di iperparametri con interpretazione intuitiva. Questa semplicità facilita la riproducibilità degli esperimenti e la comprensione del comportamento del modello, aspetti particolarmente rilevanti in un contesto di ricerca come quello del presente lavoro.

4.3.2 Provenienza e motivazione degli iperparametri

La configurazione degli iperparametri del Random Forest adottata nel presente lavoro è stata ripresa dal codice open-source associato al lavoro di riferimento [5], che opera nello stesso dominio applicativo e sullo stesso dataset CTU-13. Questa scelta garantisce che i valori utilizzati siano già stati validati in un contesto analogo a quello del presente lavoro, riducendo il rischio di introdurre scelte arbitrarie che potrebbero influenzare i risultati in modo difficilmente interpretabile. La scelta è inoltre coerente con il lavoro DReLAB [19], che costituisce un ulteriore punto di riferimento metodologico per il presente lavoro e adotta una configurazione del Random Forest sostanzialmente analoga per task di classificazione su NetFlow estratti dallo stesso dataset.

Un ulteriore aspetto su cui porre l'attenzione riguarda la scelta di non condurre alcun processo di ottimizzazione sistematica degli iperparametri, detto anche tuning o fine tuning. Questa è una decisione deliberata, motivata dalla natura dell'obiettivo del lavoro. L'oggetto centrale della sperimentazione è l'effetto dell'adversarial training come strategia difensiva, e non l'ottimizzazione del classificatore di base. Avviare una ricerca estesa nello spazio delle possibili configurazioni avrebbe introdotto un secondo asse di variabilità nell'analisi, rendendo più difficile attribuire le differenze di prestazione osservate all'effetto delle perturbazioni piuttosto che a differenze nella configurazione del modello. Fissare gli iperparametri a priori, sulla base di valori già consolidati in letteratura, consente invece di isolare con maggiore chiarezza il contributo specifico dell'adversarial training sui risultati sperimentali.

4.3.3 Descrizione degli iperparametri

Una volta stabilita la provenienza degli iperparametri, è opportuno descriverne il significato e motivare i valori adottati, segnalando al contempo le due modifiche introdotte rispetto alla configurazione originale di [5]. Di seguito si illustrano i parametri più rilevanti dal punto di vista concettuale, rimandando alla Tabella 4.1 per una visione d'insieme completa.

Il primo parametro di rilievo è `n_estimators`, che stabilisce il numero di alberi decisionali che compongono il modello. Nel presente lavoro questo valore è fissato a 200: un numero sufficientemente elevato da garantire predizioni stabili grazie all'aggregazione di molti alberi, senza introdurre oneri computazionali eccessivi.

Il parametro `criterion` specifica il criterio con cui il modello valuta, durante l'addestramento, quale suddivisione dei dati sia più utile per separare le classi. Il valore adottato è `'gini'`, una misura comunemente adottata nei problemi di classificazione che quantifica quanto è eterogenea la distribuzione delle classi in un nodo. Ad ogni passo, l'algoritmo seleziona la suddivisione che riduce maggiormente questa eterogeneità.

Il parametro `max_depth` controlla la profondità massima consentita a ciascun albero. Il valore `None` indica che non viene imposto alcun vincolo, lasciando al modello piena libertà di catturare anche pattern complessi nei dati. Il rischio che questa libertà porti il modello ad adattarsi eccessivamente ai dati di training è mitigato dal fatto che le predizioni finali derivano dall'aggregazione di molti alberi diversi tra loro, ciascuno dei quali tende a commettere errori diversi, portando quindi ad una compensazione gli errori dei singoli alberi.

Il parametro `max_features` determina quante feature vengono considerate in ciascun nodo nella ricerca della suddivisione più efficace. Il valore `'sqrt'` indica che, in ogni nodo, viene esaminato un sottoinsieme di feature pari alla radice quadrata del numero totale di feature disponibili. Come descritto in precedenza, questa randomizzazione è

uno dei fattori che garantisce la diversità tra i singoli alberi e contribuisce alla robustezza complessiva del modello. Nel paper di riferimento [5] questo parametro è impostato al valore `'auto'`, che nelle versioni storiche di scikit-learn era definito come sinonimo di `'sqrt'` per i problemi di classificazione. Nelle versioni più recenti della libreria il valore `'auto'` è stato rimosso, rendendo necessario l'utilizzo esplicito di `'sqrt'`: la modifica è quindi imposta dall'API corrente di scikit-learn e non costituisce una scelta progettuale, poiché il comportamento del modello rimane identico a quello del paper di riferimento.

Il parametro `bootstrap` stabilisce se ciascun albero viene addestrato sull'intero training set oppure su una sua porzione estratta casualmente. Il valore `True` fa sì che ogni albero veda una versione leggermente diversa dei dati, introducendo quella variabilità tra gli alberi che è alla base della stabilità del modello complessivo.

Il parametro `n_jobs`, impostato a `2`, abilita la parallelizzazione dell'addestramento su due thread, riducendo i tempi di esecuzione senza influenzare il comportamento del modello.

Infine, il parametro `random_state` fissa il seme del generatore di numeri casuali utilizzato durante l'addestramento. Il valore adottato è `0`, in luogo del valore `None` presente nel paper di riferimento [5]. Con `None`, ogni esecuzione dell'addestramento produce un modello leggermente diverso poiché la sorgente di casualità non è fissata; portare il valore a `0` garantisce che due esecuzioni con gli stessi dati di input producano esattamente lo stesso modello. Questa modifica non influenza le prestazioni attese del classificatore.

I parametri non discussi nel testo sono lasciati ai valori di default dell'implementazione scikit-learn, in quanto privi di rilevanza concettuale nel contesto del presente lavoro. La Tabella 4.1 riporta la configurazione completa del classificatore.

Parametro	Valore	Ruolo
n_estimators	200	Numero di alberi nel modello
criterion	'gini'	Funzione di impurità per le divisioni
max_depth	None	Profondità massima degli alberi (nessun vincolo)
max_features	'sqrt'	Feature considerate a ogni nodo
bootstrap	True	Campionamento con reinserimento
random_state	0	Seme per la riproducibilità
min_samples_split	2	Default scikit-learn
min_samples_leaf	1	Default scikit-learn
min_weight_fraction_leaf	0.0	Default scikit-learn
max_leaf_nodes	None	Default scikit-learn
min_impurity_decrease	0.0	Default scikit-learn
oob_score	False	Default scikit-learn
n_jobs	2	Thread per la parallelizzazione
warm_start	False	Default scikit-learn
class_weight	None	Default scikit-learn
ccp_alpha	0.0	Default scikit-learn
max_samples	None	Default scikit-learn

Tabella 4.1: Configurazione completa del classificatore Random Forest adottato nel presente lavoro. I parametri contrassegnati come *default scikit-learn* sono lasciati al valore predefinito dell'implementazione e non hanno rilevanza concettuale nel contesto del lavoro.

4.3.4 Uniformità della configurazione tra modelli clean e hardened

La configurazione degli iperparametri descritta nei paragrafi precedenti è adottata in modo uniforme per tutti i modelli addestrati nel presente lavoro, senza alcuna distinzione tra i modelli baseline, addestrati esclusivamente su traffico pulito, e i modelli hardened, addestrati su un training set ampliato con esempi adversariali. Questa scelta non è casuale, ma risponde a una precisa esigenza metodologica.

L'obiettivo centrale della sperimentazione è misurare l'effetto dell'adversarial training sulla robustezza del classificatore. Affinché questo confronto sia diretto e interpretabile, è necessario che i modelli baseline e hardened differiscano tra loro per una sola variabile, ovvero la composizione del training set su cui sono stati addestrati. Se i due tipi di modello fossero addestrati con configurazioni di iperparametri diverse, qualsiasi differenza di prestazione osservata nei risultati potrebbe essere attribuita tanto all'effetto dell'adversarial training quanto alle differenze di configurazione, rendendo impossibile isolare il contributo specifico della strategia difensiva.

Fissare la stessa configurazione per tutti i modelli garantisce quindi che i risultati presentati nel Capitolo 5 siano direttamente confrontabili tra loro, e che le differenze di prestazione osservate tra modelli baseline e hardened possano essere attribuite con ragionevole certezza all'effetto dell'adversarial training.

4.4 Descrizione dell'implementazione dell'approccio

Le sezioni precedenti di questo capitolo hanno introdotto l'ambiente di esecuzione, il dataset e il classificatore adottato. La presente sezione descrive come questi elementi sono stati uniti tra loro per realizzare concretamente il framework proposto nel Capitolo 3, seguendo l'ordine cronologico delle tre fasi operative del lavoro: la costruzione e l'addestramento del modello baseline su traffico non perturbato, la generazione del traffico adversariale in *traffic space*, e l'adversarial training dei modelli hardened. Il codice sorgente relativo a tutte le fasi del lavoro sperimentale è disponibile pubblicamente nel repository GitHub del progetto [17].

4.4.1 Fase 1 - Costruzione e addestramento del modello baseline

Preparazione dei dati malevoli e suddivisione in split Come discusso nel Capitolo 3, il traffico malevolo del dataset CTU-13 è distribuito in formato PCAP, uno per ciascuno dei tredici scenari. Poiché più scenari possono appartenere alla stessa famiglia di botnet, il primo passo operativo consiste nell'aggregare i PCAP per famiglia tramite `mergecap`, ottenendo sette file distinti. Su ognuno di essi viene eseguito `pcapfix` per verificarne l'integrità strutturale e correggere eventuali anomalie negli header. Successivamente, ciascun PCAP verificato viene sottoposto a un filtraggio tramite `tcpdump` che seleziona esclusivamente i pacchetti con protocollo di trasporto TCP:

```
tcpdump -r input.pcap -w output_tcp.pcap tcp
```

Ciascun PCAP filtrato viene poi suddiviso negli split di training, validation e test secondo una proporzione del 70%, 15% e 15%. La suddivisione opera a livello di conversazione bidirezionale, garantendo che tutti i pacchetti appartenenti alla stessa comunicazione tra due host finiscano nello stesso split. Questa scelta previene forme di *data leakage* intra-conversazione che si verificherebbero trattando i pacchetti come unità indipendenti.

Dal punto di vista implementativo, due pacchetti vengono considerati parte della stessa conversazione se producono la stessa chiave canonica, costruita ordinando lessicograficamente la coppia di endpoint e aggiungendo il numero di protocollo:

```
key = tuple(sorted([(src_ip, src_port), (dst_ip, dst_port)]))
   + (proto,)
```

L'ordinamento garantisce che i pacchetti nelle due direzioni della stessa comunicazione producano la stessa chiave. La procedura di split si articola in quattro passi. In una prima passata sul PCAP vengono raccolte tutte le chiavi di conversazione distinte presenti nel file. Le chiavi vengono poi mescolate in modo deterministico tramite `random.seed(0)` e partizionate: il primo 70% viene assegnato al training, il 15% successivo al validation e il restante 15% al test. In una seconda passata sul PCAP, ogni pacchetto viene letto, la sua chiave viene calcolata, e il pacchetto viene scritto nel file dello split cui quella chiave è stata assegnata. Il risultato finale sono tre split in formato PCAP per ciascuna famiglia di botnet, pronti per la conversione in formato NetFlow.

Estrazione dei campioni benigni I campioni benigni non vengono estratti dai PCAP grezzi ma direttamente dai file *binetflow* distribuiti dagli autori del CTU-13, che contengono i flussi di rete già in formato tabulare. Vengono considerati benigni tutti i flussi la cui etichetta non indica traffico malevolo, comprendendo le categorie `Background` e `Normal`. Poiché i file binetflow non contengono la colonna `DstBytes` in forma esplicita, essa viene ricavata come `TotBytes - SrcBytes` per allineare la struttura di questi dati a quella dei campioni malevoli convertiti con Argus. Il risultato è un unico pool benigno condiviso tra tutte le famiglie di botnet, dal quale vengono prelevati sottoinsiemi distinti per ciascun esperimento.

Conversione PCAP in NetFlow tramite Argus Ciascuno split in formato PCAP viene convertito in formato NetFlow tabulare tramite una pipeline a due stadi basata su Argus, adottando i file di configurazione rilasciati dagli autori di [5], ma con alcune modifiche. Il primo stadio produce un file binario intermedio nel formato nativo di Argus:

```
argus -F argus.conf -r input.pcap -w output.argus
```

Il secondo stadio converte il file binario in un CSV testuale:

```
ra -F ra.conf -n -Z b -r output.argus > output.csv
```

I file di configurazione sono derivati da quelli rilasciati pubblicamente da Apruzzese et al. in [5]. Il file di configurazione per `argus` viene utilizzato senza modifiche, e imposta la generazione di flussi bidirezionali con chiave a cinque elementi, il tracciamento delle metriche di performance TCP e la produzione di dati di jitter e tempo di risposta. Il file di configurazione per `ra` viene invece modificato nel campo `RA_FIELD_SPECIFIER` per estrarre il sottoinsieme di sedici campi necessari al presente lavoro: `stime`, `dur`, `proto`, `saddr`, `sport`, `dir`, `daddr`, `dport`, `state`, `stos`, `dtos`, `pkts`, `bytes`, `sbytes`, `dbytes`, `label`. Le opzioni `-n` e `-Z b` disabilitano rispettivamente la risoluzione DNS e impostano la modalità di output dei record di stato su entrambe le direzioni del flusso.

Preprocessing e costruzione del training set baseline I CSV prodotti dalla conversione Argus vengono sottoposti a una pipeline di preprocessing che li trasforma nella rappresentazione numerica su cui opera il classificatore. Il preprocessing viene applicato sull'unione di tutti i campioni disponibili, benigni e malevoli dei tre split, trattati temporaneamente come un unico insieme, in modo che le trasformazioni dipendenti dalla distribuzione globale dei dati siano coerenti tra training, validation e test set.

Le operazioni della pipeline seguono un ordine preciso. Si parte dalle trasformazioni delle colonne categoriche, che comprendono la codifica degli indirizzi IP in flag binari interni/esterni, la conversione delle porte nei tre range IANA, l'espansione della direzione del flusso tramite one-hot encoding e la scomposizione dello stato della connessione TCP in sedici colonne binarie, otto per direzione. Queste trasformazioni sono descritte nel dettaglio nella Sezione 4.2.2. A questo punto la pipeline si biforca nelle due rappresentazioni di feature previste dal lavoro. Nella rappresentazione NO_FD si procede direttamente all'applicazione dei filtri sui valori anomali. Nella rappresentazione FD viene prima eseguito il calcolo delle quattro feature derivate, `BytesPerPkt`, `BytesPerSec`, `PktsPerSec` e `RatioOutIn`, il cui significato è illustrato nella Sezione 4.2.3, e poiché le ultime tre possono assumere valore infinito quando il denominatore è zero, i valori infiniti vengono sostituiti con il massimo valore finito della rispettiva feature calcolato sull'intero insieme, dopodiché si applicano i filtri. In entrambi i casi vengono esclusi i flussi con durata superiore a 300 secondi, con più di 100 pacchetti totali, con più di 10000 byte sorgente e con più di 60000 byte destinazione; nella rappresentazione FD si aggiungono i filtri su `BytesPerSec` superiore a 400000 e su `PktsPerSec` superiore a 10000. Questi valori soglia sono mutuati dalla configurazione adottata in DReLAB [19].

Un passaggio particolarmente rilevante della pipeline riguarda l'inserimento della colonna booleana `infect`, che identifica i flussi effettivamente perturbabili nelle fasi successive:

```
data['infect'] = data['SrcAddr'].isin(infected_ips)
data.loc[(data['infect'] == True) & (data['SrcStateP'] == 0),
   'infect'] = False
```

Un flusso riceve `infect = True` se e solo se soddisfa congiuntamente due condizioni: l'indirizzo IP sorgente appartiene alla lista degli host infetti della famiglia considerata, come documentata negli scenari CTU-13, e il flusso contiene almeno un pacchetto con flag TCP PSH in direzione sorgente. La seconda condizione è necessaria perché il packet modifier, descritto nel macro-blocco successivo, applica il padding esclusivamente ai pacchetti TCP con flag PSH: un flusso privo di tali pacchetti non verrebbe perturbato, e includerlo nel calcolo della recall darebbe una misura distorta dell'impatto reale delle perturbazioni. La colonna `infect` non partecipa all'addestramento del classificatore, ma viene propagata nei file di output e utilizzata nelle fasi di valutazione per circoscrivere il

calcolo della recall ai soli flussi perturbabili.

Al termine del preprocessing, i campioni malevoli vengono ricondotti ai rispettivi split di appartenenza. A ciascuno split vengono aggiunti campioni benigni prelevati casualmente dal pool in numero tale da mantenere un rapporto di dieci campioni benigni per ogni campione malevolo, scelta che riflette lo sbilanciamento tipico del traffico di rete reale [19]. Questo processo viene eseguito separatamente per i due percorsi di preprocessing: il training set nella rappresentazione NO_FD e il training set nella rappresentazione FD vengono costruiti e salvati indipendentemente, e su ciascuno dei due viene addestrato un modello Random Forest baseline distinto, con gli iperparametri descritti nella Sezione 4.3.

Verifica delle prestazioni del modello baseline su traffico clean. Con i modelli baseline addestrati, è possibile condurre una prima verifica delle loro prestazioni sui test set completi prodotti al termine del preprocessing, contenenti sia campioni malevoli sia campioni benigni. Questa valutazione ha una funzione di controllo preliminare: verificare che i modelli raggiungano prestazioni adeguate in condizioni ordinarie, come confermato dai risultati, prima di sottoporli a traffico perturbato. Come discusso nel Capitolo 3, un modello che non funzioni correttamente su traffico non perturbato non costituisce una baseline significativa per la valutazione successiva. I valori numerici ottenuti in questa configurazione sono riportati nel Capitolo 5.

4.4.2 Fase 2 - Generazione del traffico perturbato

Riferimento implementativo ed estensioni. Conclusa la costruzione del modello baseline, la fase successiva consiste nel generare il traffico adversariale che verrà utilizzato sia per valutare la vulnerabilità del modello clean sia per costruire il training set ampliato della fase di adversarial training. Come già anticipato nel Capitolo 3, le perturbazioni consistono nell'aggiungere ai pacchetti selezionati una sequenza di caratteri privi di qualsiasi significato semantico, il cui unico effetto è quello di alterare le statistiche del flusso osservate dal rilevatore. L'implementazione di questa operazione prende come punto di partenza il codice Python rilasciato open-source da Apruzzese et al. in [5], basato sulla libreria Scapy, e ne estende la logica su tre fronti principali.

La prima estensione riguarda il vincolo sulla dimensione massima dei pacchetti. Nel codice originale il limite di dimensione adottato è di 65.000 byte, un valore sostanzialmente ininfluente in una rete Ethernet standard. Nel presente lavoro questo valore viene sostituito con il limite MTU (*Maximum Transmission Unit*) standard di Ethernet, pari a 1.500 byte. Un pacchetto che superasse questo limite verrebbe frammentato dallo stack di rete, o scartato se il flag DF (*Don't Fragment*) è attivo, introducendo artefatti rilevabili e rendendo l'attacco facilmente identificabile. Fissare il limite a 1.500 byte è quindi una

scelta di realismo che garantisce che tutti i pacchetti perturbati siano trasmissibili senza frammentazione nell'infrastruttura di rete considerata.

La seconda estensione riguarda l'insieme delle tipologie di perturbazione considerate. Il tool originale prevede due varianti, entrambe con padding randomico in un intervallo ristretto. Il presente lavoro amplia questo insieme a 15 tipologie distinte, descritte nel paragrafo successivo, introducendo in particolare il padding fisso come categoria autonoma, assente nel riferimento originale.

La terza estensione riguarda lo scoping sul protocollo: mentre il tool originale perturba sia pacchetti UDP sia pacchetti TCP con flag PSH, il presente lavoro si concentra esclusivamente su questi ultimi, in coerenza con la composizione prevalentemente TCP del traffico botnet presente nel dataset CTU-13.

Criterio di selezione dei pacchetti e costruzione del pacchetto perturbato. Lo script itera su tutti i pacchetti del PCAP di input e applica il seguente criterio di selezione: vengono perturbati esclusivamente i pacchetti TCP che presentano il flag PSH attivo; tutti gli altri pacchetti, siano essi UDP, pacchetti TCP di controllo privi di flag PSH, pacchetti non-IP o pacchetti con errori di parsing, vengono scritti nel PCAP di output senza alcuna modifica.

Il criterio combinato `TCP` $\wedge$ `PSH` garantisce che il padding venga aggiunto solo ai pacchetti che trasportano payload applicativo, preservando la semantica del protocollo TCP: intervenire sui pacchetti di controllo della connessione altererebbe il comportamento del protocollo e potrebbe compromettere il funzionamento della comunicazione malevola. Per ciascun pacchetto selezionato, la costruzione del pacchetto perturbato segue una sequenza precisa. Il padding viene generato tramite la funzione `randStr`, che estrae casualmente caratteri da un alfabeto alfanumerico ASCII composto da lettere maiuscole e cifre. La lunghezza del padding non è il valore nominale della tipologia, ma viene calcolata dinamicamente tenendo conto dello spazio residuo del pacchetto rispetto al limite MTU:

```
def randStr(chars = string.ascii_uppercase + string.digits,
   lower=0, upper=100):
    if upper < 0:
        upper = lower = 0
    if upper < lower:
        lower = upper
    padding = random.randint(lower, upper)
    return ''.join(random.choice(chars) for _ in range(padding
       ))
```

I due controlli iniziali gestiscono i casi limite: se il pacchetto ha già raggiunto o superato il limite MTU, nessun byte viene aggiunto; se lo spazio residuo è inferiore al

valore minimo dell'intervallo, il padding viene ridotto allo spazio disponibile. In questo modo si garantisce che nessun pacchetto perturbato superi i 1.500 byte.

La funzione viene chiamata passando come limite superiore il minimo tra il valore nominale della tipologia e lo spazio residuo calcolato sulla dimensione corrente del pacchetto:

```
padding = randStr(lower=lower, upper=min((max_size - pkt.len),
    upper))
```

Il padding così generato viene aggiunto al pacchetto tramite le primitive di Scapy, i campi di lunghezza e checksum degli header IP e TCP vengono cancellati per forzarne il ricalcolo automatico al momento della serializzazione, e il timestamp originale viene preservato per mantenere la coerenza temporale del PCAP ai fini della successiva conversione con Argus. Il risultato è un pacchetto TCP valido a tutti gli effetti, con header consistenti e dimensione compatibile con i vincoli dell'infrastruttura di rete.

Le tipologie di padding implementate. Le 15 tipologie di padding implementate si articolano in due categorie che differiscono per il modo in cui viene determinata la quantità di byte da aggiungere a ciascun pacchetto. Questa distinzione non è solo tecnica, ma riflette due approcci di attacco differenti, con implicazioni diverse sull'effetto che le perturbazioni producono sulle statistiche del flusso.

La prima categoria è il *padding fisso*, in cui ogni pacchetto perturbabile riceve un numero costante di byte aggiuntivi, indipendentemente dalle sue dimensioni originali. Sono state considerate 11 istanze di questa categoria, corrispondenti ai valori 1, 8, 16, 32, 64, 96, 128, 256, 352, 512 e 1024 byte. La progressione copre un ampio spettro di intensità di modifica: dai singoli byte, che producono una variazione pressoché impercettibile sulle statistiche aggregate del flusso, fino al kilobyte, che introduce uno scostamento considerevole in feature come `TotBytes`, `SrcBytes` e nelle grandezze derivate della rappresentazione FD. Questa categoria modella uno scenario in cui l'attaccante adotta una strategia uniforme e deterministica, replicando sempre la stessa modifica su ogni pacchetto. Dal punto di vista delle statistiche del flusso, il padding fisso produce un effetto sistematico e prevedibile: poiché tutti i pacchetti perturbabili ricevono la stessa quantità di byte aggiuntivi, l'incremento sulle feature volumetriche del flusso è proporzionale al numero di pacchetti PSH presenti nel flusso, e la distribuzione delle dimensioni dei pacchetti si sposta rigidamente verso l'alto di un valore costante.

La seconda categoria è il *padding randomico*, in cui il numero di byte aggiunto a ciascun pacchetto viene estratto uniformemente all'interno di un intervallo predefinito. Questa categoria introduce una componente stocastica che differenzia il comportamento da pacchetto a pacchetto all'interno dello stesso flusso, modellando uno scenario in cui l'attaccante varia continuamente la propria strategia per rendere il traffico perturbato

meno riconoscibile per pattern. Sono stati considerati tre intervalli: `[1, 100]`, che si allinea all'intervallo adottato nel riferimento originale [5] e produce perturbazioni di entità contenuta; `[100, 1024]`, che esplora la fascia di perturbazioni medio-grandi; e `[1, 1500]`, che copre l'intero spazio di variazione ammissibile fino al limite MTU, consentendo perturbazioni che spaziano dal singolo byte alla dimensione massima del pacchetto. A differenza del padding fisso, l'effetto del padding randomico sulle statistiche del flusso non è deterministico ma dipende dai valori casuali estratti: flussi diversi perturbati con la stessa tipologia randomica possono presentare variazioni nelle feature volumetriche molto diverse tra loro, il che rende questa categoria più difficile da anticipare per il rilevatore, ma anche più difficile da studiare in modo riproducibile, come discusso nel paragrafo successivo.

Seeding e run multipli per le tipologie randomiche. La natura stocastica del padding randomico introduce una dipendenza dei risultati sperimentali dai valori casuali estratti durante la generazione. Per gestire questa variabilità in modo controllato, ogni esecuzione dello script imposta un seed deterministico per il generatore di numeri casuali, garantendo la riproducibilità di ciascuna istanza di perturbazione. Per le tipologie di padding fisso, dove la perturbazione è completamente deterministica, un singolo run è sufficiente. Per le tipologie randomiche, un singolo run potrebbe non essere rappresentativo del comportamento medio: la recall misurata su un singolo PCAP perturbato dipende dai valori specifici estratti, e una sola esecuzione potrebbe sovrastimare o sottostimare l'effetto reale della perturbazione. Per questa ragione, per ciascuna delle tre tipologie randomiche vengono eseguiti 10 run indipendenti, ciascuno con un seed diverso, producendo 10 PCAP di test set distinti per ogni tipologia. La recall riportata nelle tabelle di valutazione è la media dei valori ottenuti sui 10 run, fornendo una stima statisticamente più robusta dell'effetto della perturbazione.

I PCAP perturbati prodotti in questa fase vengono infine convertiti in NetFlow tramite la stessa pipeline Argus descritta nel paragrafo 4.4.1, producendo i file CSV che alimenteranno sia la valutazione della vulnerabilità del modello clean sia la fase di adversarial training.

Valutazione della vulnerabilità del modello baseline su traffico perturbato. Con i NetFlow perturbati disponibili, è possibile realizzare il primo dei due impieghi anticipati nell'apertura di questa fase: la valutazione della vulnerabilità del modello baseline su traffico adversariale. La recall viene calcolata sui test set perturbati, circoscrivendo il calcolo ai soli flussi con `infect = True` secondo il criterio introdotto nel paragrafo 4.4.1, e ripetendo la misurazione per ciascuna combinazione di famiglia di botnet e tipologia di perturbazione. I risultati mostrano un calo della recall rispetto al caso non perturbato, confermando che il modello baseline non è in grado di classificare correttamente il traffico

malevolo una volta perturbato. È a partire da questa osservazione, già anticipata sul piano metodologico nel Capitolo 3, che si giustifica il passaggio alla Fase 3: se il modello baseline mostrasse già una robustezza naturale alle perturbazioni in *traffic space*, non vi sarebbe ragione di intervenire sulla procedura di addestramento. I valori numerici di questa valutazione sono riportati nel Capitolo 5.

4.4.3 Fase 3 - Adversarial training e costruzione dei modelli hardened

Costruzione del training set per i modelli hardened. Con i PCAP perturbati convertiti in NetFlow, sono disponibili tutti gli elementi necessari per costruire i modelli hardened. La logica che guida questa fase è concettualmente semplice: si vuole che il modello, durante l'addestramento, osservi non solo traffico malevolo nella sua forma originale, ma anche traffico malevolo nella forma in cui si presenterebbe dopo essere stato perturbato da un attaccante. Per ottenere questo risultato, il training set viene ricostruito da zero a partire da tre sorgenti distinte, combinate e preprocessate seguendo le stesse modalità della fase 1.

La prima sorgente è il traffico malevolo clean del training set, nella forma grezza prodotta dalla conversione Argus nella fase 1, prima di qualsiasi preprocessing. La seconda sorgente è il validation set perturbato, anch'esso nella forma grezza prodotta dalla riconversione Argus dei PCAP perturbati: si tratta quindi del traffico malevolo del validation set dopo che le perturbazioni di padding sono state applicate a livello di pacchetto. La terza sorgente è il pool di campioni benigni estratto dai binetflow del CTU-13 nella fase 1. È importante sottolineare che tutte e tre le sorgenti si trovano allo stato grezzo nel momento in cui vengono unite: questa scelta non è casuale, ma risponde alla stessa logica adottata nella fase baseline, per cui il preprocessing viene applicato sull'unione di tutti i campioni disponibili trattandoli come un unico insieme, garantendo che le trasformazioni dipendenti dalla distribuzione globale dei dati siano coerenti sull'intero training set.

Una volta unite le tre sorgenti, si applica la stessa pipeline di preprocessing descritta nella Sezione 4.4.1, che produce le due rappresentazioni di feature previste dal lavoro. Al termine del preprocessing, il numero di campioni benigni da includere nel training set finale viene calcolato applicando lo stesso rapporto di dieci campioni benigni per ogni campione malevolo adottato nella fase 1. Poiché la componente malevola comprende ora sia i campioni clean di training sia i campioni perturbati del validation set, il totale dei malevoli è maggiore rispetto alla fase 1, e di conseguenza anche il numero di benigni ricampionati dal pool è maggiore. Il training set risultante contiene quindi tre categorie di campioni, traffico malevolo non perturbato, traffico malevolo perturbato e traffico benigno, nella proporzione stabilita, ed è strutturalmente identico per metodologia a quello della fase 1, ma più ampio e più vario nella sua componente malevola.

Selezione delle famiglie e addestramento dei modelli hardened Una scelta implementativa da evidenziare è che la valutazione completa viene condotta esclusivamente per le famiglie di botnet per cui il numero di flussi con `infect = True` presenti nel test set supera una soglia minima. Questa condizione è direttamente collegata al criterio di perturbabilità definito nella Sezione 4.4.1: la colonna `infect`, come descritto nella Sezione 4.4.1, identifica i flussi che contengono almeno un pacchetto TCP con flag PSH proveniente dall'host infetto, e quindi i flussi effettivamente perturbabili. La soglia adottata è di 100 campioni perturbabili per famiglia: al di sotto di questo valore, calcolare la recall su un numero così ridotto di campioni non produce una stima affidabile della capacità di rilevamento del modello, indipendentemente dal confronto con il modello baseline. Applicando questo criterio, delle sette famiglie originariamente presenti nel CTU-13, tre soddisfano la condizione e vengono mantenute nell'analisi, ossia Murlo, Neris e Virut, mentre le restanti quattro, Rbot, NSIS, Menti e Sogou, vengono escluse. La scelta della soglia di 100 è empirica e rappresenta un compromesso tra ampiezza del campione e copertura delle famiglie disponibili.

Il modello hardened viene addestrato sul training set ricostruito utilizzando gli stessi iperparametri del modello baseline. Questa scelta è deliberata: mantenere invariata la configurazione del classificatore garantisce che la composizione del training set sia l'unica variabile che differenzia il modello hardened dal modello baseline, rendendo il confronto tra i due diretto e interpretabile, senza che differenze nelle prestazioni possano essere attribuite a variazioni nella struttura del classificatore.

Un aspetto rilevante della strategia implementata riguarda la granularità con cui i modelli hardened vengono costruiti. Non viene addestrato un unico modello hardened generico, capace di resistere a tutte le tipologie di perturbazione simultaneamente, ma un modello distinto per ciascuna combinazione di famiglia di botnet, rappresentazione di feature e tipologia di padding considerata nel lavoro. Ciascun modello hardened è quindi specializzato sulla specifica tipologia di perturbazione con cui il suo training set è stato costruito, e per ciascuna delle tre famiglie mantenute esiste un modello baseline e un modello hardened per ogni tipologia di perturbazione investigata. Questa granularità è funzionale a una delle domande centrali del lavoro: se un modello addestrato contro una specifica tipologia di perturbazione acquisisca robustezza anche nei confronti di perturbazioni diverse da quelle viste durante l'addestramento, o se la sua resistenza sia circoscritta alla variante specifica su cui è stato hardened. La risposta a questa domanda, che emerge dall'analisi dei risultati presentati nel Capitolo 5, ha ricadute dirette sulla praticabilità dell'adversarial training come strategia difensiva in scenari operativi reali.

Capitolo 5

Valutazione e Risultati

Il Capitolo 4 ha documentato la messa in pratica del framework proposto: la costruzione del dataset di lavoro, la scelta del classificatore e dei suoi iperparametri, e l'implementazione delle perturbazioni in *traffic space* che costituiscono il nucleo dell'approccio difensivo. Ha inoltre evidenziato come i modelli hardened siano stati costruiti separatamente per ciascuna tipologia di perturbazione, aprendo la questione se tale specializzazione produca robustezza generalizzata o limitata alla sola variante di addestramento.

Il presente capitolo raccoglie i risultati della sperimentazione, con l'obiettivo di verificare se l'adversarial training in *traffic space* produca modelli al contempo robusti verso il traffico adversarial e stabili su traffico pulito. L'asse portante della valutazione è il confronto sistematico tra modelli addestrati su dati clean[1] e modelli addestrati tramite adversarial training, condotto su entrambe le rappresentazioni di feature adottate nel lavoro.

Research Questions La valutazione sperimentale è costruita attorno a due domande di ricerca complementari, che orientano la struttura dell'intero capitolo.

La prima domanda riguarda la **vulnerabilità del sistema di rilevamento**: un modello addestrato su traffico non perturbato subisce una degradazione significativa delle proprie prestazioni quando il traffico malevolo che gli viene sottoposto è stato modificato tramite perturbazioni in *traffic space*? Questa domanda è preliminare rispetto a qualunque considerazione difensiva: se le perturbazioni considerate non producessero alcun effetto apprezzabile sulla capacità di rilevamento, non vi sarebbe alcuna motivazione per introdurre una strategia di hardening.

La seconda domanda riguarda l'**efficacia della difesa proposta**: un modello sottoposto ad adversarial training recupera robustezza nei confronti del traffico perturbato, e lo fa senza degradare le proprie prestazioni sul traffico ordinario non perturbato? Quest'ultimo aspetto è altrettanto critico: una strategia difensiva che aumentasse la robustezza verso il

[1] Nel seguito, i termini *modello clean* e *modello baseline* sono usati come sinonimi per indicare il modello addestrato esclusivamente su traffico non perturbato, prima dell'adversarial training.

traffico adversarial a scapito di una significativa riduzione della capacità di rilevamento del traffico malevolo pulito non sarebbe accettabile in un sistema operativo reale, poiché il traffico perturbato rappresenta solo una frazione del traffico malevolo che un rilevatore incontra nella pratica.

Le due domande sono interdipendenti e richiedono una struttura di valutazione che le affronti in modo organizzato. A tal fine, la sperimentazione è articolata in due scenari distinti, ciascuno progettato per rispondere a una porzione specifica del quadro valutativo complessivo, descritti nella Sezione 5.1.

Organizzazione del capitolo Il capitolo è organizzato in quattro sezioni. La Sezione 5.1 definisce gli scenari di valutazione adottati e le tipologie di perturbazione considerate nei test. La Sezione 5.2 descrive le metriche impiegate e motiva le scelte operate nella loro selezione. La Sezione 5.3 presenta i risultati del modello baseline su traffico non perturbato e su traffico perturbato, fornendo la risposta empirica alla prima domanda di ricerca. La Sezione 5.4 riporta i risultati del modello hardened su traffico non perturbato e su traffico perturbato, documentando le proprietà di non-regressione e di generalizzazione dell'approccio proposto, e rispondendo alla seconda domanda di ricerca.

5.1 Scenari di valutazione

Le due domande di ricerca enunciate nell'introduzione del capitolo sono interdipendenti: la verifica dell'efficacia della difesa ha senso solo se la vulnerabilità preesistente del modello baseline è stata empiricamente stabilita. Per affrontarle in modo organizzato, la sperimentazione è articolata in due scenari distinti, ciascuno progettato per rispondere a una porzione specifica del quadro valutativo complessivo.

5.1.1 Scenario 1 - Valutazione su traffico non perturbato

Prima di procedere con la valutazione adversarial vera e propria, è necessario accertare che i modelli di partenza siano effettivamente in grado di svolgere il proprio compito in condizioni operative ordinarie. A tal fine, in questo primo scenario, ciascun modello clean viene sottoposto a una valutazione su un test set composto sia da sample malevoli sia da sample benigni, nella proporzione di dieci sample benigni per ogni sample malevolo, coerentemente con il rapporto adottato in fase di addestramento e descritto nella Sezione 4.2.

Questa verifica coinvolge tutte e sette le famiglie di botnet presenti nel dataset CTU-13, ossia Menti, Murlo, Neris, NSIS, Rbot, Sogou e Virut. Tuttavia, è necessario fare un'anticipazione: non tutte le famiglie si prestano allo stesso modo alla valutazione adversarial che costituisce il nucleo del presente lavoro. Come discusso nella Sezione 4.2,

le perturbazioni in *traffic space* sono applicabili esclusivamente ai pacchetti TCP con flag PSH provenienti dagli host infetti. L'analisi dei file PCAP ha mostrato che quattro delle sette famiglie del dataset — Menti, NSIS, Rbot e Sogou — presentano un numero di pacchetti perturbabili inferiore a 100 nel test set: una soglia al di sotto della quale il numero di flussi malevoli effettivamente modificati dalla perturbazione è troppo esiguo per sostenere una valutazione statisticamente significativa della recall.

Questa soglia costituisce un *criterio di selezione* esplicito: famiglie con meno di 100 campioni perturbabili nel test set vengono escluse dalla valutazione adversarial, sia perché la stima della recall su campioni così rari è altamente instabile, sia perché una tale scarsità di traffico perturbabile ridurrebbe significativamente il numero di flussi su cui addestrare i modelli hardened, compromettendo la validità del confronto. Per questa ragione, la valutazione adversarial — sia la verifica di vulnerabilità del modello clean sia la valutazione dei modelli hardened — è circoscritta alle tre famiglie che soddisfano il criterio: Murlo, Neris e Virut.[2]

È importante sottolineare che la condizione di separazione netta tra i dati di addestramento e quelli di valutazione è il presupposto che conferisce significato alla misurazione: ciò che si quantifica in questo scenario non è la capacità del modello di riprodurre ciò che ha appreso, ma la sua capacità di generalizzare a traffico reale mai incontrato durante la costruzione del classificatore. Senza una stima affidabile delle prestazioni su traffico ordinario, non sarebbe possibile distinguere, negli scenari successivi, quanto della variazione osservata sia attribuibile all'effetto delle perturbazioni e quanto invece a una debolezza preesistente del modello. I risultati di questo scenario sono presentati nella Sezione 5.3.

5.1.2 Scenario 2 - Valutazione su traffico perturbato

Nel secondo scenario, i modelli clean e i modelli hardened vengono valutati su test set perturbati, nei quali il traffico malevolo è stato modificato tramite l'aggiunta di padding ai pacchetti TCP con flag PSH provenienti dagli host infetti. Le perturbazioni utilizzate nei test appartengono alle due categorie implementate nel Capitolo 4: il padding fisso, in cui ciascun pacchetto perturbabile riceve un numero costante di byte aggiuntivi, e il padding randomico, in cui la quantità di byte viene estratta uniformemente all'interno di un intervallo predefinito. Complessivamente, le tipologie di perturbazione considerate nella valutazione sono quattordici: undici varianti di padding fisso, corrispondenti a valori crescenti di byte aggiunti, e tre varianti di padding randomico, corrispondenti a intervalli di ampiezza crescente. I dettagli implementativi di entrambe le categorie, incluse le specifiche tipologie considerate e il vincolo MTU applicato alla costruzione dei pacchetti perturbati, sono stati descritti nella Sezione 4.4.

[2]Le quattro famiglie escluse (Menti, NSIS, Rbot, Sogou) vengono comunque incluse nella valutazione su traffico non perturbato (Scenario 1), per la quale la dimensione del test set clean è adeguata in tutti i casi.

Questo scenario è strutturato in tre confronti distinti. Il primo valuta i modelli clean su traffico perturbato, quantificando la vulnerabilità di un classificatore addestrato su dati ordinari quando esposto a perturbazioni in *traffic space*. Il secondo valuta i modelli hardened sugli stessi test set non perturbati dello Scenario 1, verificando che il processo di hardening non abbia compromesso le prestazioni su traffico ordinario. Il terzo valuta i modelli hardened su traffico perturbato, misurando la robustezza acquisita tramite adversarial training e confrontandola direttamente con le prestazioni dei modelli clean negli stessi test set. I risultati di questo scenario sono presentati nelle Sezioni 5.3 e 5.4.

5.2 Metriche di valutazione

La scelta delle metriche con cui misurare le prestazioni di un classificatore non è una decisione neutra. In un contesto come quello dei sistemi di rilevamento delle intrusioni di rete, dove classificare erroneamente un flusso come benigno o come malevolo porta a conseguenze operative molto diverse, e dove il traffico benigno supera strutturalmente quello malevolo di diversi ordini di grandezza [7], la scelta della metrica determina quali aspetti del comportamento del modello vengono effettivamente quantificati dalla valutazione. Prima di presentare i risultati delle sezioni successive, è quindi necessario introdurre le metriche considerate, illustrarne le proprietà e motivare le scelte operate in ciascuno dei due scenari di valutazione descritti nella Sezione 5.1.

5.2.1 Definizione delle metriche

Per valutare le prestazioni di un classificatore binario, è utile distinguere i possibili esiti della classificazione in quattro categorie fondamentali.

Dato un classificatore che distingue tra campioni malevoli (classe positiva) e campioni benigni (classe negativa), un campione sottoposto al classificatore può produrre uno dei seguenti risultati: un campione malevolo classificato correttamente come tale costituisce un *vero positivo* (TP, *True Positive*); un campione benigno classificato erroneamente come malevolo è un *falso positivo* (FP, *False Positive*); un campione malevolo classificato erroneamente come benigno è un *falso negativo* (FN, *False Negative*); infine, un campione benigno classificato correttamente come tale è un *vero negativo* (TN, *True Negative*).

Da queste quattro quantità si derivano le metriche aggregate utilizzate in letteratura per sintetizzare il comportamento del classificatore. La prima, e centrale per questo lavoro, è la **recall**, anche denominata *True Positive Rate* (TPR) o *tasso di rilevamento*:

$$\text{Recall} = \text{TPR} = \frac{TP}{TP + FN} \tag{5.1}$$

Essa misura la proporzione di campioni malevoli effettivamente rilevati sul totale dei campioni malevoli presenti nel test set. Un valore di recall pari a 1 indica che nessun attacco è sfuggito al rilevamento; un valore pari a 0 indica che il classificatore ha mancato sistematicamente tutti gli attacchi.

La **precision** misura la proporzione di campioni classificati come malevoli che sono effettivamente malevoli:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

Essa quantifica quante delle segnalazioni prodotte dal classificatore corrispondono a minacce reali: una precision bassa si traduce in un elevato numero di falsi allarmi, mentre una precision alta indica che le segnalazioni prodotte sono prevalentemente corrette.

L'**F1-score** è la media armonica di precision e recall, e nasce dall'esigenza di sintetizzare in un unico valore scalare due grandezze che descrivono aspetti complementari del comportamento del classificatore:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

La media armonica penalizza fortemente gli squilibri tra i due termini: un classificatore con precision molto alta e recall molto bassa otterrà un F1-score vicino al valore del termine più basso, non alla loro media. Questa proprietà lo rende più informativo dell'accuracy in presenza di classi sbilanciate.

La scelta di quale tra queste metriche adottare come riferimento primario non è però una questione puramente tecnica, ma dipende dal contesto applicativo e dalle priorità operative del sistema che si vuole valutare.

In ambito IDS, classificare un flusso malevolo come benigno e classificare un flusso benigno come malevolo non sono errori equivalenti dal punto di vista delle loro conseguenze operative.

Un falso positivo – un flusso benigno segnalato erroneamente come malevolo – genera un allarme non fondato che può essere gestito dagli analisti di sicurezza con un costo operativo aggiuntivo, ma non lascia traccia di un attacco nell'infrastruttura. Un falso negativo, al contrario, corrisponde a un attacco reale che il sistema di rilevamento non ha identificato: l'attaccante ha agito indisturbato, e il danno si è consumato prima che alcuna risposta fosse possibile. La minimizzazione dei falsi negativi è quindi la priorità strutturale di qualsiasi sistema di rilevamento delle intrusioni [8].

Per questa ragione, la recall è adottata come metrica primaria in tutto il presente lavoro. È la grandezza su cui si confrontano i modelli clean e i modelli hardened, ed è l'unica metrica riportata nella valutazione su traffico perturbato.

Questa scelta è ulteriormente motivata dalla composizione dei test set adottati nella fase di valutazione su traffico perturbato, descritta nella Sezione 5.3, che prevede test set composti esclusivamente da campioni malevoli. Questa composizione è conseguenza

diretta dell'approccio adottato per la generazione del traffico perturbato, descritto nella Sezione 4.4: le perturbazioni vengono applicate a livello di pacchetto nei file PCAP malevoli, producendo test set che non includono traffico benigno.

In queste condizioni, la presenza di falsi positivi è strutturalmente impossibile: non esistono campioni benigni che possano essere erroneamente classificati come malevoli, quindi $FP = 0$ e $TN = 0$ per costruzione. Le conseguenze sul piano delle metriche sono immediate. La precision risulta pertanto pari a 1 in ogni configurazione, indipendentemente dal comportamento reale del classificatore, e non porta alcuna informazione discriminante. L'F1-score, essendo funzione della precision, collassa a una riformulazione della sola recall, senza aggiungere contenuto informativo. L'accuracy, infine, coincide anch'essa con la recall quando non esistono veri negativi, perdendo il suo ruolo di misura aggregata del comportamento complessivo del classificatore.

In questo scenario, la recall è l'unica metrica che misura effettivamente ciò che interessa valutare: la proporzione di campioni malevoli perturbati che il classificatore riesce ancora a rilevare nonostante la perturbazione. Escludere F1 e accuracy non è un'omissione, ma una scelta metodologica coerente con il setup sperimentale adottato.

5.3 Risultati del modello *Random Forest* baseline su traffico non perturbato e perturbato

I capitoli precedenti hanno descritto il framework sperimentale nel suo insieme: il dataset, le rappresentazioni di feature, il classificatore, le tipologie di perturbazione e la struttura della valutazione. Si presentano ora i risultati numerici, a partire dallo Scenario 1, in cui i modelli clean vengono valutati su traffico non perturbato.

L'obiettivo di questa sezione è stabilire la baseline di prestazioni a partire dalla quale tutti i confronti successivi vengono interpretati. Un modello che non funzioni correttamente in condizioni ordinarie non costituisce un punto di partenza valido per nessuna analisi di robustezza: se la recall di partenza fosse già inadeguata, non sarebbe possibile distinguere, nelle sezioni successive, se il calo osservato sul traffico perturbato sia conseguenza delle perturbazioni stesse o di una debolezza strutturale preesistente nel modello. La solidità della baseline è quindi un risultato a sé, non un presupposto secondario dell'analisi.

La metrica riportata è la recall, per le ragioni illustrate nella Sezione 5.2. Il test set utilizzato in questo scenario contiene sia campioni malevoli sia campioni benigni, nella proporzione 10:1 adottata durante l'addestramento e descritta nella Sezione 4.2, e nessuno di questi campioni ha partecipato alla costruzione del modello.

5.3.1 Prestazioni del modello *Random Forest* baseline su traffico non perturbato

Le Tabelle di seguito riportano i valori ottenuti dai sette modelli clean – uno per ciascuna famiglia di botnet – valutati su entrambe le rappresentazioni di feature adottate nel lavoro (NO_FD e FD), per un totale di quattordici combinazioni famiglia-rappresentazione.

Tabella 5.1: Precision, recall e F1-score dei modelli clean su test set non perturbato, rappresentazione NO_FD (base). Le metriche si riferiscono ai sample malevoli.

Famiglia	**Precision**	**Recall**	**F1-score**
Menti	1.00	0.9062	0.95
Murlo	1.00	0.9659	0.98
Neris	0.98	0.9036	0.94
NSIS	0.91	0.8936	0.90
Rbot	1.00	0.9375	0.97
Sogou	1.00	1.0000	1.00
Virut	1.00	0.9393	0.97

Tabella 5.2: Precision, recall e F1-score dei modelli clean su test set non perturbato, rappresentazione FD (estesa). Le metriche si riferiscono ai sample malevoli.

Famiglia	**Precision**	**Recall**	**F1-score**
Menti	1.00	0.9375	0.97
Murlo	1.00	0.9636	0.98
Neris	0.98	0.9050	0.94
NSIS	0.93	0.8936	0.91
Rbot	1.00	0.9377	0.98
Sogou	1.00	0.9991	1.00
Virut	1.00	0.9344	0.97

La quasi totalità dei modelli raggiunge valori di recall prossimi o superiori a 0.90, confermando che i modelli di partenza sono effettivamente in grado di riconoscere traffico malevolo in condizioni operative ordinarie, e che quindi la baseline è sufficientemente solida.

La lettura delle tabelle mette in evidenza due aspetti che meritano attenzione.

Il primo riguarda il profilo prestazionale complessivo. La grande maggioranza delle famiglie raggiunge valori di precision, recall e F1-score prossimi o uguali a 1.00 su entrambe le rappresentazioni. Fanno eccezione Neris e NSIS, che presentano recall più

contenute: Neris si attesta attorno a 0.90 in entrambe le rappresentazioni, mentre NSIS raggiunge 0.89 in entrambi i casi. In tutte le altre famiglie il classificatore identifica correttamente la quasi totalità dei flussi malevoli, con al più un numero trascurabile di falsi negativi. I valori di precision sono in tutti i casi pari o prossimi a 1.00, indicando un tasso di falsi positivi sostanzialmente nullo: il classificatore non confonde traffico benigno con traffico malevolo. Questa differenza tra famiglie non è necessariamente un difetto del modello: può riflettere caratteristiche intrinseche del traffico di ciascuna botnet, come la variabilità interna dei flussi o la sovrapposizione con le distribuzioni del traffico benigno. Ciò che importa rilevare in questa sede è che le varie famiglie non partono dalla stessa posizione: ad esempio, una degradazione della recall di uguale entità assoluta pesa proporzionalmente di più su Neris, che parte da un valore più basso, rispetto a Murlo o Virut. Questa asimmetria va considerata nella lettura dei risultati che seguono, dove i valori di recall vanno sempre interpretati in relazione alla baseline di partenza della famiglia corrispondente.

Il secondo aspetto riguarda il confronto tra le due rappresentazioni di feature. In tutti i casi, la differenza di recall tra la rappresentazione base e quella estesa è rascurabile, inferiore a un punto percentuale per tutte le famiglie. Le feature derivate aggiunte dalla rappresentazione FD — `BytesPerPkt`, `BytesPerSec`, `PktsPerSec` e `RatioOutIn` — non apportano alcun miglioramento misurabile alla capacità discriminativa del modello su traffico non perturbato. Su traffico pulito, il segnale discriminante tra flussi malevoli e benigni è già interamente catturato dalle feature di base, e le quattro grandezze derivate non apportano informazione aggiuntiva. Questo è un primo dato di confronto tra le due rappresentazioni; la domanda se questa equivalenza si mantenga anche quando i flussi malevoli vengono perturbati trova risposta nella parte che segue.

Alla luce di quanto osservato, i risultati presentati in questa sezione rispondono all'obiettivo preliminare definito nella Sezione 5.1: capire se i modelli addestrati su traffico pulito siano in grado di svolgere il proprio compito in condizioni ordinarie. La risposta è affermativa per tutte e sette le famiglie e per entrambe le rappresentazioni di feature, senza eccezioni.

Vale la pena sottolineare un aspetto che verrà ripreso nelle conclusioni del capitolo. Il fatto che le due rappresentazioni producano prestazioni pressoché identiche su traffico non perturbato non significa che siano equivalenti in tutti i contesti: significa soltanto che, in assenza di perturbazioni, la distinzione tra flussi malevoli e benigni non richiede le informazioni aggiuntive catturate dalle feature derivate. Se e quanto questa equivalenza si mantenga sotto pressione adversarial è una questione empiricamente aperta, che i risultati presentati nel seguito permetteranno di affrontare con dati concreti.

Con la baseline stabilita, l'analisi si sposta ora sulla fase adversarial: i modelli clean appena descritti verranno sottoposti a traffico perturbato, per quantificare la loro vulnerabilità alle perturbazioni in *traffic space*.

5.3.2 Vulnerabilità del modello *Random Forest* baseline su traffico perturbato

Per ciascuna delle tre famiglie di botnet e per entrambe le rappresentazioni di feature, i modelli clean vengono valutati su tutti e quattordici i test set perturbati costruiti nel Capitolo 4: undici varianti di padding fisso, con dimensioni crescenti da 1 a 1024 byte, e tre varianti di padding randomico, con intervalli di ampiezza crescente.

La Tabella 5.3 riporta, per ciascuna combinazione di famiglia e rappresentazione, la recall del modello clean sul test set non perturbato e su ciascuno dei quattordici test set perturbati.

Tabella 5.3: Recall dei modelli clean sul test set non perturbato (riga *Clean*) e sui quattordici test set perturbati, per famiglia di botnet e rappresentazione di feature.

Perturbazione	**Murlo**		**Neris**		**Virut**	
	NO_FD	FD	NO_FD	FD	NO_FD	FD
Clean	0.9659	0.9636	0.9036	0.9050	0.9393	0.9344
1 B fisso	0.9470	0.9637	0.9012	0.9035	0.9369	0.9344
8 B fisso	0.7614	0.9194	0.7493	0.8388	0.8995	0.9016
16 B fisso	0.7462	0.8669	0.6766	0.7619	0.8009	0.8685
32 B fisso	0.7273	0.8065	0.5951	0.6116	0.7418	0.7647
64 B fisso	0.5530	0.7742	0.5019	0.5047	0.5366	0.6508
96 B fisso	0.5341	0.5887	0.4855	0.4718	0.4105	0.4688
128 B fisso	0.5265	0.5691	0.4410	0.4364	0.3885	0.4324
256 B fisso	0.0000	0.0000	0.3778	0.3508	0.2921	0.2626
352 B fisso	0.0000	0.0000	0.3610	0.3356	0.3048	0.2022
512 B fisso	0.0000	0.0000	0.3693	0.3191	0.1962	0.1011
1024 B fisso	0.0000	0.0000	0.3073	0.0865	0.1023	0.0399
Rand. $[1, 100]$	0.6439	0.7419	0.5565	0.5418	0.6494	0.6934
Rand. $[100, 1024]$	0.0079	0.0000	0.3407	0.2417	0.1826	0.1011
Rand. $[1, 1500]$	0.0433	0.0000	0.3447	0.2344	0.1935	0.1093

I dati mostrano una vulnerabilità netta e consistente in tutte le configurazioni considerate. La recall media sul traffico perturbato si colloca tra 0.3922 e 0.5006, contro valori di baseline compresi tra 0.9036 e 0.9659 su traffico non perturbato: la degradazione è quindi compresa tra circa 45 e 55 punti percentuali, a seconda della famiglia e della rappresentazione di feature. Tradotto in termini operativi, un modello che in condizioni ordinarie riconosce correttamente tra il 90% e il 97% dei flussi malevoli, una volta esposto a traffico perturbato con padding in *traffic space*, arriva a identificarne mediamente meno della metà.

Un aspetto che merita attenzione è la diversa entità del crollo tra le tre famiglie. Per Murlo, la recall minima tocca il valore di 0.000 in entrambe le rappresentazioni: il padding

fisso da 256 byte è sufficiente a far sì che il modello non rilevi alcun campione malevolo nel test set, di fatto azzerando la capacità di rilevamento. Questa è una vulnerabilità particolarmente marcata, tanto più sorprendente se si considera che Murlo è la famiglia con la baseline più alta tra le tre (0.966 su traffico non perturbato). Per Neris e Virut, il minimo non raggiunge lo zero, ma rimane comunque su valori criticamente bassi: 0.3073 e 0.0865 per Neris (NO_FD e FD rispettivamente), 0.1023 e 0.0399 per Virut. Per Neris e Virut, il valore minimo si osserva in corrispondenza del padding fisso da 1024 byte, il più grande tra le undici varianti considerate; per Murlo, la recall si azzera già a partire dal padding da 256 byte, e rimane pari a zero anche per tutte le varianti di dimensione superiore.

I dati suggeriscono una tendenza generale per cui le varianti di padding più piccolo producono una degradazione più contenuta, mentre le varianti di padding più grandi producono le degradazioni più severe, anche se questa relazione non è strettamente monotona in tutte le famiglie e rappresentazioni.

Questa tendenza è comprensibile se si considera il modo in cui i flussi perturbati vengono rappresentati dopo la conversione in formato NetFlow. Il classificatore non osserva i singoli pacchetti, ma una serie di grandezze aggregate calcolate sull'intero flusso, tra cui il numero totale di byte scambiati, la dimensione media per pacchetto e il rapporto tra byte nelle due direzioni della comunicazione. Aggiungere pochi byte per pacchetto altera queste aggregazioni in modo marginale, e il profilo del flusso perturbato rimane sufficientemente simile a quello visto durante il training da non compromettere la classificazione.

Al contrario, aggiungere centinaia di byte per pacchetto modifica in modo sostanziale le stesse grandezze, portando il flusso perturbato in una regione dello spazio delle feature che il modello non ha mai osservato in fase di addestramento, e per la quale non ha sviluppato capacità discriminativa. Non si tratta di un limite delle feature scelte: le grandezze aggregate del NetFlow descrivono efficacemente il traffico in condizioni ordinarie, come confermato dai risultati presentati nella prima parte di questa sezione. Il problema è che il padding, intervenendo a livello di pacchetto prima che il flusso venga calcolato, propaga i propri effetti alle feature in modo sistematico e invisibile al rilevatore, che non ha modo di distinguere il traffico perturbato da quello ordinario se non è stato esposto a perturbazioni di questo tipo durante il training.

È esattamente questo il punto di partenza che motiva l'adversarial training, i cui risultati sono presentati nella Sezione 5.4.

Risposta alla Research Question 1 — Vulnerabilità del sistema di rilevamento

Sì: le perturbazioni in *traffic space* producono una degradazione significativa e sistematica della capacità di rilevamento dei modelli clean, in tutte le configurazioni valutate. La recall media scende di 40–58 punti percentuali rispetto alla baseline su traffico non perturbato; per Murlo si azzera completamente a partire dal padding da 256 byte. Le perturbazioni in *traffic space* costituiscono quindi una minaccia concreta per i ML-NIDS addestrati esclusivamente su traffico non perturbato.

5.4 Risultati del modello *Random Forest* hardened su traffico non perturbato e perturbato

Con la vulnerabilità del modello baseline documentata nella Sezione 5.3, si presentano in questa sezione i risultati dei modelli hardened, valutati su traffico non perturbato e su traffico perturbato.

La sezione affronta due aspetti in sequenza. Il primo verifica che il processo di adversarial training non abbia compromesso la capacità dei modelli hardened di operare correttamente su traffico non perturbato. Il secondo valuta la robustezza acquisita quando i modelli hardened vengono esposti a traffico perturbato, sia nella configurazione in cui la tipologia di test coincide con quella di training, sia nel caso più generale in cui le perturbazioni di test non siano state osservate durante l'addestramento.

E' importante ricordare che, conformemente al criterio enunciato nella Sezione 5.1.1, la valutazione è condotta sulle sole famiglie Murlo, Neris e Virut, le uniche per cui il test set perturbato conta più di 100 campioni malevoli e per cui sono stati costruiti i corrispondenti modelli hardened.

5.4.1 Non-regressione del modello *Random Forest* hardened su traffico non perturbato

Prima di valutare la robustezza dei modelli hardened sul traffico perturbato, è necessario verificare che il processo di adversarial training non abbia compromesso la loro capacità di operare correttamente su traffico non perturbato. Questa verifica non è un passaggio secondario: una regressione, anche contenuta, sulla recall ordinaria, annullerebbe il valore pratico del rafforzamento, rendendo il modello hardened inutilizzabile in produzione.

Per ciascuna famiglia e rappresentazione di feature, si considera, per ciascuna delle tre metriche, il valore medio osservato tra tutti i quattordici modelli hardened quando valutati sul test set non perturbato. La media fornisce una stima del comportamento tipico del modello hardened su traffico ordinario, più rappresentativa del caso peggiore isolato. La Tabella 5.4 riporta, per ciascuna combinazione di famiglia e rappresentazione, le metriche

del modello clean sulla baseline, la media dei quattordici modelli hardened sul test set clean, e il delta tra i due.

Tabella 5.4: Confronto tra il modello clean (baseline) e la media dei quattordici modelli hardened, valutati sul test set non perturbato, per famiglia di botnet e rappresentazione di feature. Il delta è calcolato come differenza tra il valore medio degli hardened e la baseline.

		Baseline (clean)			**Hardened medio**			**Delta**		
Famiglia	**Rapp.**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**ΔPrec.**	**ΔRec.**	**ΔF1**
Murlo	NO_FD	1.0000	0.9659	0.9827	1.0000	0.9743	0.9870	+0.0000	+0.0084	+0.0043
Murlo	FD	1.0000	0.9637	0.9815	0.9997	0.9718	0.9855	−0.0003	+0.0081	+0.0040
Neris	NO_FD	0.9777	0.9036	0.9392	0.9788	0.9119	0.9442	+0.0011	+0.0083	+0.0050
Neris	FD	0.9816	0.9050	0.9418	0.9806	0.9055	0.9416	−0.0010	+0.0005	−0.0002
Virut	NO_FD	1.0000	0.9393	0.9687	0.9991	0.9367	0.9669	−0.0009	−0.0026	−0.0018
Virut	FD	1.0000	0.9344	0.9661	1.0000	0.9366	0.9673	+0.0000	+0.0022	+0.0012

Il controllo di non-regressione è superato in tutte le configurazioni. L'unica variazione negativa di recall si osserva per Virut NO_FD, dove il delta medio si attesta a -0.0026. Tutte le variazioni sono inferiori a tre centesimi di punto in valore assoluto e operativamente trascurabili. In diverse configurazioni il delta di recall è addirittura positivo, e analogamente per precision e F1, il che significa che il modello hardened con la recall più bassa sul traffico clean supera comunque la baseline del modello clean corrispondente. Questo risultato, a prima vista controintuitivo, può essere attribuito all'incremento di varietà nel training set: i campioni perturbati aggiuntivi ampliano la distribuzione degli esempi malevoli osservati durante l'addestramento, e questo arricchimento può riflettersi positivamente, seppur in misura contenuta, sulla capacità del modello di classificare anche esempi non perturbati. In ogni caso, che il delta sia leggermente positivo o leggermente negativo, l'entità delle variazioni osservate è in tutti i casi trascurabile dal punto di vista operativo.

Vale la pena sottolineare che questa stabilità vale per il caso peggiore tra i quattordici modelli hardened per ciascuna famiglia: la variabilità complessiva tra i modelli hardened nella stessa configurazione è contenuta, e nessuno di essi produce una regressione apprezzabile sulla recall clean. L'adversarial training, nella forma implementata in questo lavoro, non introduce quindi alcun trade-off rilevante tra robustezza verso il traffico perturbato e capacità di rilevamento del traffico ordinario.

5.4.2 Robustezza del modello *Random Forest* hardened sulla perturbazione utilizzata in addestramento

Con il controllo di non-regressione superato, è possibile verificare che l'adversarial training abbia effettivamente prodotto robustezza verso la tipologia di perturbazione su cui è stato addestrato, ovvero quando il modello hardened viene valutato sul test set perturbato

con la stessa tipologia di padding utilizzata durante il suo addestramento. Questo confronto, che si può chiamare valutazione sulla *diagonale*, costituisce una conferma di correttezza del framework prima di affrontare il caso più interessante della generalizzazione verso perturbazioni mai viste.

I valori diagonali confermano che l'adversarial training funziona in tutte le configurazioni considerate. Per Murlo la recall media sulla diagonale si attesta attorno a 0.9619 (NO_FD) e 0.9598 (FD), con valori minimi di 0.9382 e 0.9194 rispettivamente. Per Neris i valori medi sono 0.8575 (NO_FD) e 0.8295 (FD), con minimi di 0.7478 e 0.6630. Per Virut i valori medi sono 0.8746 (NO_FD) e 0.8650 (FD), con minimi di 0.7665 e 0.7087. In tutte e tre le famiglie i valori diagonali più bassi si osservano in corrispondenza delle tipologie di padding randomico ad ampio intervallo, in particolare Random 1-1500, il che è attribuibile alla variabilità intrinseca dei test set stocastici, generati con seed diversi su dieci run indipendenti.

Stabilito che l'adversarial training produce robustezza nel caso in cui la perturbazione di test coincide con quella di addestramento, la domanda che rimane aperta riguarda la capacità di generalizzazione verso perturbazioni mai viste in training, e a essa è dedicata la parte conclusiva di questa sezione.

5.4.3 Capacità di generalizzazione verso tipologie di perturbazione non osservate in addestramento

I risultati fin qui presentati hanno documentato tre fatti: i modelli clean sono vulnerabili alle perturbazioni in *traffic space*, i modelli hardened non regrediscono su traffico ordinario, e — come mostrato in precedenza — l'adversarial training recupera recall elevata quando il modello viene valutato sulla stessa tipologia di perturbazione utilizzata in training. Si tratta di una conferma necessaria, ma non sufficiente a stabilire l'utilità pratica della difesa proposta.

Il threat model adottato in questo lavoro, descritto nella Sezione 3.1, non garantisce che l'attaccante utilizzi esattamente la stessa strategia di perturbazione contro cui il difensore si è preparato: l'avversario è modellato come un soggetto che opera in modo cieco rispetto al classificatore e che ha libertà di scelta sulla dimensione del padding da applicare ai pacchetti. In uno scenario operativo reale, un difensore che addestrasse il proprio sistema contro perturbazioni di una specifica dimensione fissa potrebbe trovarsi di fronte ad attacchi con dimensioni diverse, e non vi è alcuna garanzia a priori che la robustezza acquisita su una tipologia si trasferisca automaticamente alle altre.

Questa osservazione apre la domanda centrale della presente sezione: un modello hardened su una specifica tipologia di padding è in grado di rilevare con sufficiente affidabilità traffico perturbato con tipologie diverse, mai incontrate durante l'addestramento?

E, più in generale, la capacità di generalizzazione dipende dalla scelta della tipologia di padding utilizzata in training?

Per rispondere a queste domande, si analizza il comportamento dei modelli hardened non più sulla diagonale — ovvero quando la tipologia di test coincide con quella di training — ma sull'intera matrice di valutazione, confrontando sistematicamente la recall di ciascun modello su tutte le quattordici tipologie di perturbazione disponibili, incluse quelle su cui non è stato addestrato. Il confronto mette in luce due comportamenti opposti. Da un lato, i modelli addestrati su padding fisso[3] mostrano un profilo di *specializzazione*: raggiungono recall elevata sulla propria tipologia di training, ma la perdono in modo significativo quando vengono valutati su tipologie diverse. Dall'altro, i modelli addestrati su padding randomico ad ampio intervallo mostrano un profilo di *generalizzazione*: mantengono recall elevata su un insieme molto più ampio di tipologie di test, incluse quelle deterministiche su cui non sono stati addestrati. La distinzione tra questi due profili è il nucleo del confronto che segue.

5.4.4 Specializzazione dei modelli *Random Forest* hardened su padding fisso

I modelli addestrati con perturbazioni a padding fisso presentano un comportamento caratterizzato da alta recall su uno scenario ristretto e scarsa generalizzazione al di fuori di esso. Quando il test set corrisponde alla stessa dimensione di padding usata in training, la recall è consistentemente elevata, come già documentato nella Sezione 5.4.2. Questa coerenza si interrompe però non appena il test set viene generato con una dimensione diversa: i valori di recall calano in modo considerevole e, in diversi casi, scendono a valori prossimi allo zero.

Le Tabelle 5.5 e 5.6 illustrano questo pattern su un sottoinsieme rappresentativo di configurazioni per le famiglie Murlo, Neris e Virut, rispettivamente per le rappresentazioni NO_FD e FD, riportando per ciascun modello hardened la recall osservata sul proprio test set di riferimento, la recall media sugli altri tredici test set perturbati e il valore minimo osservato tra questi ultimi.

[3]Nel seguito il padding fisso è indicato talvolta anche come padding *deterministico*, per enfatizzare la sua natura non stocastica in contrapposizione al padding randomico. I due termini sono usati come sinonimi.

Tabella 5.5: Recall dei modelli hardened su padding fisso: confronto tra la recall sul test set corrispondente alla propria tipologia di training (*propria*), la recall media sugli altri tredici test set perturbati (*media altri*) e il valore minimo osservato (*min altri*). Configurazione NO_FD; configurazioni selezionate a titolo rappresentativo.

Modello	**Famiglia**	**Recall propria**	**Media altri**	**Min altri**
Hardened 8 byte	Murlo NO_FD	0.974	0.386	0.000
Hardened 128 byte	Murlo NO_FD	0.958	0.520	0.000
Hardened 1024 byte	Murlo NO_FD	0.965	0.510	0.008
Hardened 8 byte	Neris NO_FD	0.910	0.490	0.271
Hardened 128 byte	Neris NO_FD	0.852	0.593	0.321
Hardened 1024 byte	Neris NO_FD	0.890	0.550	0.426
Hardened 8 byte	Virut NO_FD	0.930	0.492	0.085
Hardened 128 byte	Virut NO_FD	0.868	0.629	0.134
Hardened 1024 byte	Virut NO_FD	0.872	0.577	0.348

Tabella 5.6: Recall dei modelli hardened su padding fisso: confronto tra la recall sul test set corrispondente alla propria tipologia di training (*propria*), la recall media sugli altri tredici test set perturbati (*media altri*) e il valore minimo osservato (*min altri*). Configurazione FD; configurazioni selezionate a titolo rappresentativo.

Modello	**Famiglia**	**Recall propria**	**Media altri**	**Min altri**
Hardened 8 byte	Murlo FD	0.976	0.414	0.000
Hardened 128 byte	Murlo FD	0.963	0.546	0.000
Hardened 1024 byte	Murlo FD	0.954	0.826	0.634
Hardened 8 byte	Neris FD	0.903	0.450	0.108
Hardened 128 byte	Neris FD	0.844	0.574	0.141
Hardened 1024 byte	Neris FD	0.752	0.551	0.395
Hardened 8 byte	Virut FD	0.934	0.447	0.051
Hardened 128 byte	Virut FD	0.884	0.584	0.051
Hardened 1024 byte	Virut FD	0.858	0.583	0.260

Lo scarto tra la recall sul proprio test set e la media sugli altri è il dato che caratterizza questo comportamento. Con riferimento alla Tabella 5.5 (rappresentazione NO_FD): per Murlo, un modello hardened su 8 byte raggiunge una recall di 0.974 sul test set con 8 byte di padding fisso, ma scende a una media di 0.386 sugli altri tredici test set, con un minimo pari a zero. Valori analoghi si osservano per i modelli addestrati su 128 e

1024 byte: recall elevata sul proprio test, crollo sugli altri, con minimi che toccano o si avvicinano allo zero per quasi tutte le dimensioni di padding fisso su questa famiglia. Su Neris il pattern si ripete con entità minore: il minimo non scende a zero ma rimane comunque molto distante dalla recall sul test di riferimento, con scarti che superano i 30 punti percentuali. Su Virut il comportamento è intermedio tra i due: i minimi si attestano tra 0.09 e 0.35 a seconda della dimensione del padding, e la media sugli altri test set non supera in nessun caso 0.68, contro recall proprie comprese tra 0.85 e 0.93.

I risultati della rappresentazione FD (Tabella 5.6) confermano in larga misura lo stesso pattern, con un'eccezione degna di nota: il modello hardened su 1024 byte per Murlo mostra una generalizzazione sensibilmente superiore rispetto alla controparte NO_FD (minimo 0.634 contro 0.008), suggerendo che le feature derivate modifichino in modo non trascurabile la risposta del modello alla perturbazione di maggiore entità. Nelle altre configurazioni i valori rimangono qualitativamente analoghi a quelli della rappresentazione base.

Pur con intensità diversa da famiglia a famiglia e tra le due rappresentazioni, il pattern di specializzazione è quindi consistente: la recall crolla non appena il test set si discosta dalla dimensione di padding usata in training.

La conseguenza pratica è immediata: un modello addestrato su padding fisso garantisce robustezza soltanto nello scenario in cui l'attaccante utilizzi esattamente la stessa dimensione di padding su cui il difensore si è preparato. Poiché il threat model adottato non vincola l'attaccante a questa scelta, questo tipo di hardening non risulta sufficiente a proteggere il sistema in uno scenario realistico.

5.4.5 Generalizzazione dei modelli *Random Forest* hardened su padding randomico

I modelli addestrati con perturbazioni a padding randomico presentano un comportamento sensibilmente diverso da quello descritto nel blocco precedente, con differenze rilevanti a seconda dell'ampiezza dell'intervallo di campionamento adottato in training.

I modelli addestrati sui due intervalli più ampi, `[100, 1024]` e `[1, 1500]`, mostrano un profilo di generalizzazione esteso: la recall rimane elevata non soltanto quando il test set è generato con la stessa tipologia di training, ma anche quando viene generato con tipologie di perturbazione mai osservate durante l'addestramento, incluse le varianti a padding fisso. Le Tabelle 5.7 e 5.8 riportano la recall media calcolata sull'insieme di tutti e quattordici i test set perturbati disponibili, rispettivamente per le rappresentazioni NO_FD e FD.

Tabella 5.7: Recall media dei modelli hardened con padding randomico ad ampio intervallo, calcolata sull'intero insieme dei quattordici test set perturbati. Rappresentazione NO_FD (base).

Modello / Famiglia	**Recall media (14 test set)**
Random 100–1024 — Murlo	0.8729
Random 1–1500 — Murlo	0.8845
Random 100–1024 — Neris	0.7087
Random 1–1500 — Neris	0.7519
Random 100–1024 — Virut	0.7618
Random 1–1500 — Virut	0.7554

Tabella 5.8: Recall media dei modelli hardened con padding randomico ad ampio intervallo, calcolata sull'intero insieme dei quattordici test set perturbati. Rappresentazione FD (estesa).

Modello / Famiglia	**Recall media (14 test set)**
Random 100–1024 — Murlo	0.9037
Random 1–1500 — Murlo	0.9047
Random 100–1024 — Neris	0.6778
Random 1–1500 — Neris	0.7132
Random 100–1024 — Virut	0.7586
Random 1–1500 — Virut	0.7700

La recall media su tutti i test set si attesta su valori decisamente superiori a quelli osservati per i modelli a padding fisso sulla stessa dimensione di aggregazione, come sarà reso esplicito nel seguito. Per Murlo i valori oscillano tra 0.8729 e 0.9047 a seconda della rappresentazione; per Neris tra 0.6778 e 0.7519; per Virut tra 0.7554 e 0.7700. Questi risultati includono sia test set della stessa tipologia randomica usata in training sia test set con padding fisso, confermando che la robustezza acquisita non è circoscritta a perturbazioni strutturalmente simili a quelle viste durante l'addestramento.

5.4.6 Discussione e risultati principali

Il caso limite del padding randomico su intervallo ristretto. Un caso che merita attenzione separata è quello del modello addestrato sull'intervallo `[1, 100]`, che è l'intervallo adottato nel riferimento implementativo originale [5] e copre perturbazioni di entità

contenuta. A differenza dei modelli addestrati su intervalli più ampi, questo modello non generalizza in modo soddisfacente verso test set che includono padding di dimensioni superiori ai 100 byte. La Tabella 5.9 riporta la recall di questo modello quando valutato sul test set `Random 1-1500`, che comprende padding di tutte le dimensioni fino al limite MTU.

Tabella 5.9: Recall del modello hardened addestrato con padding randomico `[1, 100]` quando valutato sul test set perturbato con padding randomico `[1, 1500]`, per famiglia di botnet e rappresentazione di feature.

Famiglia	**NO_FD**	**FD**
Murlo	0.0764	0.0172
Neris	0.3730	0.2413
Virut	0.2551	0.1384

I valori sono molto distanti da quelli osservati per i modelli addestrati su intervalli più ampi: su Murlo la recall scende a 0.0764 nella rappresentazione NO_FD e a 0.0172 nella rappresentazione FD. Il modello Random 1–100, durante il training, non ha mai osservato perturbazioni superiori ai 100 byte, e il test set `Random 1-1500` include invece padding che possono raggiungere i 1500 byte: il test set cade parzialmente al di fuori della distribuzione su cui il modello è stato addestrato, e la recall riflette questa mancata copertura.

Questo caso limite agisce da controprova della conclusione principale: non è la natura stocastica della perturbazione in sé a produrre generalizzazione, ma la copertura dell'intervallo di variazione durante il training. Un modello addestrato su un intervallo randomico ristretto si comporta, dal punto di vista della generalizzazione, in modo più vicino a un modello a padding fisso che a un modello addestrato su un intervallo ampio.

Sintesi e implicazioni per la progettazione della difesa. Il confronto tra i due approcci produce una conclusione che non era scontata a priori. La Tabella 5.10 rende il divario quantitativo esplicito: per ciascuna famiglia e rappresentazione di feature, si riporta la recall media su tutti e quattordici i test set perturbati, mettendo a confronto i modelli random ampi con la media aggregata dei modelli a padding fisso.

Tabella 5.10: Recall media su tutti e quattordici i test set perturbati: confronto tra la media aggregata dei modelli hardened su padding fisso (undici modelli per configurazione) e i modelli hardened Random 100–1024 e Random 1–1500.

Famiglia	**Rapp.**	**Fisso (media)**	**Random 100–1024**	**Random 1–1500**
Murlo	NO_FD	0.5242	0.8729	0.8845
Murlo	FD	0.6000	0.9037	0.9047
Neris	NO_FD	0.5708	0.7087	0.7519
Neris	FD	0.5556	0.6778	0.7132
Virut	NO_FD	0.5999	0.7618	0.7554
Virut	FD	0.5782	0.7586	0.7700

Il divario è consistente in tutte le configurazioni. La media aggregata dei modelli hardened su padding fisso si colloca, a seconda della famiglia e della rappresentazione, tra 0.5242 e 0.6000; i modelli random ampi si attestano invece tra 0.6778 e 0.9047. Lo scarto minimo osservato è di circa 12 punti percentuali, quello massimo supera i 30.

Va sottolineato che la media aggregata dei modelli hardened su padding fisso è una misura ottimistica rispetto a un singolo modello hardened considerato isolatamente, poiché riflette il contributo di undici modelli ciascuno dei quali ha recall alta sulla propria diagonale. Un singolo modello hardened su padding fisso, preso isolatamente, mostra medie sugli altri test nettamente inferiori, come già documentato nelle Tabelle 5.5 e 5.6. Il confronto è quindi favorevole al padding fisso nel senso più generoso possibile, e i modelli random ampi lo superano comunque con margine ampio.

Dal punto di vista della progettazione della difesa, la conseguenza pratica è diretta. Poiché il threat model adottato prevede un attaccante con libertà di scelta sulla dimensione del padding, un difensore che optasse per l'adversarial training su una singola dimensione fissa sarebbe robusto soltanto nello scenario improbabile in cui l'attaccante adotti esattamente quella strategia. Un adversarial training condotto su un intervallo randomico ampio produce invece un modello che si difende efficacemente da una gamma molto più ampia di strategie di padding, senza richiedere di anticipare le scelte specifiche dell'avversario.

Tra i due intervalli ampi considerati, `[100, 1024]` e `[1, 1500]`, le differenze in termini di recall media sono contenute, come mostrato nelle tabelle precedenti. Il modello Random 1–1500, tuttavia, copre per costruzione l'intero spettro di variazione compatibile con il vincolo MTU, includendo anche perturbazioni di piccola entità che il modello Random 100–1024 non incontra in training. Questa copertura più ampia lo rende la scelta preferibile in un'ottica difensiva prudente, indipendentemente dal margine numerico rispetto all'alternativa.

I risultati complessivi presentati in questo capitolo indicano che l'adversarial training

in *traffic space* costituisce una strategia difensiva efficace e praticabile: i modelli hardened recuperano robustezza sul traffico perturbato senza penalizzare le prestazioni su traffico ordinario, e l'adozione di perturbazioni randomiche ad ampio intervallo in fase di training produce generalizzazione verso tipologie di attacco non anticipate. Le implicazioni di questi risultati, unitamente alle limitazioni del lavoro e alle direzioni di sviluppo futuro, sono discusse nel Capitolo 6.

Risposta alla Research Question 2 — Efficacia della difesa proposta

Sì: l'adversarial training in *traffic space* produce modelli al contempo robusti verso il traffico perturbato e stabili su traffico ordinario. Il controllo di non-regressione è superato in tutte le sei configurazioni valutate, con variazioni assolute di recall inferiore a un punto percentuale. La robustezza acquisita dipende però dalla tipologia di perturbazione adottata in addestramento: i modelli hardened su padding fisso non generalizzano fuori dalla dimensione di training, mentre i modelli addestrati su padding randomico ad ampio intervallo mantengono recall media compresa tra 0.68 e 0.90 sull'intero insieme dei quattordici test set perturbati, incluse le tipologie mai osservate in addestramento.

Capitolo 6

Conclusioni

I sistemi di rilevamento delle intrusioni basati su machine learning offrono prestazioni elevate su traffico ordinario ma risultano vulnerabili, anche in misura significativa, di fronte a manipolazioni che un attaccante può applicare al traffico in transito. Il quadro emerso dal Capitolo 2 mostra che la letteratura ha affrontato questo problema soprattutto sul versante dell'attacco e prevalentemente in *feature space*, lasciando meno esplorata la combinazione fra adversarial training come strategia difensiva e perturbazioni costruite direttamente sul traffico di rete. È in questo perimetro che si è collocato il presente lavoro, che ha definito, implementato e valutato un framework di adversarial training operante in *traffic space*, in cui gli esempi adversariali utilizzati per l'addestramento derivano da perturbazioni protocol-compliant applicate a livello di pacchetto. Lo scenario sperimentale è stato circoscritto in modo esplicito già nel Capitolo 3: classificazione binaria per famiglia di botnet su una selezione di scenari CTU-13, classificatore Random Forest, perturbazioni applicate al solo traffico TCP, modello di attaccante cieco rispetto al sistema bersaglio. Su questo perimetro sono stati addestrati i modelli baseline, ne è stata misurata la vulnerabilità a quattordici tipologie di perturbazione di padding e sono stati infine prodotti i corrispondenti modelli hardened tramite adversarial training su validation set perturbati.

I risultati raccolti nel Capitolo 5 si articolano su tre piani complementari. Anzitutto, i modelli baseline si sono mostrati fragili rispetto al traffico perturbato: la recall media è scesa di circa 45–55 punti percentuali rispetto al traffico clean, con cadute a valori prossimi allo zero in corrispondenza dei padding fissi di entità maggiore. La vulnerabilità si manifesta dunque non solo come degrado statistico, ma in alcune configurazioni come incapacità sostanziale di riconoscere il traffico malevolo perturbato; senza una difesa, modelli che in condizioni ordinarie classificano correttamente oltre il novanta per cento dei flussi malevoli arrivano a riconoscerne meno della metà. A questo quadro l'adversarial training in *traffic space* ha risposto producendo modelli capaci di recuperare la recall sul traffico perturbato della stessa tipologia osservata in addestramento, riportandola in media fra 0.83 e 0.96 a seconda della famiglia e della rappresentazione di feature, e ciò senza che la prestazione su traffico clean ne risentisse: la variazione assoluta osserva-

ta sulla recall non perturbata si è mantenuta inferiore a un punto percentuale in tutte le configurazioni. La difesa adottata non ha quindi introdotto un trade-off apprezzabile fra robustezza acquisita e capacità del modello di operare su traffico ordinario, condizione necessaria perché un intervento di hardening possa essere considerato deployabile. A questa evidenza si affianca un terzo risultato, che riguarda la scelta della tipologia di perturbazione utilizzata in training. Il confronto fra le strategie basate su padding deterministico e quelle basate su padding randomico ad ampio intervallo ha messo in evidenza un comportamento opposto delle due famiglie di modelli: i modelli addestrati su padding fisso si specializzano sulla dimensione vista in training e perdono efficacia altrove, mentre quelli addestrati su un intervallo randomico ampio mantengono recall media compresa fra 0.68 e 0.90 sull'intero insieme dei test set perturbati, incluse tipologie mai osservate durante l'addestramento, con uno scarto rispetto alla media aggregata dei modelli a padding fisso che si colloca fra circa dodici e oltre trenta punti percentuali a seconda della famiglia e della rappresentazione di feature.

Letti nel loro insieme, questi risultati offrono due indicazioni di portata più generale. La prima è che adversarial training e perturbazioni in *traffic space*, combinati nella forma descritta in questa tesi e applicati a un classificatore standard sul CTU-13, producono modelli con robustezza misurabile contro la classe di perturbazioni considerata: un'evidenza empirica che riguarda specificamente lo scenario qui sperimentato, ma che contribuisce ad arricchire un'area meno presidiata dalla letteratura esaminata nel Capitolo 2. La seconda indicazione, di natura più operativa, riguarda la progettazione della difesa: per un difensore che non possa anticipare la dimensione esatta delle perturbazioni adottate dall'attaccante, una strategia di adversarial training basata su un intervallo randomico ampio produce robustezza più trasferibile rispetto a una strategia che fissi a priori la dimensione del padding. È un'indicazione che riguarda specificamente il framework adottato e che merita di essere verificata su strategie perturbative diverse da quella considerata in questa tesi.

Queste considerazioni vanno tuttavia lette tenendo presenti i confini entro cui il lavoro si è mosso. Alle scelte di perimetro già richiamate in apertura si aggiungono due ulteriori elementi di scoping rilevanti per la lettura dei risultati. Il confronto fra modelli baseline e modelli hardened, in particolare, è stato condotto sulle tre famiglie di botnet del CTU-13 — Murlo, Neris e Virut — corrispondenti agli scenari che hanno superato la soglia minima di flussi perturbabili adottata per ottenere stime statisticamente sostenibili della recall; l'analisi è inoltre condotta su un unico algoritmo di classificazione, il Random Forest, scelta giustificata dalla sua diffusione in letteratura e coerente con i riferimenti metodologici adottati, ma che non esclude che modelli di natura diversa possano rispondere all'adversarial training in modo differente.

Proprio a partire da questi confini si delineano le direzioni di estensione che il lavoro lascia aperte. La più immediata riguarda la copertura del CTU-13 sul lato delle

famiglie di botnet: Rbot, NSIS, Menti e Sogou contengono quantità di flussi perturbabili inferiori al limite adottato per garantire stime statisticamente sostenibili della recall, e il loro inserimento nel quadro sperimentale potrebbe avvenire tramite strategie di augmentation mirata o tramite l'integrazione di dataset complementari che coprano scenari analoghi, completando così la copertura del dataset senza compromettere la solidità delle stime. Una seconda estensione naturale è l'ampliamento del vocabolario di perturbazioni considerate: il lavoro si è concentrato sul padding come trasformazione di riferimento, sufficiente a costruire una varietà ampia di test set adversariali ma chiaramente non esaustiva, e trasformazioni quali splitting dei pacchetti, riordinamento, manipolazione delle temporizzazioni e inserzione di pacchetti fittizi possono essere integrate nello stesso framework per verificare se la generalizzazione osservata per il padding randomico si conservi anche su altre dimensioni di variazione. Un passo successivo, e più impegnativo, consiste nell'estensione del modello di attaccante oltre quello cieco adottato in questo lavoro: le perturbazioni considerate sono stocastiche e indipendenti dal classificatore bersaglio, scelta coerente con il threat model dichiarato ma che non esaurisce lo spettro dei comportamenti possibili, e attaccanti più informati, capaci di sfruttare un qualche tipo di feedback sul classificatore per orientare le proprie perturbazioni, costituiscono una classe di avversari più aggressiva rispetto alla quale la robustezza ottenuta in questo lavoro non è stata misurata. Resta infine la prospettiva di una replica del framework su ulteriori dataset di traffico. Il CTU-13 è uno standard consolidato per la ricerca su ML-NIDS e ha rappresentato in questo lavoro la scelta naturale, in virtù della sua diffusione in letteratura e della disponibilità di PCAP malevoli accessibili; un'estensione ad altri dataset, di scala maggiore o di natura complementare, consentirebbe di osservare in che misura le indicazioni ottenute si conservino al variare delle caratteristiche del traffico e della composizione del dataset.

Il quadro complessivo che emerge è quello di una difesa praticabile contro perturbazioni realistiche del traffico di rete, costruita interamente sul piano del traffico stesso e in grado di mantenere la propria efficacia anche a fronte di varianti dell'attacco non anticipate, almeno nei limiti dello scenario considerato. Le direzioni qui delineate tracciano il percorso lungo il quale questa praticabilità potrà essere ulteriormente messa alla prova, ed è in tale percorso che si colloca, idealmente, il prosieguo di questa linea di ricerca.

Bibliografia

[1] Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. Hardening random forest cyber detectors against adversarial attacks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(4):427–439, 2020.

[2] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management*, 17(4):1975–1987, 2020.

[3] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. Modeling realistic adversarial attacks against network intrusion detection systems. *ACM Digital Threats: Research and Practice*, 3(3):1–31, 2022.

[4] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[5] Giovanni Apruzzese, Aurore Fass, and Fabio Pierazzi. When adversarial perturbations meet concept drift: An exploratory analysis on ML-NIDS. In *Proceedings of the 17th ACM Workshop on Artificial Intelligence and Security (AISec '24)*, Salt Lake City, UT, USA, 2024. ACM.

[6] Giovanni Apruzzese, Luca Ferretti, Michele Colajanni, and Mirco Marchetti. Addressing adversarial attacks against security systems based on machine learning. In *Proceedings of the 11th IEEE International Conference on Cyber Conflict (CyCon)*, pages 1–18. IEEE, 2019.

[7] Giovanni Apruzzese, Luca Ferretti, Mirco Marchetti, Michele Colajanni, and Alessandro Guido. On the effectiveness of machine and deep learning for cyber security. In *Proceedings of the 10th IEEE International Conference on Cyber Conflict (CyCon)*, pages 371–390. IEEE, 2018.

[8] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. SoK: Pragmatic assessment of machine learning for network intrusion detection. In *Proceedings of the 8th*

IEEE European Symposium on Security and Privacy (EuroS&P), pages 457–476. IEEE, 2023.

[9] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova. The role of machine learning in cybersecurity. *ACM Digital Threats: Research and Practice*, 4(1):1–38, 2023.

[10] Marta Catillo, Antonio Pecchia, Aniello Repola, and Umberto Villano. Towards realistic problem-space adversarial attacks against machine learning in network intrusion detection. In *Proceedings of the 19th International Conference on Availability, Reliability and Security (ARES 2024)*. ACM, 2024.

[11] Jacopo Cortellazzi, Feargus Pendlebury, Daniel Arp, Erwin Quiring, Fabio Pierazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. *ACM Transactions on Privacy and Security*, 26(4):1–41, 2023.

[12] Dimitri Galli, Giovanni Gambigliani Zoccoli, Daniele Bianchini, Dario Stabili, and Mirco Marchetti. Evading ML network intrusion detection systems for Modbus TCP with problem-space perturbations. In *Proceedings of the Joint National Conference on Cybersecurity (ITASEC & SERICS 2026)*, volume 4198 of *CEUR Workshop Proceedings*, Cagliari, Italy, 2026. CEUR-WS.org.

[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, 2015.

[14] Dongqi Han, Zhiliang Wang, Wenqi Zhong, Weiran Chen, Jiahai Yang, Shuqing Lu, Xingang Shi, and Xia Yin. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal on Selected Areas in Communications*, 39(8):2632–2647, 2021.

[15] Keane Lucas, Mahmood Pai, Weiran Lin, Lujo Bauer, Michael K. Reiter, and Mahmood Sharif. Adversarial training for raw-binary malware classifiers. In *Proceedings of the 32nd USENIX Security Symposium*, pages 1–18, Anaheim, CA, USA, 2023. USENIX Association.

[16] Aleksander Mądry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*, Vancouver, BC, Canada, 2018.

[17] Matteo Manocchia. Adversarial training in traffic space per ML-NIDS. `https://github.com/matteo-manocchia/Tesi_Adeversarial_Training_MLNIDS`, 2025. Repository GitHub del codice sperimentale associato alla tesi.

[18] Stratosphere IPS. CTU-13 Dataset. https://www.stratosphereips.org/datasets-ctu13, 2013. Czech Technical University, Prague.

[19] Andrea Venturi, Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. DReLAB – deep REinforcement learning adversarial botnet: A benchmark dataset for adversarial attacks against botnet intrusion detection systems. *Data in Brief*, 34:106631, 2021.

[20] Miel Verkerken, Laurens D'hooge, Bruno Volckaert, Filip De Turck, and Giovanni Apruzzese. "What is the Problem Space?" Defining Host-space Adversarial Perturbations against Network Intrusion Detection Systems. In *Proceedings of the 21st ACM Asia Conference on Computer and Communications Security (ASIA CCS '26)*, Bangalore, India, 2026. ACM.