Università degli Studi di Milano
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

CORSO DI LAUREA TRIENNALE IN
SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE

DYNAMIC ARP INPECTION OPEN SOURCE UNA SOLUZIONE ACCESSIBILE E ADATTABILE AL SERVIZIO DELLA SICUREZZA DEGLI APPARATI DI RETE E DEGLI UTENTI

Relatore: Prof. Ernesto Damiani

Elaborato Finale di:
Alessandro Meloni
Matr. Nr. 984857

Anno Accademico 2025-2026

Questo lavoro è dedicato ai miei Genitori e alla mia Fidanzata

Mamma e Papà vi voglio infinitamente bene.

Sapere che il vostro supporto fosse incondizionato è stato fondamentale durante questo percorso tortuoso.

Mi ha permesso di sentire sempre la vostra fiducia, anche quando pensavo di non essere all'altezza voi avete sempre puntato su di me.

Rendervi orgogliosi è la soddisfazione più forte di questa esperienza.

L'educazione e l'affetto datomi in questi anni è il dono più grande che abbia mai ricevuto, ve ne sarò eternamente grato.

Mi auguro di essere all'altezza di farvi venire gli stessi occhi lucidi in molte altre occasioni.

Sono grato di avere due Genitori così, vi ringrazio di tutto.

Cate, ti amo.

Fin dal primo sguardo, fin dal 2017. Da allora una sola costante. Tu.

Diventasti Tu il motivo dei miei giorni, dei miei sorrisi. Mi cambiasti per sempre.

Me ne innamorai perdutamente, di quella splendida Ragazza.

Il tuo amore incondizionato riscalderà la mia anima per il resto dei miei giorni.

Sarò onorato di rispettarlo e ricambiarlo, con tutto me stesso.

Il tuo Melo.

Ringraziamenti

Genitori, come detto in precedenza nelle dediche, ...

Cate, ...

Successivamente, diventasti parte della mia quotidianeità nello stesso periodo della mia prima sessione. Da allora, ne passarono molte altre. Ma tu c'eri sempre, anche quando non eravamo insieme.

La tua pazienza di arrivare a questo obbiettivo è stata proporzionale all'amore che provavi per me, grazie a questo ai potuto rinunciare a molte cose più di aspettare con ansia il mio successo. Un sacrificio nobile di una persona più che altruista.

Indice

Capitolo 1

Introduzione

Il presente elaborato si pone l'obiettivo di analizzare le criticità di sicurezza nelle reti locali, con particolare riferimento alle vulnerabilità del protocollo ARP, e di proporre una soluzione difensiva open source.

La trattazione è organizzata secondo un percorso logico che parte dai fondamenti teorici delle reti di calcolatori, indispensabili per comprendere il contesto operativo. Successivamente, viene discusso lo stato dell'arte, esaminando le soluzioni di sicurezza già presenti in letteratura e identificandone i limiti che hanno motivato lo sviluppo di una nuova implementazione.

Il cuore del lavoro verte sulla progettazione e lo sviluppo di un'architettura software inedita, di cui vengono dettagliate le scelte tecniche e le strategie di ottimizzazione. In conclusione, vengono presentate le evidenze sperimentali raccolte durante la fase di testing e delineate le possibili evoluzioni future del progetto.

1.1 Reti di Elaboratori, livello Fisico e Data Link del modello ISO/OSI

Le reti di elaboratori sono costituite da innumerevoli protocolli e dispositivi, la cui unione garantisce una trasmissione continua di dati attraverso molteplici client, server e reti. Lo schema logico più rappresentativo con il quale si riesce a modellare al meglio i principali protocolli e dispositivi di rete è il modello ISO/OSI, che prevede 7 livelli a disposizione per associare protocolli come IP, Ethernet, TCP, HTTP, etc.

I primi tre livelli, in particolare, ovvero il livello Fisico, Data Link e Rete, sono le fondamenta indispensabili per la connessione tra elaboratori; essi stabiliscono le regole di ingaggio tra dispositivi di rete fisici, gli indirizzi di comunicazione con cui individuare gli utenti, tra cui MAC e IP, e la gestione di reti multiple con gli indirizzi IP.

Nel dettaglio:

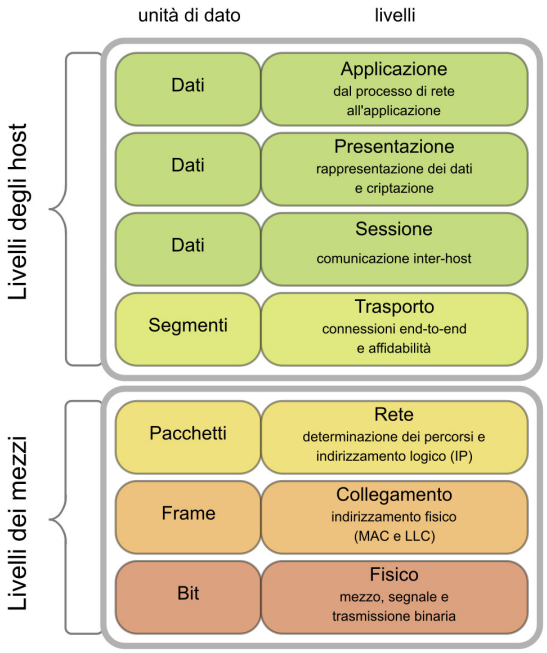

Figura 1: Modello ISO/OSI

- Al livello fisico appartengono le convenzioni di comunicazione fisica tramite cavi in rame o fibra ottica, le conversioni da segnale elettrico ad analogico e gli standard di voltaggio per comunicare dati sotto forma di bit.
- Al secondo livello di Data Link risiedono protocolli, tra cui:
  - MAC - Media Access Control
  - LLC - Logical Link Control (IEEE 802.2)
  - Ethernet (IEEE 802.3)
  - Wi-Fi (IEEE 802.11)

Il primo protocollo regola l'accesso al mezzo fisico trasmissivo tramite il quale più schede di rete possono comunicare usando l'omonimo indirizzo MAC, composto da 48 bit o 12 cifre esadecimali univocamente identificabili (es. 00:1A:2B:3C:4D:5E). L'indirizzo MAC è usato come indirizzo di origine e di destinazione all'interno del frame del Data Link. Lo switch legge l'indirizzo MAC di destinazione per sapere esattamente su quale porta inoltrare il frame, garantendo che i dati vengano consegnati solo al dispositivo corretto sulla rete locale.

Il secondo livello, a differenza del MAC, si concentra maggiormente sul collegamento logico tra due dispositivi, su come stabilirlo e mantenerlo. Infatti, viene considerata la metà

superiore del livello Data Link, in quanto fornisce funzionalità utili al livello sovrastante di Rete. Il multiplexing è una di queste: la possibilità di distinguere pacchetti di livello 3 appartenenti a diversi protocolli nello stesso campo dati di una trama Data Link, come ad esempio IP e IPX. C'è da dire che ormai gran parte del traffico è incapsulato nelle trame utilizzando IP a livello 3 (per esempio, anche lo stesso IPSec è un pacchetto IP).

Ethernet e Wi-Fi sono, rispettivamente, due versioni del protocollo MAC su mezzo fisico cablato e wireless.

Ora, tenendo sempre ben presente i livelli dello stack ISO/OSI, verranno descritti i protocolli protagonisti del livello di Rete e gli intermediari a cavallo tra esso e il livello Data Link.

1.2 Livello di Rete e il ruolo del protocollo ARP

Il livello di Rete è responsabile della comunicazione tra reti geograficamente e logicamente differenti, seguendo il percorso non solo dal mittente al destinatario di una rete locale, come il livello Data Link, ma anche attraverso le molteplici reti e dispositivi intermedi Esso costituisce l'infrastruttura di rete, grazie principalmente al protocollo IP, Internet Protocol [1], il quale assegna a ogni dispositivo un indirizzo di 32 bit arbitrario, in base alla configurazione statica o dinamica.

Un indirizzo, oltre al MAC, già di per sé univoco per ogni dispositivo di rete, nasce dalla necessità di una migliore scalabilità delle reti, in quanto l'indirizzo MAC è univoco, ma non gerarchico. Assumendo l'assenza di un'infrastruttura multi-reti appartenente al livello Tre, si avrebbe una gigantesca rete locale di livello Due con innumerevoli host e un ipotetico switch di livello due che permette loro di comunicare: ogni pacchetto inviato comporterebbe una ricerca nella tabella di enormi dimensioni contenente le proprie schede e il relativo MAC dell'host collegato, comportando inevitabilmente una notevole latenza ad ogni pacchetto. Oltre al protocollo IP, spina dorsale di questo livello, sono presenti i protagonisti oggetto della discussione: DHCP e ARP.

1.2.1 DHCP

Il Dynamic Host Configuration Protocol [2], come suggerisce il nome, è il servizio di configurazione dinamica degli Host tramite una modalità di domanda-offerta che si conclude con l'attribuzione di indirizzi IP, ovvero un leasing di indirizzi con scadenza. Di fatti, assumendo che sia abilitata l'assegnazione dinamica degli indirizzi IP, nel momento in cui un dispositivo si connette a una rete possiede solo il proprio MAC Address per comunicare e non ha accesso a una comunicazione superiore al livello 2. Inoltre, non conosce nemmeno gli indirizzi MAC degli altri dispositivi; dunque ha la necessità di individuarli per poter iniziare una comunicazione.

Per completezza, bisogna specificare che il DHCP non risiede strettamente a livello 3, bensì è classificato come protocollo/servizio di livello Applicazione. Ciò è dovuto al fatto che fornisce un servizio di configurazione di rete utile alle applicazioni, o più genericamente al sistema operativo, per connettersi alla rete. Utilizza pacchetti UDP, livello di Trasporto, e porte dedicate: 67 per la ricezione e 68 per l'invio al client.

Il leasing di indirizzi IP è composto da quattro fasi, note con l'acronimo "DORA":

- DHCP Discover
- DHCP Offer
- DHCP Request
- DHCP Acknowledge

La prima azione effettuata dal client è chiedere la presenza di un server DHCP nella rete che possa aiutarlo ad acquisire un indirizzo IP. Viene inviato un pacchetto che ha come indirizzi IP e MAC destinatari il valore di broadcast, rispettivamente a 32 e 48 bit tutti composti da 1 (255.255.255.255 e FF:FF:FF:FF:FF:FF); oltre ad essi, è presente l'indirizzo IP sorgente interamente impostato a zero e il MAC del mittente.

A seguito del pacchetto DHCP Discover, avviene la ricezione da parte del server sulla porta apposita, l'elaborazione della risposta in funzione degli IP disponibili e il conseguente invio del pacchetto DHCP Offer: a livelli superiori al livello di Rete, ovvero nel corpo del pacchetto UDP, è presente l'IP da assegnare al client e le opzioni della lease, come il tempo di scadenza, la maschera di rete e l'indirizzo del Gateway e DNS. A livello Tre sono presenti l'IP mittente del server e l'IP di destinazione impostato con indirizzo di broadcast; d'altronde, il client non ne possiede ancora uno. A livello Due vengono trascritti il MAC del server e quello del client.

Il client riceve una o più offerte dai server DHCP, dato che è possibile avere un numero arbitrario di server DHCP in rete. Scegliere quale offerta ha maggiore priorità dipende principalmente dall'efficienza. In genere, si predilige la prima offerta che viene validata, eseguendo un controllo che i campi del messaggio siano corretti e plausibili.

Scelta l'offerta, viene inviato il pacchetto DHCP Acknowledge per confermare al server DHCP la scelta e rendere ufficiale il lease. Da questo punto in poi, il client ha accesso al livello di Rete, può utilizzare IP e tutti i servizi che si basano su di esso: inoltrare pacchetti in differenti reti, avere accesso a servizi di livello Applicazione, etc.

Conclusa la spiegazione riguardo al leasing degli indirizzi IP, segue un breve riepilogo del suo ruolo nella rete. Esso, oltre a inizializzare gli host per la comunicazione a livello Tre, gestisce in memoria una tabella che contiene tutti i lease effettuati con i parametri descritti in precedenza. Dato che solitamente il servizio viene eseguito su router, come nel caso di una LAN con un solo dispositivo di livello Tre, quest'ultimo ha modo di vedere tutto il traffico in transito nello stesso dispositivo dove viene eseguito il DHCP. Questa

nota tornerà utile successivamente nelle sezioni riguardanti ARP e i problemi di sicurezza di quest'ultimo.

1.2.2 ARP

L'Address Resolution Protocol (ARP) [3] è un protocollo di mappatura dinamica fondamentale che opera come ponte tra il livello di Rete e il livello Data Link all'interno di una rete locale. La sua esistenza è resa necessaria dal fatto che, sebbene i pacchetti IP contengano un indirizzo logico universale per l'instradamento globale, la consegna fisica effettiva di un frame a un nodo adiacente richiede l'indirizzo hardware specifico, ovvero il MAC address. ARP risolve il problema di trovare l'indirizzo MAC di destinazione quando si conosce solo il suo indirizzo IP. Ad esempio, tramite DHCP viene fornito l'indirizzo del Gateway: 192.168.1.1. Non conoscendo il MAC, non si è in grado di realizzare un frame di livello 2 che possa incapsulare l'IP; da qui la necessità di chiedere a quale MAC appartenga un certo indirizzo IP.

Il funzionamento di ARP si basa su un meccanismo di richiesta e risposta, formalmente ARP Request e ARP Reply. Quando un dispositivo desidera comunicare con un altro host sulla stessa rete locale conoscendo il suo indirizzo IP ma non il suo indirizzo MAC, il dispositivo mittente genera un messaggio ARP Request. Questo messaggio incapsula l'indirizzo IP del destinatario richiesto e viene inviato in broadcasting a tutti i nodi della rete locale. A Livello 2, il frame che trasporta la richiesta ARP ha l'indirizzo MAC di destinazione impostato su un valore di broadcast, garantendo che ogni dispositivo sulla LAN riceva e processi la richiesta.

Tutti i dispositivi ricevono la richiesta ARP e, leggendo il campo che contiene l'indirizzo IP richiesto, lo confrontano con il proprio indirizzo IP. Solo il dispositivo che riconosce l'indirizzo IP come proprio risponde. Questo dispositivo genera un messaggio ARP Reply, che è di tipo unicast, indirizzato specificamente all'indirizzo MAC del mittente originale della richiesta. Il messaggio di risposta contiene la mappatura desiderata: "L'indirizzo IP X.X.X.X corrisponde all'indirizzo MAC Y:Y:Y:Y:Y:Y".

Una volta ricevuta la risposta ARP, il dispositivo mittente memorizza l'associazione IP-MAC in una memoria temporanea chiamata cache ARP, sostanzialmente una tabella simile alla tabella delle lease DHCP. Questa cache è cruciale per l'efficienza: finché l'associazione rimane valida (cioè non scade), il dispositivo non ha bisogno di ripetere il processo ARP per lo stesso indirizzo IP, ma può inserire direttamente il MAC address di destinazione nel frame di Livello 2 ogni volta che deve inviare pacchetti al determinato host. La durata della validità delle voci nella cache è gestita da un timer; una volta scaduta la voce, il dispositivo dovrà eseguire una nuova richiesta ARP per confermare la mappatura.

ARP è un protocollo di instradamento locale e non attraversa mai i router. Quando un dispositivo deve inviare traffico a un host che si trova su una rete diversa, non esegue l'ARP per l'host remoto. Invece, esegue l'ARP per il suo Gateway Predefinito (il router).

Dopo aver ottenuto il MAC address del router, il dispositivo incapsula il pacchetto IP (che ha ancora l'IP di destinazione remoto) nel frame di Livello 2 indirizzato al MAC del router. Il router, a sua volta, estrae il pacchetto IP e ne determina il percorso successivo, ripetendo il processo ARP sulla rete successiva se necessario. Questo dimostra chiaramente il ruolo di ARP come meccanismo essenziale di risoluzione degli indirizzi per la consegna del "next hop" sulla LAN.

1.3 I problemi di sicurezza del protocollo ARP

L'Address Resolution Protocol è notoriamente conosciuto per essere stato sviluppato senza schemi di autenticazione o verifica dei messaggi; la sua semplicità lo rende completamente privo di controlli di sicurezza e causa diverse vulnerabilità.

Il principale e più noto problema di sicurezza associato all'ARP è l'ARP Cache Poisoning (o ARP Spoofing). L'ARP Poisoning è una tecnica di attacco in cui l'attaccante sfrutta una fondamentale vulnerabilità del protocollo: il fatto che sia stateless, ovvero privo di stato di esecuzione, e non richieda autenticazione. Ciò comporta che il dispositivo aggiorni la sua cache ARP basandosi su qualsiasi messaggio ARP Reply ricevuto, anche se non ha mai inviato una richiesta ARP iniziale.

L'attacco consiste nell'invio di messaggi ARP Reply falsificati a uno o più dispositivi. Per sfruttare al meglio la vulnerabilità, l'attaccante invia un ARP Reply falsificato al Gateway, il router, affermando che l'indirizzo IP del client vittima corrisponde al MAC address dell'attaccante; al contempo, invia un secondo ARP Reply alla vittima, affermando che l'indirizzo IP del router corrisponde al MAC address dell'attaccante. Sia il router che il client aggiornano le proprie cache con le informazioni falsificate. La conseguenza diretta è l'attacco Man-in-the-Middle (MITM), in cui l'attaccante si interpone nel percorso di comunicazione. Tutto il traffico destinato a una delle due parti (client o router) viene inoltrato all'indirizzo MAC dell'attaccante, il quale può intercettare, monitorare e potenzialmente alterare i dati; per risultare invisibile agli occhi delle vittime, potrà poi inoltrare il traffico alle rispettive destinazioni.

QUA FIGURA SCHEMA DI ATTACCO

Capitolo 2

Soluzioni ad ARP Poisoning

Nella letteratura vengono proposte numerose soluzioni a questa vulnerabilità, presentando diversi modi per risolvere le lacune intrinseche del protocollo ARP. Per chiarezza di analisi, le soluzioni presenti in letteratura sono state suddivise in due gruppi: le proposte distribuite, implementate sui singoli host presenti sulla rete, e quelle centralizzate, implementate sui router.

2.1 Soluzioni Host-Oriented

In primo luogo, presentiamo le soluzioni implementate sui singoli nodi della rete per validare l'intera infrastruttura: un paradigma interessante, ma con alcuni punti critici.

2.1.1 S-ARP: Secure Address Resolution Protocol di D. Bruschi, A. Ornaghi e E. Rosti

Nell'articolo [4] viene presentata un'evoluzione del protocollo ARP, estendendolo con robusti meccanismi di autenticazione e integrità basati sulla crittografia asimmetrica per contrastare gli attacchi di ARP poisoning. L'architettura di S-ARP non mira a fornire la riservatezza del traffico, funzione delegata ai livelli superiori dello stack OSI, ma si concentra esclusivamente sull'autenticazione.

Ogni host abilitato a S-ARP possiede una coppia di chiavi pubblica/privata. L'infrastruttura di sicurezza è centralizzata attorno all'Authoritative Key Distributor (AKD), un host fidato che funge da repository di chiavi. L'AKD gestisce il binding tra l'identità dell'host e la sua chiave pubblica, validata dal gestore di rete. Un ruolo critico dell'AKD è anche la distribuzione di un valore di orologio (time-stamp), fondamentale per la sincronizzazione degli host e la prevenzione degli attacchi di replay. Tutti i messaggi ARP Reply sono firmati digitalmente dal mittente con la sua chiave privata. Il destinatario verifica la firma

utilizzando la chiave pubblica del mittente, recuperandola dall'AKD se è assente o non valida nella cache locale.

Nelle reti con indirizzamento dinamico, S-ARP richiede l'implementazione di un S-DHCP personalizzato. Il server S-DHCP comunica con l'AKD per verificare l'autorizzazione di un host e per aggiornare dinamicamente il binding tra la chiave pubblica dell'host e l'indirizzo IP che gli viene assegnato. Questo processo garantisce che, anche in ambienti dinamici, la corrispondenza IP-Chiave sia sempre autenticata dall'autorità fidata.

L'adozione di S-ARP, pur offrendo integrità e autenticità a livello crittografico, richiede un'infrastruttura centralizzata dedicata e una modifica al protocollo e al software di gestione presente sugli host. Risulta quindi difficile da implementare in reti con elevata eterogeneità dei dispositivi, dove l'integrazione del software viene meno. Inoltre, nelle realtà in cui le reti vengono messe a disposizione di un bacino di utenza libero di accedervi, come le reti pubbliche degli aeroporti o simili, si presenta un fenomeno di alta volatilità dei nodi; ciò comporta l'impossibilità di garantire la presenza di tale protocollo di sicurezza su ogni dispositivo partecipe alla rete, rendendo inutilizzabile tale soluzione.

Si vedrà come le difficoltà di compatibilità e l'inefficacia nelle reti ad alto ricambio di host si ripresentano spesso nelle soluzioni non centralizzate.

2.1.2 ArpON

ArpON è una soluzione Open Source ideata e sviluppata da Andrea Di Pasquale per ridurre gli effetti negativi di un attacco ARP Cache Poisoning.[5]

ArpON è un daemon che opera nello spazio utente del sistema operativo, in parallelo con il protocollo ARP residente nel kernel. L'obiettivo primario di ArpON non è la prevenzione assoluta dell'attacco, quanto la mitigazione rapida degli effetti dell'avvelenamento della cache. Quando si verifica il Cache Poisoning, ArpON interviene per ripristinare la cache ARP del sistema locale a uno stato consistente nel più breve tempo possibile, rendendo l'attacco Man-in-the-Middle praticamente inefficace.

ArpON è strutturato in tre moduli distinti: SARPI, DARPI e HARPI, ciascuno dei quali affronta un diverso scenario di indirizzamento di rete. Il modulo SARPI (Static ARP Inspection) è dedicato agli ambienti in cui gli indirizzi IP sono assegnati in modo statico e persistente. In questo contesto, SARPI mantiene una propria cache di mapping $\langle$IP, MAC$\rangle$ considerati affidabili dalla sincronizzazione periodica e viene forzata nella cache ARP del sistema operativo.

Il modulo DARPI (Dynamic ARP Inspection) affronta lo scenario più comune delle reti dinamiche, in cui gli indirizzi sono assegnati tramite DHCP. Poiché in questo ambiente non esiste una lista di binding affidabili pre-configurata, DARPI aggiunge una nozione di stato al protocollo ARP, tradizionalmente stateless. Questa funzionalità è ottenuta tracciando tutti i messaggi ARP Request in uscita in una cache interna temporanea. Un ARP Reply in entrata è considerato valido e affidabile solo se corrisponde a una richiesta pendente

recentemente inviata dall'host locale. Se un ARP Reply non è sollecitato o non corrisponde a una richiesta registrata, esso viene classificato come anomalo e la sua associazione viene considerata maligna e scartata (Deny Policy), prevenendo l'avvelenamento della cache locale.

Il modulo HARPI (Hybrid ARP Inspection) combina infine le funzionalità di SARPI e DARPI, gestendo in modo diverso le associazioni statiche e quelle dinamiche all'interno della stessa rete.

Questa soluzione non è esente da difetti, in quanto presenta le stesse difficoltà di implementazione della precedente in reti eterogenee e con un alto ricambio degli host. Inoltre, è necessario disabilitare a livello di sistema operativo l'accettazione delle gratuitous Reply, ovvero le ARP Reply senza una ARP Request precedente.

Queste modifiche al kernel sono invasive perché intervengono direttamente sul comportamento di rete del sistema operativo; possono essere pericolose se non gestite correttamente, causando potenziali interruzioni della rete. La stessa documentazione di ArpON avverte sui rischi di queste modifiche in caso di terminazione impropria:

"Remember to restore the values of the arp_ignore and arp_accept kernel parameters of the specified network interface (the default values are 0 for both), if you have terminated the ArpON daemon with other signals, e.g. kill signal (SIGKILL)."

2.1.3 Arpwatch

Arpwatch [6] è uno strumento open source che fornisce una soluzione di monitoraggio passivo per la rilevazione di attacchi ARP Spoofing all'interno di una rete locale. A differenza di soluzioni attive come ArpON, Arpwatch opera come daemon di semplice ispezione e logging, intercettando i messaggi ARP nel traffico di rete e confrontandoli con un database locale di associazioni IP-MAC note.

Il funzionamento di Arpwatch attua una strategia di vigilanza continua sul traffico ARP locale. Il daemon intercetta ogni pacchetto ARP che attraversa l'interfaccia di rete, osservando in particolare le risposte di tipo ARP Reply. Queste risposte includono un'associazione tra un indirizzo IP e un indirizzo MAC che rappresenta il nodo mittente.

Per ogni ARP Reply ricevuta, Arpwatch esegue un confronto con la sua cache interna, un database locale storico contenente i precedenti binding IP-MAC considerati affidabili. Quando l'associazione indicata nella risposta corrisponde a quanto già registrato, il pacchetto viene ignorato in quanto conforme al normale comportamento di rete.

Tuttavia, se Arpwatch rileva un cambiamento inatteso, ovvero un indirizzo IP che si mappa su un indirizzo MAC differente da quello presente nella cache, interpreta tale evento come potenzialmente anomalo e sospetto. A questo punto viene generato un allarme, che può concretizzarsi nell'invio di una notifica via email agli amministratori, nella scrittura di un messaggio sul sistema di log o nell'attivazione di script di reazione personalizzati, consentendo un intervento tempestivo.

In sintesi, Arpwatch monitora in modo passivo il traffico ARP, tenendo traccia dei binding IP-MAC e segnalando le modifiche potenzialmente indicative di un ARP spoofing o di un attacco Man-in-the-Middle. Questa metodologia permette di rilevare efficacemente comportamenti anomali, pur non fornendo un meccanismo di blocco attivo degli attacchi.

Arpwatch presenta significativi limiti che ne riducono l'efficacia come soluzione di sicurezza attiva. In primo luogo, è una soluzione puramente passiva: non previene gli attacchi, ma si limita a rilevarli e a segnalarli. In secondo luogo, in ambienti dinamici come le reti DHCP, dove gli indirizzi IP vengono frequentemente riassegnati, Arpwatch genera numerosi falsi positivi, creando un carico amministrativo considerevole. Inoltre, il suo funzionamento dipende esclusivamente dalla memorizzazione di binding già noti, il che la rende vulnerabile a sofisticati attacchi che non modificano le associazioni già conosciute.

La sua implementazione rimane semplice e leggera, richiedendo interventi minimali sulla rete; tuttavia, questo vantaggio è compensato dall'assenza di meccanismi di risposta automatica e dall'incapacità di prevenire attivamente i danni derivanti da ARP Poisoning. Arpwatch è quindi più adatto a contesti in cui il monitoraggio e l'analisi forense post-attacco sono prioritari rispetto alla prevenzione in tempo reale.

2.1.4 Static ARP Table

Le tabelle ARP statiche rappresentano una soluzione semplice e storicamente riconosciuta per contrastare gli attacchi di ARP Poisoning. Questa tecnica prevede la configurazione manuale e preventiva della mappatura fissa tra indirizzi IP e indirizzi MAC su ciascun dispositivo della rete, impedendo così la possibilità di alterare queste associazioni tramite pacchetti ARP falsificati.

Dal punto di vista tecnico, l'implementazione di tabelle statiche significa che ogni nodo tiene in memoria una lista esplicita di binding IP-MAC, e rifiuta aggiornamenti dinamici o modifiche non autorizzate. In questo modo, i pacchetti ARP Reply che presentano associazioni differenti rispetto a quelle fissate manualmente vengono ignorati, proteggendo efficacemente contro i tentativi di avvelenamento della cache ARP.

Tuttavia, questa soluzione presenta importanti limiti pratici che ne riducono notevolmente l'applicabilità nelle reti moderne:

- **Scarsa scalabilità**: la configurazione deve essere eseguita manualmente su ciascun host della rete, comportando un elevato overhead amministrativo e la possibilità di errori.
- **Incompatibilità con reti dinamiche**: in ambienti dove gli indirizzi IP vengono assegnati dinamicamente, come le reti DHCP, la gestione delle tabelle statiche diventa impraticabile e può causare problemi di connettività.

- **Rigidità e complessità di manutenzione**: ogni variazione nella rete richiede una modifica manuale e coordinata delle tabelle, rendendo difficile la gestione in presenza di host mobili o reti in rapida evoluzione.

Per questi motivi, l'uso delle tabelle ARP statiche è generalmente riservato a reti di piccole dimensioni o a contesti di laboratorio e test, dove le condizioni di rete sono controllate e stabili. Nelle implementazioni di sicurezza più avanzate e scalabili, si preferiscono soluzioni dinamiche e automatizzate che possano adattarsi alla complessità e variabilità delle reti reali.

2.1.5 Gratuitous ARP Filtering

Il filtraggio dei Gratuitous ARP rappresenta una tecnica di rafforzamento a livello di sistema operativo, utile per mitigare alcuni tipi di attacchi ARP Spoofing. I pacchetti Gratuitous ARP sono particolari messaggi ARP in cui un host annuncia il proprio indirizzo IP associato senza che sia stata precedentemente inviata una richiesta specifica. Sebbene questi messaggi servano normalmente a segnalare aggiornamenti o a prevenire conflitti IP, possono essere utilizzati in modo malevolo per manipolare le tabelle ARP degli host della rete.

Il filtraggio dei Gratuitous ARP consiste nell'abilitare, tramite parametri del kernel dei sistemi operativi come Linux, regole di controllo che limitano o bloccano l'accettazione di questi messaggi quando sono considerati non necessari o sospetti. Questo aiuta a ridurre la superficie di attacco indiretta associata a pacchetti ARP inattesi o falsificati, diminuendo la probabilità di accettare risposte malevoli o di inserire binding non autorizzati nella cache ARP.

Dal punto di vista tecnico, il filtro può essere configurato agendo sui parametri `arp_ignore` e `arp_accept` del kernel, impostandoli a valori che obblighino il sistema a rispondere o accettare pacchetti ARP solo in circostanze definite, ad esempio solo quando sono associati all'interfaccia corretta o alla configurazione IP attiva. Ciò migliora la robustezza del sistema contro alcune varianti di ARP Spoofing senza intervenire in modo massiccio sul traffico o sull'architettura di rete.

Tuttavia, questa soluzione presenta alcune limitazioni [7]:

- **Potenziali incompatibilità**: modifiche aggressive alle impostazioni di accettazione ARP possono causare problemi di connettività, in particolare in reti con configurazioni IP complesse o cambi frequenti di indirizzo.
- **Falsa sensazione di sicurezza**: pur migliorando la resistenza contro gli attacchi basati su Gratuitous ARP, questa tecnica non protegge da tutte le varianti di ARP Poisoning, né da attacchi più sofisticati basati su risposte ARP regolari ma malevole.

- **Richiede una gestione attenta**: la scelta e il tuning dei parametri kernel devono essere eseguiti con conoscenza e cautela per evitare effetti collaterali indesiderati.

In definitiva, il filtraggio dei Gratuitous ARP costituisce un buon complemento di sicurezza a livello di sistema operativo, ma solo come parte di una strategia più ampia che include tecniche di monitoraggio e prevenzione più robuste per contrastare efficacemente gli attacchi ARP Spoofing.

2.2 Soluzioni Centralizzate

2.2.1 SNORT - Intrusion Detection System (IDS)

Un Intrusion Detection System (IDS) è uno strumento progettato per monitorare il traffico di rete o le attività di sistema al fine di individuare comportamenti sospetti, anomalie o eventuali attacchi informatici. Gli IDS possono essere classificati in diverse tipologie, tra cui quelli basati sul riconoscimento di firme (o *signature*) e quelli basati su comportamenti anomali.

SNORT [8] è uno degli IDS open source più diffusi e affermati, capace di effettuare un'analisi approfondita dei pacchetti di rete in tempo reale. Attraverso un sistema di regole configurabili, SNORT è in grado di riconoscere i pattern corrispondenti a molteplici tipologie di attacco, incluse le varie forme di ARP Poisoning.

Per quanto riguarda specificamente gli attacchi di ARP spoofing, SNORT monitora i pacchetti ARP Reply e può generare alert qualora rilevi risposte sospette, in particolare quando un ARP Reply non è preceduto da una richiesta ARP corrispondente o quando si individua una variazione anomala nelle associazioni IP-MAC. Questo consente agli amministratori di rete di essere tempestivamente informati di potenziali compromissioni o tentativi di Man-in-the-Middle.

Nonostante l'efficacia nel rilevamento, SNORT è uno strumento limitato in questo contesto: non dispone infatti di meccanismi integrati per bloccare o mitigare attivamente gli attacchi segnalati, limitandosi invece a sollevare notifiche, le quali devono essere interpretate e gestite manualmente. Inoltre, la capacità di SNORT di identificare gli attacchi di ARP spoofing non è basata su una certezza assoluta, ma su un modello di allerta probabilistico, che si traduce in segnalazioni di eventi sospetti piuttosto che in conferme definitive di compromissione.

Questa natura comporta che molti degli allarmi generati possano risultare falsi positivi, soprattutto in ambienti di rete complessi o dinamici, dove i cambiamenti naturali delle associazioni IP-MAC e il traffico legittimo possono attivare i meccanismi di rilevamento. L'inevitabile presenza di questi falsi positivi impone un oneroso lavoro di filtraggio e analisi postincidente da parte degli amministratori di rete, rallentando la capacità di risposta e riducendo l'efficacia pratica dello strumento nel contrastare attivamente gli attacchi.

In scenari operativi in cui la tempestività e la precisione sono fondamentali, questa caratteristica di SNORT costituisce una limitazione critica, sottolineando l'importanza di integrare sistemi IDS passivi con soluzioni di difesa attiva e prevenzione più robuste. In conclusione, SNORT rappresenta un sistema di supporto per il monitoraggio e la diagnosi degli attacchi ARP valido soprattutto ai fini didattici [9], ma per garantire un'efficace protezione della rete, è necessario integrarlo con sistemi di monitoraggio e prevenzione più sofisticati.

2.2.2 Port Security e Network Access Control (NAC)

Le tecniche di Port Security rappresentano un approccio spesso adottato a livello di infrastruttura di rete, in particolare sugli Switch Managed [1], per prevenire attacchi come l'ARP Spoofing, impedendo l'accesso di dispositivi non autorizzati o la falsificazione degli indirizzi MAC.

Le tecniche di Port Security e di Network Access Control (NAC) sono due metodologie complementari utilizzate a livello di infrastruttura di rete, principalmente negli switch gestiti, per prevenire accessi non autorizzati e attacchi come l'ARP Spoofing.

La Port Security consente di associare uno o più indirizzi MAC specifici a una porta fisica dello switch, garantendo che solo i dispositivi con tali indirizzi possano comunicare attraverso quella porta. Se un dispositivo tenta di utilizzare un indirizzo MAC differente o non autorizzato, l'accesso viene bloccato o limitato in conformità alle policy configurate. Questo meccanismo riduce drasticamente la possibilità per un attaccante di compromettere la rete mediante la falsificazione degli indirizzi MAC e, quindi, di realizzare attacchi di MAC spoofing.

Complementare a questo approccio è lo standard IEEE 802.1X [10], che introduce un protocollo di autenticazione basato su un modello client-server. In questo schema, un dispositivo endpoint (supplicant) deve autenticarsi attraverso uno switch o access point abilitato (authenticator) presso un server di autenticazione prima di ottenere l'accesso alla rete. Solo gli endpoint che superano questa autentificazione possono comunicare nel dominio di rete, impedendo così l'ingresso di dispositivi non autorizzati o malevoli.

L'adozione congiunta di Port Security e 802.1X contribuisce a ridurre significativamente la superficie di attacco della rete: limitando l'accesso fisico e logico esclusivamente a dispositivi autenticati e autorizzati, si prevengono efficacemente tentativi di ARP Spoofing provenienti da nodi non riconosciuti, rafforzando la sicurezza complessiva dell'infrastruttura di rete.

Tuttavia, questi meccanismi presentano anch'essi alcune limitazioni:

[1] Vengono definiti "managed" quegli Switch che svolgono le normali funzionalità di livello 2 ma, in aggiunta, forniscono le funzionalità di routing e di gestione delle regole di instradamento, solitamente implementate solo dai router.

- **Dipendenza dall'hardware**: Port Security e NAC richiedono Switch Managed e infrastrutture di rete compatibili, aumentando i costi e la complessità dell'implementazione.
- **Gestione delle eccezioni**: In ambienti dinamici o con dispositivi mobili, la configurazione statica degli indirizzi MAC o il processo di autenticazione possono generare problemi di connettività o richiedere frequenti interventi amministrativi.
- **Non sempre efficaci contro attacchi interni**: Se un dispositivo malevolo è già autorizzato o autenticato, queste tecniche non prevengono gli attacchi provenienti da tale nodo.

In definitiva, Port Security e Network Access Control rappresentano strategie robuste se integrate in un'articolata architettura di sicurezza di rete, ma non sono soluzioni complete né sufficienti da sole a prevenire ogni forma di ARP poisoning, rendendo necessaria la loro integrazione con meccanismi di rilevamento e prevenzione più sofisticati.

È importante sottolineare un limite cruciale: queste tecniche risultano inefficaci contro attacchi provenienti da dispositivi malevoli già autorizzati o autenticati [11]. Nel momento in cui un nodo compromesso riesce a ottenere o ha già ottenuto l'accesso legittimo alla rete tramite l'autenticazione o la configurazione di Port Security, le difese a livello di accesso diventano praticamente inutili. Un attaccante in possesso di credenziali valide o di un dispositivo "manomesso" può così eseguire liberamente azioni dannose, incluso l'ARP Spoofing, aggirando questi sistemi di controllo.

Per questo motivo, sebbene Port Security e NAC rappresentino importanti barriere iniziali, è indispensabile adottare sistemi complementari di ispezione del traffico e verifica dinamica degli indirizzi IP-MAC, capaci di rilevare e mitigare attacchi provenienti da entità già presenti nella rete, garantendo così un livello di protezione più completo e resiliente.

2.2.3 Stateful IP-MAC Binding Monitoring

Un approccio centrale per la prevenzione degli attacchi di ARP Poisoning è la sorveglianza dinamica e contestuale degli abbinamenti fra indirizzi IP e MAC, nota come *stateful IP-MAC binding monitoring*. Questa tecnica consiste nel monitorare in tempo reale le associazioni IP-MAC osservate nel traffico di rete, rilevando incoerenze o tentativi di modifica non autorizzati.

A livello pratico, dispositivi di rete avanzati, come Switch Managed o controller software-defined network (SDN)[2], tengono traccia dello stato corrente delle corrispondenze

[2]Vengono definite Reti "DNS" quelle infrastrutture che disaccoppiano il livello dei dispositivi di rete dal livello del software di gestione della rete. Questo permette di gestire e programmare l'intera rete come se fosse un unico sistema, configurare la rete dinamicamente tramite software e offrire una visione globale e programmabile dell'infrastruttura

IP-MAC, verificando ogni messaggio ARP in ingresso per validarne la correttezza rispetto ai binding già noti, aggiornati dinamicamente in locale tramite osservazione o mediante integrazione con i lease DHCP. Se viene rilevato un pacchetto ARP che introduce una modifica sospetta o non autorizzata, il dispositivo può bloccarlo o generare un allarme immediato.

Questa soluzione combina i benefici del controllo centralizzato con la capacità di adattarsi ai cambiamenti di rete tipici di ambienti dinamici, garantendo un controllo più granulare e tempestivo rispetto a sistemi passivi o statici.

Tuttavia, presenta alcune criticità significative:

- **Complessità di gestione**: richiede infrastrutture di rete avanzate e sistemi centralizzati capaci di integrare dati provenienti da più fonti (ad esempio DHCP, ARP, switch forwarding tables).
- **Limitazioni in ambienti eterogenei**: in reti con dispositivi disparati o senza supporto per queste funzioni, l'efficacia della contromisura viene compromessa.
- **Potenziali falsi positivi**: variazioni temporanee, come i movimenti degli host o le riassegnazioni rapide di indirizzi IP, possono innescare falsi allarmi.
- **Non previene gli attacchi interni da sistemi compromessi**: se un dispositivo compromesso è già autorizzato, l'attacco può comunque propagarsi.

In sintesi, lo stateful IP-MAC binding monitoring rappresenta un netto miglioramento rispetto alle soluzioni passive o statiche, fornendo un controllo dinamico e più affidabile delle associazioni ARP. Tuttavia, resta una soluzione che richiede la modifica degli apparati di rete e che non si integrerebbea pienamente nell'infrastruttura preesistente, comportando una soluzione ad hoc che difficilmente possa essere mantenuta e replicata in scalabilità.

2.2.4 Dynamic ARP Inspection

Dynamic ARP Inspection (DAI) [12] è una funzionalità di sicurezza integrata in molti switch di fascia enterprise (Tabella 2.2.4), progettata per prevenire attivamente gli attacchi di ARP Spoofing e garantire l'integrità delle tabelle ARP nelle reti locali.[12] DAI agisce monitorando e filtrando i pacchetti ARP ricevuti sulle interfacce di rete, verificando che ciascuno di essi sia conforme a binding validi basati su informazioni affidabili fornite, ad esempio, dal DHCP Snooping.

Il DHCP Snooping è un meccanismo di sicurezza complementare che filtra e controlla i messaggi DHCP in una rete, solitamente eseguito direttamente sugli switch di accesso o sugli switch di livello distribuzione. Durante questo processo, lo switch crea e mantiene una tabella di binding dinamica e affidabile che associa gli indirizzi IP assegnati agli indirizzi MAC dei dispositivi client, insieme alle informazioni riguardanti le porte dello switch e la

Fornitore	**Linea Prodotti / Piattaforme che supportano DAI**	**Descrizione**
Cisco	Catalyst series (e.g., 9200, 9300, 9400, 9500, 3850, 4500, 6500), Nexus (limited), Meraki MS series	Il produttore a cui si attribuisce l'invenzione della Dynamic ARP Inpection
Juniper Networks	EX Series (e.g., EX2300, EX3400, EX4300, EX4600, QFX with Junos ELS)	Viene sempre riconosciuta con Dynamic ARP Inspection
HPE Aruba	Aruba CX series (6200, 6300, 6400, 8100, 8300, 8400, 10000), 2930F/3810/5400R	CX: Dynamic ARP Inspection; Older ProVision: Dynamic ARP Protection (simile)
Fortinet	FortiSwitch series (e.g., FS-108E, FS-448E, FS-1000+ models)	Dynamic ARP Inspection; integrata in FortiGate
Huawei	CloudEngine S-series, NE-series, S5700, S6700, CloudEngine 6800/8800	Supportata come DAI o ARP strict inspection
NETGEAR	M4300, M4350, some Insight-managed models	Disponibile sui dispositivi managed di alta gamma

Tabella 1: Fornitori di switch enterprise che supportano Dynamic ARP Inspection (DAI)

VLAN da cui provengono. Questa tabella, nota come binding table, risiede nella memoria dello switch stesso e funge da riferimento sicuro per DAI nella validazione dei pacchetti ARP che transitano attraverso le sue interfacce.

DAI garantisce che solo i pacchetti ARP con associazioni IP-MAC, accertate e catalogate nella tabella definita in precedenza, vengano permessi, mentre i pacchetti sospetti o non autorizzati vengano bloccati immediatamente. Questa ispezione dinamica è basata su binding noti e consente di identificare e isolare tentativi di Man-in-the-Middle causati da manipolazioni fraudolente delle tabelle ARP, migliorando la sicurezza a livello 2.

Funzionamento di Dynamic ARP Inspection

Dynamic ARP Inspection (DAI) agisce in tempo reale controllando ogni pacchetto ARP in ingresso sulle porte configurate, avvalendosi di una tabella di binding affidabile, tipicamente creata dalla funzione di DHCP Snooping.

La tabella delle associazioni IP-MAC è aggiornata dinamicamente ogni volta che un host ottiene un indirizzo IP tramite il protocollo DHCP: ogni lease DHCP aggiunge alla tabella una riga con l'indirizzo MAC del client, l'IP assegnato, la relativa interfaccia e il tempo di validità. La tabella funge da riferimento autorevole per tutte le successive operazioni di ispezione ARP.

Le porte dello switch devono essere designate come *trusted* o *untrusted*:

- Porte **trusted** (uplink, router, server DHCP): i pacchetti ARP non sono soggetti a verifica DAI, riducendo impatti su funzioni critiche.
- Porte **untrusted** (access toward end-hosts): DAI verifica rigorosamente ogni pacchetto.

Questa distinzione è fondamentale per minimizzare i falsi positivi e massimizzare la sicurezza perimetrale. La configurazione e la gestione delle interfacce devono essere mantenute aggiornate e coerenti con la topologia della rete.

Al ricevimento di un pacchetto ARP su una porta configurata per DAI, lo switch esegue una serie di verifiche:

- **Binding Validation**: il pacchetto ARP deve presentare un'associazione IP-MAC esattamente corrispondente a una voce valida in tabella. Se l'indirizzo IP o MAC non corrisponde, il pacchetto viene scartato e segnalato.
- **Rate Limiting**: DAI applica limitazioni sul numero di pacchetti ARP accettati in un intervallo di tempo, mitigando attacchi di flood (ARP DoS).
- **Ingress Port Check**: solo i pacchetti provenienti dalle interfacce non fidate (`untru -sted`) sono sottoposti a controllo. Sulle interfacce *trusted* (tipicamente collegate a router, server DHCP, collegamenti du trunk), il traffico ARP passa senza verifica, riducendo i falsi positivi.
- **Policy Enforcement**: DAI permette di configurare politiche avanzate, come la gestione specifica dei Gratuitous ARP (blocco, passaggio condizionato), l'integrazione con logging di sicurezza, e la personalizzazione del livello di allerta o drop.
- **Logging**: Ogni evento di drop o violazione è registrato nei log dello switch e può essere attivata una notifica all'amministratore di rete.

Alcuni dei passaggi chiave possono essere riassunti così:

1. L'host ottiene l'IP tramite DHCP
2. DHCP Snooping aggiunge il binding IP-MAC alla tabella.
3. Un host invia pacchetti ARP sulla VLAN protetta / rete monitorata
4. lo switch verifica l'associazione IP-MAC.
   (a) Se il binding è valido, il pacchetto ARP viene inoltrato.
   (b) Se il binding non è valido, procede al `DROP` del pacchetto ARP e `log` dell'attività sospetta.
5. Rate limiting: accertata la validità di inoltro, se si supera la soglia configurata di pacchetti per secondo (pps), il DAI memorizza un log a riguardo e blocca la ricezione.

Punti di Forza

Grazie a questi meccanismi, DAI previene efficacemente attacchi Man-in-the-Middle (MITM) tramite ARP spoofing, poiché ogni tentativo di invio di ARP Reply falsificati da parte di dispositivi non autorizzati o con binding non validi viene bloccato prima di diffondersi nella rete. Inoltre, la sincronizzazione dinamica con DHCP Snooping garantisce che anche le reti dinamiche e in rapido cambiamento siano protette senza la necessità di configurazioni statiche onerose.

DAI consente un'ispezione ARP capillare, un adattamento dinamico alle variazioni di rete, e la mitigazione degli attacchi DoS basati su ARP, risultando compatibile con ambienti enterprise complessi grazie all'integrazione con altre funzionalità Cisco e alla centralizzazione della logica di sicurezza. La sua granularità, configurabilità e capacità di risposta immediata lo rendono una soluzione di riferimento ben oltre gli strumenti passivi e i controlli statici.

Tra i principali vantaggi di DAI si annoverano:

- **Protezione proattiva**: DAI interviene in tempo reale bloccando i pacchetti ARP sospetti, prevenendo la compromissione delle tabelle ARP e l'efficacia degli attacchi.
- **Integrazione con meccanismi esistenti**: sfruttando le informazioni di DHCP Snooping e Port Security, DAI utilizza una base dati affidabile e aggiornata per la validazione, assicurando coerenza e riduzione di falsi positivi.
- **Scalabilità**: grazie all'automazione nella gestione dei binding e alla configurazione dinamica delle interfacce trusted e untrusted, DAI può essere efficacemente implementato anche in reti di grandi dimensioni e in ambienti enterprise complessi.
- **Riduzione del carico amministrativo**: eliminando la necessità di configurazioni manuali statiche e grazie a un report di log più controllato, DAI semplifica la manutenzione e riduce il rischio di errori umani.
- **Compatibilità con ambienti dinamici**: DAI si adatta automaticamente ai cambi frequenti di indirizzi IP attraverso la collaborazione con DHCP Snooping, mantenendo la protezione senza interruzioni.
- **Supporto agli standard di rete**: essendo una soluzione standardizzata da Cisco ed adottata dai principali fornitori (Tabella 2.2.4), DAI beneficia di un ampio supporto e di una documentazione consolidata.

Questi punti di forza rendono Dynamic ARP Inspection la soluzione di riferimento nei contesti aziendali per la protezione contro gli attacchi ARP Poisoning e l'efficace salvaguardia delle infrastrutture di rete a livello di layer 2.

2.3 Conclusioni sulle Soluzioni Presenti in Letteratura

Le soluzioni di sicurezza di tipo *host-based*, come ArpON e Arpwatch analizzate nella precedente sezione, offrono un approccio distribuito alla protezione contro gli attacchi di ARP spoofing, installando un software di controllo direttamente su ciascun dispositivo della rete. Questa tecnica, pur garantendo un controllo granulare e localizzato, presenta tuttavia diversi svantaggi.

In primo luogo, la presenzae di daemon di sicurezza su tutti i nodi comporta un aumento proporzionale della superficie di attacco complessiva del sistema, in quanto ogni dispositivo diventa un potenziale punto vulnerabile che può essere compromesso o disabilitato. Come formalizzato in *un'Attack Surface Metric* da Manadhata e Wing [13], una maggiore superficie di attacco implica un rischio maggiore di successo per gli aggressori.

Inoltre, le soluzioni host-based spesso richiedono una complessa gestione distribuita, che può risultare onerosa in reti di grandi dimensioni o eterogenee, con elevati costi in termini di manutenzione, aggiornamento e sincronizzazione. Alcuni strumenti, come Arpwatch, si limitano a monitorare e segnalare anomalie senza fornire un meccanismo di prevenzione attiva, lasciando ampi margini di intervento manuale e reattivo.

Infine, le soluzioni host-based risultano tipicamente meno efficaci nei confronti di attacchi che coinvolgono nodi già compromessi o nella gestione di situazioni di mobilità e dinamicità degli indirizzi IP nella rete, condizioni sempre più frequenti negli ambienti moderni.

Per questi motivi, benché contribuiscano ad aumentare lo stato di allerta e la consapevolezza sugli attacchi, questo tipo di soluzioni presenta limiti evidenti che ne riducono l'efficacia e l'adattabilità in contesti complessi. Questo conduce naturalmente a preferire soluzioni con un'architettura centralizzata, capaci di garantire una riduzione della superficie di attacco complessiva attraverso una gestione più coesa, integrata e controllata della sicurezza di rete.

Passiamo ora ad analizzare le soluzioni centralizzate per la difesa contro gli attacchi ARP, con l'obiettivo di individuare quella più efficace e praticabile. Per questa analisi, si farà riferimento alla letteratura e in particolare allo studio *An Analysis on the Schemes for Detecting and Preventing ARP Cache Poisoning Attacks* [14], che offre una visione approfondita e critica di vari schemi di protezione esistenti nel panorama attuale.

Considereremo inoltre un tema non ancora trattato, ovvero i costi di questa soluzione.

Nel capitolo 4, "Comparison of Existing Schemes" [14], gli autori sottolineano che: *"Out of all these schemes, we believe that the Dynamic ARP Inspection (DAI) mechanism [8] is the best because it is non-intrusive, requires no changes on the network hosts, and does not slow down ARP. Unfortunately, the high costs of the switches with DAI available makes this solution prohibitive for many companies".* Questa affermazione mette in evidenza il ruolo di DAI come soluzione di riferimento, in virtù della sua efficacia e della sua non invasività. Viene anche presentato il principale svantaggio di questa soluzione, ovvero il suo contesto

di appartenenza: dispositivi enterprise con costi tendenzialmente troppo elevati per molte realtà.

Tale svantaggio ci fornisce uno spunto importante, già preso in considerazione nel capitolo 3, "Schemes for Securing ARP" [14]. Nella Sezione 3.3, "Preventing or Blocking ARP Attacks", gli autori osservano: *"Some high-end Cisco switches have a new feature called Dynamic ARP Inspection. One main disadvantage of this solution is the high cost of switches that have this feature available"*. Questa asserzione indics DAI come una tecnologia di fascia alta, riservata a reti con budget proporzionalmente adeguati.

Il principale aspetto negativo porta, infine, ad una conclusione chiara. Nel capitolo 6, "Future Work" [14], discutendo i possibili sviluppi futuri, gli autori anticipano: *"We are working in a new scheme to secure ARP that complies with the requirements listed in Section 5. Our pre-liminary design adapts some ideas of Cisco's Dynamic ARP Inspection [8] in such a way that does not require the use of costly switches, as DAI does"*. Questo conferma DAI come soluzione ideale e suggerisce la possibilità di una sua realizzazione in un contesto più fruibile. Idealmente, senza il contesto enterprise di alta gamma che la circonda, gli autori lasciano intendere che DAI potrebbe essere una funzionalità integrabile anche in dispositivi economici.

2.4 L'idea di una Dynamic ARP Inspection più fruibile

Secondo quanto discusso nella sezione precedente, la funzionalità "Dynamic ARP Inspection" merita un'implementazione meno costosa ma senza perdite di efficacia sia in termini di sicurezza che di prestazioni. L'oggetto di questo elaborato di tesi è la realizzazione di questa idea: un DAI sviluppato con l'obiettivo di superare le limitazioni dell'ambito enterprise in cui è stato confinato fino ad ora. Si tratta di una funzione integrabile in un ampia gamma di dispositivi di rete, distribuendo questo potente controllo di socurezza per il bene di tutti gli utenti, ed aumentando la sicurezza di molte realtà aziendali e private,

La scelta della modalità di sviluppo ricade sul paradigma fruibile per definizione, ovvero l'Open Source. Questo paradigma verrà approfondito nel capitolo seguente per una comprensione ottimale.

In sintesi, l'elaborato si pone come obiettivo quello di sfidare l'attuale frontiera tecnologica riguardo alla sicurezza negli attacchi di ARP Cache Poisoning presentando uno spunto di riposizionamento delle soluzioni attuali. Ciò avviene tramite la realizzazione di un software di Dynamic ARP Inspection "leggero" che possa essere considerato il punto di partenza per un'integrazione sistematica in dispositivi di rete di natura completamente eterogenea, senza limiti di natura implementativa e/o economica, restituendo agli utenti la sicurezza dei propri apparati di rete.

Capitolo 3

Open Source

La nostra implementazione del daemon di ispezione ARP si innesta in un contesto tecnologico in cui la scelta del paradigma di sviluppo Open Source non costituisce una mera preferenza operativa, ma una decisione strategica che riflette i principi di trasparenza, affidabilità e collaborazione fondamentali per la realizzazione di applicazioni critiche per la sicurezza [15]. Il Software Open Source (OSS) si definisce per l'accesso illimitato al codice sorgente, conferendo agli utenti le libertà di studiare, modificare e ridistribuire il software stesso, come stabilito dalle licenze aperte.

Questa filosofia è in netta contrapposizione con il modello proprietario (Closed Source), dove il codice rimane celato, limitando l'intervento e l'analisi da parte di terzi. La scelta di fondare il progetto su un framework e su componenti Open Source assicura l'aderenza a standard rigorosi di interoperabilità, promuovendo al contempo una portabilità essenziale per un'applicazione destinata a operare a stretto contatto con il kernel del sistema operativo.

Nel campo della sicurezza informatica, in particolare, la trasparenza del codice sorgente consente un controllo continuo e collettivo [15], un meccanismo che, secondo le stesse parole di Linus Torvalds [16], accelera l'identificazione e la correzione delle vulnerabilità in modo più efficiente rispetto ai cicli di sviluppo chiusi. Pertanto, l'adozione dell'Open Source in questa tesi supporta l'integrazione di componenti software di rete noti per la loro robustezza e affidabilità nel tempo, allineandosi a una visione di sicurezza partecipata, dove la solidità del sistema è garantita dalla revisione e dal contributo di una comunità estesa, trasformando l'infrastruttura di base in un bene comune digitale.

3.1 Introduzione e Definizione

La genesi storica dell'Open Source affonda le radici negli anni '80, in un periodo in cui la nascente industria del software proprietario iniziava a limitare l'accesso al codice sorgente, un'azione che interruppe la tradizionale prassi accademica e di ricerca di condivisione del codice. L'impulso decisivo per la formalizzazione di un modello alternativo fu dato da

Richard Stallman con la fondazione del Progetto GNU nel 1983 e la successiva creazione della General Public License (GPL). Sebbene il movimento iniziale fosse primariamente filosofico e denominato Free Software, il termine Open Source fu coniato solo nel 1998, in seguito alla decisione di Netscape di rilasciare il codice sorgente del suo browser. Questo cambio terminologico fu strategico: l'obiettivo era enfatizzare il modello di sviluppo pratico e collaborativo (il processo di apertura) piuttosto che la connotazione etica e politica della "libertà", rendendo il concetto più appetibile e accettabile per il mondo business e enterprise.

La definizione di Open Source, stabilita formalmente dalla Open Source Initiative (OSI), non si limita alla mera disponibilità del codice sorgente. Essa si basa su un insieme rigoroso di dieci criteri che devono essere soddisfatti da una licenza affinché il software possa essere definito "Open Source" a tutti gli effetti. Tali criteri vanno oltre la semplice gratuità d'uso, assicurando che la licenza garantisca la libera ridistribuzione, l'inclusione del codice sorgente e la possibilità di lavori derivati. Crucialmente, la definizione OSI stabilisce anche clausole di non discriminazione contro persone, gruppi o campi di impresa, garantendo la piena fruibilità e il potenziale contributo da parte di qualsiasi soggetto. L'autorità dell'OSI è fondamentale, in quanto essa funge da garante che il software etichettato come Open Source rispetti i principi essenziali di trasparenza e collaborazione comunitaria.

In questo contesto di sviluppo e licenza, è cruciale operare una distinzione rigorosa tra il Software Open Source e altre categorie di software che, pur essendo spesso distribuite senza oneri economici, non ne condividono la filosofia fondamentale né i diritti legali associati al codice sorgente. Il Freeware, ad esempio, costituisce un modello di distribuzione in cui il software è concesso gratuitamente all'utente finale; tuttavia, rimane un software intrinsecamente proprietario (Closed Source), con il codice sorgente mantenuto segreto.

L'assenza di costo non si traduce in libertà: l'utente è rigorosamente vincolato ai termini della licenza d'uso (EULA) e alle funzionalità predefinite dallo sviluppatore, impedendo lo studio, la modifica o la ridistribuzione. Similmente, il Public Domain (Dominio Pubblico) denota software privo di diritti d'autore (copyright) e quindi liberamente utilizzabile. Tuttavia, questa assenza di licenza, pur garantendo la massima libertà d'uso, non impone e non promuove i principi di governance collaborativa e tracciabilità delle modifiche tipici dell'Open Source.

Di conseguenza, solo l'OSS, attraverso il suo impianto legale e i criteri della definizione OSI, assicura che il codice sorgente sia non solo disponibile, ma che vengano garantiti i diritti di fork e di evoluzione autonoma, elementi essenziali per l'affidabilità a lungo termine e l'adattabilità critica in ambiti come la sicurezza informatica.

3.2 Differenze tra Open Source e Software Proprietario

Il contrasto tra il paradigma di sviluppo Open Source e il Software Proprietario è fondamentale per comprendere non solo le differenze di licenza, ma anche l'impatto sulla sicurezza, l'innovazione e la governance tecnologica. L'Open Source si manifesta come un

acceleratore di miglioramento e beneficio comune, grazie al suo modello intrinsecamente collaborativo e trasparente. La libera accessibilità al codice sorgente elimina le barriere all'innovazione, consentendo a ricercatori e sviluppatori di evolvere su fondamenta tecnologiche già collaudate.

Questa dinamica è alimentata da una peculiare convergenza di interessi: da un lato, esiste una componente di interesse individuale che motiva gli sviluppatori a contribuire, risolvendo problemi specifici e migliorando le proprie competenze. Dall'altro lato, ogni modifica, correzione di bug o miglioramento di funzionalità, anche se originato da un bisogno personale, viene immediatamente reso disponibile alla comunità, in un processo virtuoso in cui l'interesse del singolo si traduce automaticamente in beneficio collettivo, elevando l'affidabilità e la qualità dell'intero ecosistema.

Nel campo della sicurezza informatica, il vantaggio della trasparenza conferita dall'Open Source è insuperabile, trasformando il codice sorgente in un elemento chiave di fiducia e resilienza. Questo si fonda sul principio, spesso citato come Legge di Linus (Linus's Law), secondo cui "più occhi vedono, più facile è trovare l'errore". La disponibilità pubblica del codice garantisce che il software sia sottoposto a un audit continuo e un testing distribuito da parte di una comunità globale di ricercatori, specialisti e sviluppatori. Questo processo collaborativo accelera drasticamente l'identificazione e la correzione delle vulnerabilità, riducendo in modo significativo il "Window Of Exposure", la finestra temporale in cui una falla è sfruttabile.

Inoltre, la trasparenza del codice contribuisce direttamente alla verificabilità crittografica e alla fiducia nel programma. A differenza del software proprietario, in cui si accettano algoritmi di sicurezza e protocolli assumendo che siano stati implementati correttamente, l'Open Source permette di ispezionare l'implementazione crittografica a livello binario e sorgente. Questo riduce il rischio di fail-open logici[1] e assicura che non siano presenti funzioni nascoste o backdoor malevole. La resistenza a tali manipolazioni è intrinseca, poiché qualsiasi tentativo di infiltrazione nel codice sarebbe potenzialmente scoperto e reso pubblico, salvaguardando l'integrità del sistema.

Infine, l'infrastruttura di sviluppo aperta facilita la CIR (Computer Incident Response). Quando una vulnerabilità zero-day viene scoperta in un componente Open Source ampiamente utilizzato, la comunità può immediatamente analizzare la causa, sviluppare e distribuire una patch correttiva in tempi che spesso superano la reattività delle aziende proprietarie, che sono vincolate dai cicli di rilascio interni e dai processi di controllo centralizzati. Per le applicazioni di network monitoring e sicurezza, come quelle trattate in questa tesi, la rapidità di risposta è un fattore critico di resilienza.

In contrapposizione ai benefici intrinseci dell'Open Source, il modello proprietario introduce rischi significativi, non solo sotto il profilo tecnico, ma anche in termini di

[1] Con "fail-open logici" si intende un errore di programmazione in un sistema di sicurezza per cui, in caso di malfunzionamento, il sistema fallisce aprendosi (cioè permette l'accesso o il flusso di dati) anziché fallire chiudendosi (bloccando tutto, fail-safe o fail-closed).

Governance del rischio economico e operativo. L'impossibilità di accedere al codice sorgente causa una profonda opacità sui meccanismi interni, impedendo audit indipendenti di terze parti e rendendo necessario un atto di fede assoluta nell'integrità dello sviluppatore. Questo rischio di vulnerabilità, sia intenzionali che involontarie, rimane celato, in quanto il processo di scoperta dei bug è ristretto al controllo interno dell'azienda.

Dal punto di vista strategico, il software proprietario impone il fenomeno del Vendor Lock-in, creando una dipendenza tecnologica che va oltre il mero utilizzo del programma. L'utente è costretto a investire in formazione, infrastruttura e dati specifici per l'ecosistema di un unico fornitore. Questa dipendenza limita drasticamente la flessibilità e la capacità di migrazione verso soluzioni alternative, rendendo l'utente vulnerabile a cambiamenti arbitrari nelle politiche di supporto, nei prezzi e nelle condizioni di licenza, con conseguenze potenzialmente distruttive a lungo termine.

Infine, il modello Closed Source accentua il rischio di discontinuità del servizio e di obsolescenza programmata. L'innovazione è centralizzata e la sopravvivenza del software è legata alla solidità finanziaria e alle decisioni strategiche dell'azienda produttrice. Qualora l'azienda decidesse di interrompere il supporto (End-of-Life), gli utenti rimarrebbero privi del diritto legale o tecnico di accedere al codice sorgente per correggerlo, evolverlo o mantenerlo. Questa assenza di controllo condanna rapidamente il prodotto all'obsolescenza, minando la longevità e l'affidabilità di sistemi complessi che si basano su tali componenti. In sintesi, mentre il software proprietario ottimizza il controllo e il profitto per il singolo ente, l'Open Source massimizza la resilienza, la sicurezza e l'adattabilità del software, ponendo il sistema tecnologico al servizio della comunità.

3.3 Open Source nel contesto della Dynamic ARP Inspection

Come detto nel capitolo precedente, la Dynamic ARP Inspection è stata concepita e implementata come una caratteristica proprietaria integrata nelle apparecchiature di rete di classe enterprise. Tale integrazione comporta i rischi precedentemente discussi nel paragrafo sopra, risultando limitativa in funzione degli elementi evidenziati in precedenza. Si consideri soprattutto che, per gli amministratori di rete come per gli utenti comuni, risulta impossibile usufruire dell'implementazione qualora non siano in possesso di sistemi enterprise. L'impossibilità di permettersi migliaia di Euro di dispositivi professionali priva gli utenti del diritto a una funzione di sicurezza fondamentale per la mitigazione degli attacchi in LAN private o pubbliche.

L'obiettivo di questo progetto consiste nel democratizzare e rendere integrabile una funzionalità di sicurezza fondamentale. La scelta di implementare DAI utilizzando strumenti e componenti aperti, e senza dipendenza da licenze commerciali, realizza la conversione di una funzionalità semi-proprietaria in una soluzione neutra e funzionale. Questo approccio

garantisce immediatamente la trasparenza e la fiducia: il codice sorgente è completamente esposto, consentendo una verifica completa della logica di validazione dei messaggi ARP. L'amministratore di rete non è più tenuto a riporre fiducia implicita nel produttore dell'hardware, ma può verificare l'esatta esecuzione della validazione, elemento cruciale in un meccanismo che interviene attivamente sul flusso del traffico di rete.

Inoltre, basandosi su standard di programmazione aperti, la flessibilità di integrazione è svincolata da specifiche piattaforme hardware. Ciò ne consegue l'installazione su qualsiasi switch, router, server o dispositivo embedded compatibile, offrendo una soluzione di sicurezza perimetrale implementabile su hardware generico e a costo contenuto. Infine, il modello di sviluppo aperto facilita l'adattabilità e l'evoluzione sostenibile: la longevità del progetto non è vincolata alle decisioni di End-of-Life di un singolo fornitore, ma alla capacità della comunità di adattare e migliorare il sistema per mitigare nuove varianti di attacchi di ARP Spoofing che potrebbero emergere.

In sintesi, l'adozione del paradigma Open Source per la Dynamic ARP Inspection è una dimostrazione pratica di come i principi di libertà e trasparenza possano essere impiegati per costruire strumenti di sicurezza più resilienti, affidabili e accessibili rispetto alle tradizionali soluzioni chiuse.

Criterio	**Descrizione Sintetica**
Libera Ridistribuzione	La licenza non deve impedire la vendita o la donazione del software come componente di una distribuzione.
Codice Sorgente	Il programma deve includere il codice sorgente e consentirne la libera distribuzione in forma leggibile.
Lavori Derivati	La licenza deve permettere modifiche e la creazione di opere derivate, consentendone la distribuzione sotto gli stessi termini.
Integrità del Codice Sorgente dell'Autore	Può essere richiesto che i lavori derivati vengano distribuiti solo come "patch" o che siano chiaramente distinti dall'originale.
Non Discriminazione contro Persone o Gruppi	La licenza non deve discriminare persone o gruppi di persone.
Non Discriminazione contro Campi di Impresa	La licenza non deve impedire l'uso del software in uno specifico campo di attività (ad esempio, per scopi commerciali).
Distribuzione della Licenza	I diritti collegati al programma devono applicarsi a tutti coloro a cui il programma è ridistribuito, senza la necessità di un'ulteriore licenza.
La Licenza non deve essere Specifica per un Prodotto	I diritti concessi dal programma non devono dipendere dal fatto che esso faccia parte di una specifica distribuzione.
La Licenza non deve Contaminare altro Software	La licenza non deve imporre restrizioni su altri software che sono distribuiti insieme al programma con licenza aperta.
La Licenza deve essere Tecnologia Neutra	Nessuna disposizione della licenza deve essere basata su una singola tecnologia o tipo di interfaccia.

Tabella 2: I Dieci Criteri della Definizione Open Source (OSD) secondo la Open Source Initiative (OSI) [17]

Capitolo 4

Ambiente di Simulazione di Reti e di Sviluppo

Il capitolo presenta l'ambiente di sviluppo e virtualizzazione utilizzato per simulare, nel modo più fedele possibile, una rete demo composta inizialmente da un router e fino a quattro client, permettendo la scalabilità da scenario base (LAN singola) a scenario multi-rete.

4.1 Obiettivi e Criteri di Progettazione

L'allestimento dell'ambiente simulativo è stato guidato da due criteri metodologici primari: la fedeltà operativa e l'aderenza al paradigma Open Source. L'obiettivo principale era creare una piattaforma di testing che replicasse con la massima accuratezza possibile il contesto di una rete locale (LAN) gestita, dove i meccanismi di assegnazione dinamica degli indirizzi (DHCP) e la risoluzione degli indirizzi (ARP) siano attivi e vulnerabili.

In linea con la filosofia del progetto, la selezione degli strumenti di virtualizzazione e dei servizi di rete è stata orientata esclusivamente verso soluzioni Open Source. Questo assicura che l'intera catena di esecuzione, dal sistema operativo ai servizi di rete (DHCP/DNS) e all'isolamento (firewalling), sia completamente verificabile e riproducibile. Tale scelta non solo rafforza la coerenza del progetto con i principi di trasparenza, ma elimina ogni dipendenza da componenti proprietari che potrebbero alterare il comportamento dei protocolli o impedire l'analisi dettagliata del traffico a basso livello.

Di conseguenza, le decisioni successive, che riguardano l'adozione di un hypervisor specifico, la selezione della distribuzione Linux e l'impiego di servizi DHCP e di Firewall, sono tutte subordinate al requisito di garantire un ambiente di sviluppo stabile, controllabile e replicabile da parte di qualsiasi soggetto esterno.

4.2 Ambiente Virtuale

Oracle VM VirtualBox è stato scelto come hypervisor di riferimento per la sua affidabilità e la sua architettura modulare, essenziale per la simulazione di reti complesse [18]. Questo strumento consente l'avvio e la gestione simultanea di macchine virtuali basate su diverse distribuzioni guest e, fondamentale, offre il supporto nativo per la configurazione di molteplici schede di rete su ciascuna VM. Questa specifica funzionalità si è rivelata indispensabile, poiché ha permesso di assegnare all'entità simulante il router le interfacce virtuali necessarie a gestire la segmentazione e l'instradamento del traffico in reti logicamente separate.

Inoltre, l'utilizzo di VirtualBox ha facilitato la configurazione di funzionalità di rete avanzate, quali l'Internal Networking e il NAT, garantendo che l'interazione tra i client e i servizi di rete interni al laboratorio virtuale replicasse fedelmente le dinamiche di un ambiente fisico, mantenendo al contempo l'isolamento necessario per eseguire test di sicurezza e sniffing di pacchetti a basso livello.

Le impostazioni di VirtualBox permettono di allocare fino a quattro interfacce di rete per ogni macchina virtuale, ciascuna configurabile secondo diverse modalità operative: `NAT`, `Rete Interna`, `Bridge` e altre. In particolare, la modalità `Rete Interna` simula uno switch virtuale che collega tra loro solo le macchine associate allo stesso nome di rete interna, mentre con la modalità `NAT` le VM ottengono l'accesso controllato a Internet tramite il router virtuale incorporato nell'hypervisor. Questa flessibilità permette di replicare segmenti di LAN isolati, testare il routing tra subnet e simulare ambienti multihomed e polinodo ideali per l'analisi del traffico e degli attacchi.

Un ulteriore livello di personalizzazione è offerto dalla possibilità di configurare manualmente l'indirizzo MAC di ciascuna interfaccia di rete, essenziale per la tracciabilità negli esperimenti di sicurezza, la gestione delle lease DHCP e il funzionamento degli strumenti di monitoraggio come DAI.

Le istanze di calcolo virtuali sono state configurate impiegando la distribuzione Linux Debian, scelta per la sua rinomata stabilità, la sua natura interamente Open Source e la sua ridotta impronta operativa. La leggerezza del sistema operativo guest risulta essenziale per massimizzare l'efficienza nell'hypervisor, consentendo un impiego agile e l'uso di risorse hardware minime: ogni macchina virtuale è stata allocata con una memoria RAM nominale di 512 MB e un singolo core di processo. Questa parsimonia nell'allocazione delle risorse garantisce che le prestazioni del sistema di network monitoring non siano influenzate da sovraccarichi del sistema ospite.

Ai fini della simulazione, sono state implementate quattro entità client che rappresentano i dispositivi terminali (*end device*) della rete locale (LAN). Di queste, due sono designate come utenze legittime, configurate per operare in conformità ai protocolli di rete, una terza e una quarta possono essere assegnate a ruoli di attaccante, permettendo la validazione dell'efficacia reattiva dell'implementazione DAI sul router su più segmenti

di rete. L'ultima istanza di calcolo virtuale viene designata come nodo principale della rete simulata, assumendo il ruolo di router: tale VM è configurata per erogare i servizi di rete essenziali (DHCP, DNS, NAT, IP forwarding, DAI) utilizzando tool come Dnsmasq e Netfilter/IPTables.

In Tabella 3 è riassunta la composizione dell'ambiente virtuale per una fruizione immediata:

Nodo Virtuale	**Ruolo nella Simulazione**	**Servizi/Funzioni Chiave**
Debian Router	Nodo principale della rete e Gateway	DHCP (`Dnsmasq`), DNS, NAT (`Netfilter/IPTables`), IP Forwarding, Daemon DAI
Debian 1 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime
Debian 2 (Attaccante)	Sorgente dell'attacco	Generazione di pacchetti ARP spoofed (ARP Poisoning)
Debian 3 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime/ruolo attaccante
Debian 4 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime/ruolo attaccante

Tabella 3: Ruoli e Servizi dei Nodi nell'Ambiente di Simulazione

4.3 Topologia di Rete

La simulazione poggia su una topologia di rete volta a replicare un ambiente reale in cui sia possibile osservare fenomeni di routing, distribuzione di indirizzi tramite DHCP e attacchi ARP su segmenti di rete distinti. Per questo motivo, sono previste due configurazioni di riferimento: la prima, rappresentativa di una singola LAN, vede partecipare gli host appartenenti alla stessa rete, simulando lo scenario più semplice possibile; in secondo luogo, vengono incrementate le interfacce e configurati opportunamente i dispositivi per rappresentare una duplice rete e assistere a una simulazione più completa che possa essere scalabile.

4.3.1 Scenario Base

Nel contesto iniziale, è predisposta una topologia composta da un router Debian e tre host client su una singola sottorete interna. Tutti i client sono connessi tramite la modalità `Rete Interna` (intnet1), formando una LAN privata isolata dal traffico esterno, come illustrato nella Figura 2.

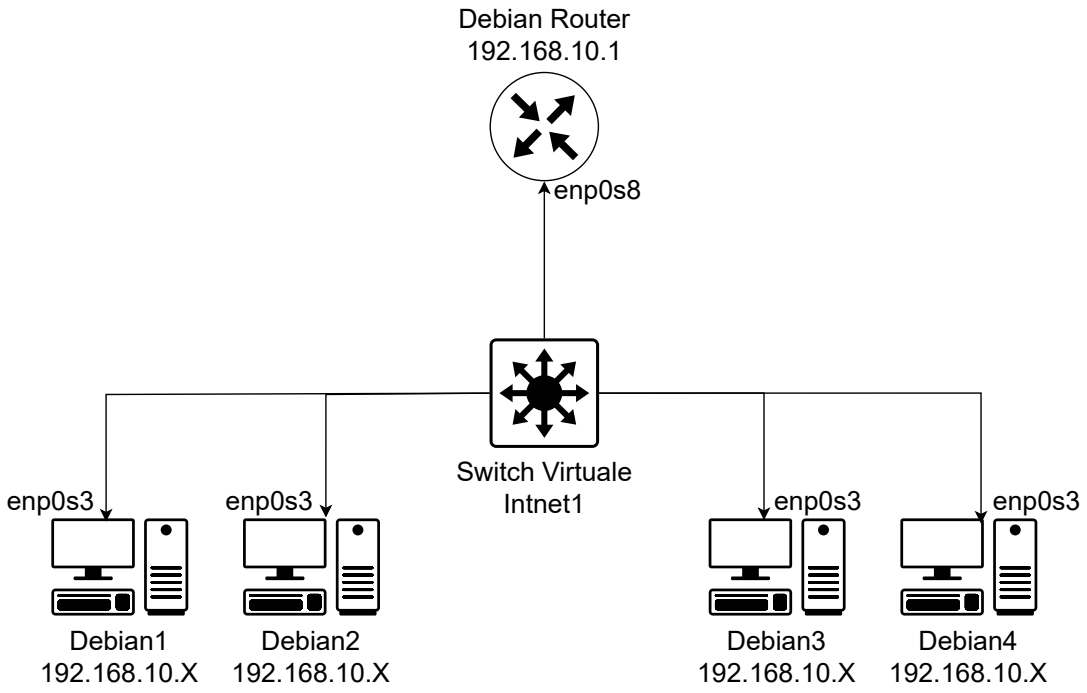

Figura 2: Topologia di rete base con router Debian e quattro client su intnet1

La seguente tabella 4 riassume i ruoli e le funzioni chiave dei nodi, mentre la tabella 5 riassume la configurazione di indirizzi e servizi di questo scenario.

Nodo Virtuale	**Ruolo nella Simulazione**	**Servizi/Funzioni Chiave**
Debian Router	Nodo principale della rete e Gateway	DHCP (`Dnsmasq`), DNS, NAT (`Netfilter/IPTables`), IP Forwarding, Daemon DAI
Debian 1 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime
Debian 2 (Attaccante)	Sorgente dell'attacco	Generazione di pacchetti ARP spoofed (ARP Poisoning)
Debian 3 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime
Debian 4 (Client Legittimo)	End device (Utente conforme)	Richiesta DHCP, Operazioni legittime

Tabella 4: Ruoli e Servizi dei Nodi nella Rete Singola

Nodo Virtuale	Interfaccia di Rete	Segmento
Debian Router	enp0s8	intnet1 (192.168.10.1)
Debian 1	enp0s3	intnet1 (192.168.10.x)
Debian 2	enp0s3	intnet1 (192.168.10.x)
Debian 3	enp0s3	intnet2 (192.168.10.x)
Debian 4	enp0s3	intnet2 (192.168.10.x)

Tabella 5: Configurazione dei Nodi nello Scenario Base

In questo contesto, la simulazione è in grado di offrire un primo scenario di test basilare sul funzionamento dell'infrastruttura e sugli attacchi ARP Spoofing in una LAN.

4.3.2 Scenario Multi-Rete

Per simulare scenari più complessi, la rete viene estesa aggiungendo una seconda sotto-rete interna (`intnet2`) e un ulteriore client. Si ottiene così una topologia parallela con due reti distinte (`192.168.10.0/24` e `192.168.20.0/24`), ciascuna presidiata dal router tramite opportune interfacce (`enp0s8` e `enp0s9`). In tale contesto, si possono orchestrare attacchi simultanei su sotto-reti diverse e verificare l'efficacia dell'applicativo DAI sia in condizioni isolate che in presenza di traffico misto.

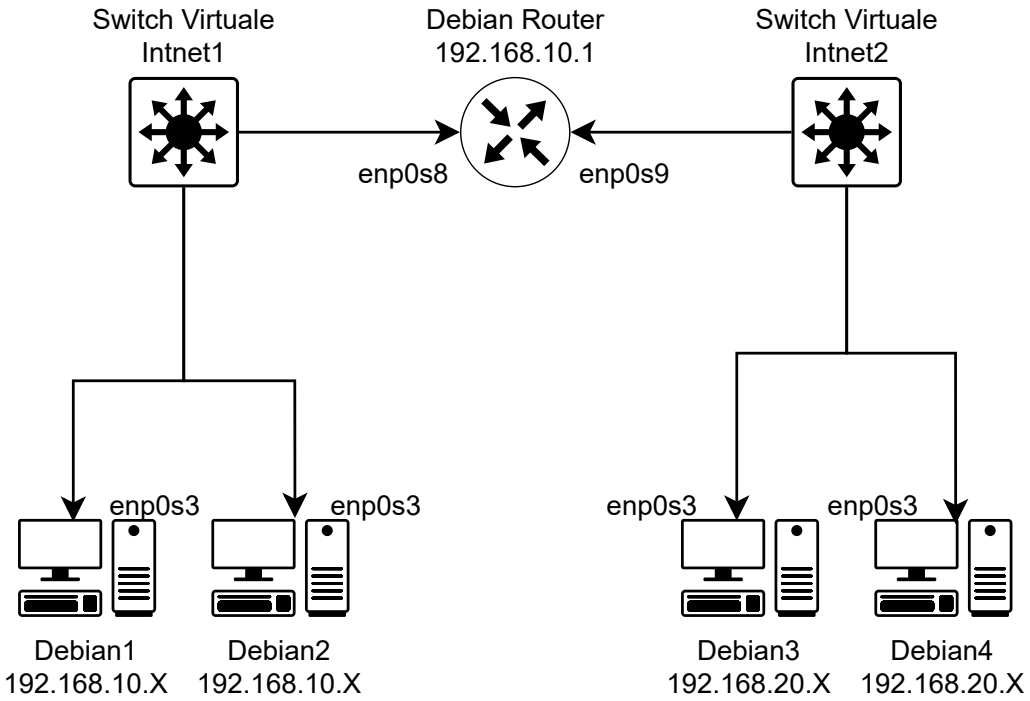

Figura 3: Topologia multi-rete con router Debian e quattro client su intnet1 e intnet2

La tabella 6 aggiornata per il nuovo scenario è la seguente:

Nodo Virtuale	**Interfaccia di Rete**	**Segmento**
Debian Router	enp0s8 / enpo0s9	intnet1 (192.168.10.1) / intnet2 (192.168.20.1)
Debian 1	enp0s3	intnet1 (192.168.10.x)
Debian 2	enp0s3	intnet1 (192.168.10.x)
Debian 3	enp0s3	intnet2 (192.168.20.x)
Debian 4	enp0s3	intnet2 (192.168.20.x)

Tabella 6: Configurazione dei Nodi nello Scenario Multi-Rete

Questo schema consente di valutare la difesa contro attacchi ARP su più segmenti simultaneamente, testare policy di routing e analizzare i log e l'efficacia del DAI Open Source in configurazioni realistiche.

4.4 Configurazioni delle Macchine Virtuali in Base alle Topologie di Rete

Nella seguente sezione vengono presentate le configurazioni delle VM implementate in funzione delle topologie di rete precedentemente discusse. Di fatto, l'impiego dell'ambiente simulato presenta delle importanti variazioni in base alla topologia a cui si fa riferimento.

4.4.1 Configurazione Client Debian Legittimi e Attaccante

L'installazione dei client Debian, sia legittimi che di attacco, è stata progettata per mantenere il sistema il più leggero e pulito possibile, in modo da ridurre la probabilità di interferenze tra i servizi e preservare l'integrità delle prove di simulazione. Per questo motivo, si è proceduto all'installazione di ambiente minimale che prevede esclusivamente la presenza dell'utente `root` e di un utente normale, escludendo qualsiasi Desktop Environment (DE) o servizio superfluo rispetto a quelli essenziali della distribuzione base. In questo modo si ottiene un ambiente operativo snello, privo di applicazioni accessorie non necessarie, utile per focalizzarsi esclusivamente sui test di rete.

Ai fini della configurazione di rete dei client, è stato modificato il file `/etc/network/interfaces`, impostando i parametri essenziali per la corretta gestione dell'interfaccia di rete. Di seguito si riporta un esempio di configurazione tipica:

- Configurazione dell'interfaccia di loopback per garantire la connettività locale.
- Configurazione dell'interfaccia principale (ad esempio `enp0s3`) con gestione dinamica tramite DHCP.

- Supporto a IPv6 tramite auto-configurazione.

Quest'impostazione garantisce una configurazione coerente e riproducibile, assicurando che ogni nodo partecipi correttamente alle operazioni di rete previste negli esperimenti di simulazione.

Per quanto riguarda il nodo attaccante, è stato previsto un ulteriore passaggio: la realizzazione e l'installazione di un applicativo Python, denominato `ARPReply.py`, sviluppato ad hoc per la generazione di pacchetti ARP Reply falsificati. Questo tool sfrutta la libreria `Scapy`[1] e consente di costruire e inviare pacchetti falsificati, impostando l'interfaccia, l'indirizzo IP e MAC di origine, il destinatario e il numero di pacchetti da inviare. Tramite questa soluzione, è possibile simulare un ARP Spoofing nella rete virtuale, fornendo una base concreta per l'analisi degli effetti e per la validazione dei meccanismi di difesa implementati, come quelli presentati nel capitolo successivo.

4.4.2 Configurazione del Router Debian e Applicativo DAI

La quarta macchina Debian viene installata come le precedenti: sistema minimale, senza un ambiente desktop né servizi aggiuntivi, con un solo utente `root` e un utente normale. Da qui si procede con modifiche mirate per trasformarla in un router multifunzione e host del servizio DAI Open Source.

Interfacce di rete e schema VMBox

Come illustrato in Tabella 7, la macchina router è collegata tramite quattro interfacce di rete: la principale con modalità NAT per l'accesso a internet tramite la macchina host, le altre in modalità "Rete Interna" per la comunicazione tra host privati.

Interfaccia	Tipo	Bridge	Descrizione
enp0s3	NAT	-	Accesso internet Host
enp0s8	Rete Interna	intnet1	Connessione rete interna

Tabella 7: Schema Interfacce di Rete VMBox

Il file `/etc/network/interfaces` riflette questa topologia. Di seguito, un estratto delle principali configurazioni:

```
auto lo
iface lo inet loopback

auto enp0s3
```

[1] `https://scapy.net/`

```
iface enp0s3 inet dhcp

auto enp0s8
iface enp0s8 inet static
    address 192.168.10.1
    netmask 255.255.255.0
```

Questa configurazione consente di gestire separatamente le comunicazioni verso ciascun client e verso l'esterno. `192.168.10.1` è l'indirizzo statico per identificare il Gateway e quindi il Router.

Servizi fondamentali: DNS/DHCP con Dnsmasq

Il servizio `Dnsmasq` è configurato per fornire DHCP e DNS locale. Di seguito, un estratto dal file di configurazione `/etc/dnsmasq.conf`:

```
interface=enp0s8
dhcp-range=192.168.10.10,192.168.10.20,24h
dhcp-option=option:router,192.168.10.1
dhcp-option=option:dns-server,8.8.8.8
```

Questi parametri consentono la distribuzione automatica di indirizzi IP e gateway per i client. In particolare, viene impostato un range di leasing degli IP da `192.168.10.100` a `192.168.10.200` e dati i riferimenti IP del Gateway e del server DNS.

Sysctl: abilitazione forwarding e protezione spoofing

Nel file `/etc/sysctl.conf`, come mostrato di seguito, viene attivato il forwarding IPv4 per permettere alla macchina di impersonare il comportamento di un router:

```
net.ipv4.ip_forward=1
```

Questa impostazione è di primaria importanza per permettere l'inoltro dei pacchetti secondo le regole stabilite nel punto seguente.

IPTables: NAT e filtri di sicurezza

Per garantire l'accesso ad Internet agli host interni e controllare il traffico tra le reti, il router Debian utilizza iptables per la gestione del NAT (Network Address Translation) e del forwarding. La principale regola del NAT applica il mascheramento alle connessioni in uscita verso la rete esterna (internet), consentendo ai dispositivi delle reti interne di condividere un unico indirizzo IP pubblico:

```
# NAT per accesso Internet dagli host interni
iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Questa regola dice al router di "nascondere" (mascherare) tutti i pacchetti che escono dall'interfaccia `enp0s3` (collegata alla rete esterna), sostituendo l'indirizzo sorgente con quello del router stesso. In questo modo, più host interni possono comunicare con l'esterno usando lo stesso indirizzo IP pubblico.

Per consentire il traffico tra le reti interne e verso internet, si aggiungono regole di forwarding specifiche. Ad esempio:

```
# Forwarding sulle interfacce interne
iptables -A FORWARD -i enp0s8 -o enp0s3 -j ACCEPT
```

Questa regola consente ai pacchetti provenienti dall'interfaccia interna `enp0s8` (ad esempio, la rete 192.168.10.0/24) di essere inoltrati verso l'esterno tramite `enp0s3`. In configurazioni multi-subnet, viene creata una regola analoga per ciascun segmento di rete interna che deve avere accesso a Internet o comunicare con altre sottoreti tramite il router.

Ulteriori regole possono essere aggiunte per:

- Limitare o filtrare il traffico tra le subnet interne secondo policy di sicurezza.
- Bloccare il traffico non autorizzato proveniente dall'esterno o tra host interni.
- Implementare segmentazione, isolamento e logging.

In sintesi, queste opzioni permettono un controllo granulare sul routing, favorendo sia la sicurezza sia la connettività, adattando il comportamento del router alle esigenze dell'infrastruttura.

Applicativo DAI Open Source

Infine, è il caso di menzionare che in questa macchina viene avviato il servizio ad hoc di Dynamic ARP Inspection. Il servizio si occuperà di monitorare i pacchetti ARP Reply ricevuti sulle interfacce interne, ovvero `enp0s8`, confrontando gli indirizzi MAC e IP dichiarati con le lease DHCP gestite da `Dnsmasq`, rilevando i tentativi di ARP poisoning.

4.4.3 Configurazione del Router Debian e Client nello Scenario Multi-Rete

Nel contesto multi-rete, il router Debian è dotato di una seconda interfaccia interna, `enp0s9`, dedicata alla gestione della subnet `192.168.20.0/24`. Oltre alla rete interna

`intnet1` (collegata a `enp0s8`), viene creata la "Rete Interna" `intnet2` , su cui si collegano i nuovi host virtuali, replicando la segmentazione tipica delle infrastrutture reali.

La tabella 8 seguente mostra la nuova topologia delle interfacce:

Interfaccia	Tipo	Rete Interna	Descrizione
enp0s3	NAT	-	Accesso internet Host
enp0s8	Rete Interna	intnet1	Connessione subnet 192.168.10.0/24
enp0s9	Rete Interna	intnet2	Connessione subnet 192.168.20.0/24

Tabella 8: Schema interfacce di rete VMBox in scenario multi-rete

Il file `/etc/network/interfaces` del router viene quindi modificato come segue:

```
auto enp0s8
iface enp0s8 inet static
    address 192.168.10.1
    netmask 255.255.255.0

auto enp0s9
iface enp0s9 inet static
    address 192.168.20.1
    netmask 255.255.255.0
```

Il servizio `Dnsmasq` è configurato per fornire DHCP in entrambe le sottoreti[2], con opzioni differenziate per ciascuna interfaccia:

```
interface=enp0s8
dhcp-range=set:enp0s8.192.168.10.10,192.168.10.20,24h
dhcp-option=:tag:enp0s8option,router,192.168.10.1
dhcp-option=tag:enp0s8option:,dns-server,8.8.8.8

interface=enp0s9
dhcp-range=set:enp0s9,192.168.20.20,192.168.20.30,24h
dhcp-option=tag:enp0s9,option:router,192.168.20.1
dhcp-option=tag:enp0s9,option:dns-server,8.8.8.8
```

Le regole `iptables` vengono adattate per consentire sia la comunicazione tra host interni e internet, sia tra le due reti private, secondo le policy di sicurezza stabilite:

[2]L'opzione `set/tag` viene abilitata per permettere a Dnsmasq di riconoscere le `dhcp-option` per ognuna delle interfacce.

```
        iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE

        iptables -A FORWARD -i enp0s8 -o enp0s3 -j ACCEPT
        iptables -A FORWARD -i enp0s9 -o enp0s3 -j ACCEPT

        iptables -A FORWARD -i enp0s8 -o enp0s9 -j ACCEPT
        iptables -A FORWARD -i enp0s9 -o enp0s8 -j ACCEPT
```

In questa configurazione, ciascuna interfaccia gestisce una propria subnet; il router, tramite il forwarding e le regole NAT, consente agli host delle due reti di comunicare tra loro e con l'esterno, mantenendo una separazione logica e applicando le opzioni di sicurezza definite in base alle esigenze dell'infrastruttura. Il servizio DAI viene configurato per monitorare entrambi i segmenti di rete, rilevando tentativi di attacco ARP in entrambi i domini; verrà argomentato come nel capitolo seguente.

Capitolo 5

Dynamic ARP Inspection Open Source

Nel seguente capitolo viene descritta in dettaglio l'implementazione open source del software DAI, partendo dalla progettazione e passando allo sviluppo vero e proprio in linguaggio C.

5.1 Introduzione e Obiettivi Architetturali

L'obiettivo primario di questo progetto è replicare le funzionalità della Dynamic ARP Inspection in un'istanza simulata descritta nel Capitolo 4, mantenendo efficienza e precisione. In tal senso, la scelta di eseguire l'implementazione in C rispecchiava pienamente i termini di efficienza per cui questo linguaggio è largamente utilizzato in questi contesti; inoltre, rende più agevole un'integrazione a basso livello nel firmware di apparati quali Switch Managed e Router.

Un requisito rigorosamente immutabile rispetto all'attuale DAI è la validazione delle associazioni ⟨IP, MAC⟩, con conseguente rilevazione di un ARP Spoofing, in tempo reale. Questo è l'obiettivo al quale è stata data priorità assoluta, che ha determinato, a cascata, buona parte della natura architetturale del progetto. Di fatto, per garantire una rilevazione in tempo reale, è necessario implementare una struttura esente da operazioni ad alto overhead. Tale obiettivo è reso più sfidante dalla necessità di estrarre e memorizzare non solo l'associazione base ⟨IP, MAC⟩, ma anche l'indirizzo MAC del mittente effettivo ⟨Sender MAC⟩ direttamente dal frame Ethernet, un dato cruciale per la successiva tracciabilità dell'attaccante. Di seguito vengono descritte le scelte progettuali che hanno contribuito alla rilevazione istantanea di ARP Spoofing.

La scelta di una struttura monolitica del progetto, senza multi-threading, comporterebbe una logica di elaborazione troppo basilare in un contesto in cui è necessario elaborare la ricezione, la memorizzazione e la validazione delle trame a livello data link. Tale struttura risulta semplicemente sottostimata rispetto al tasso di arrivo dei pacchetti di rete, comportando la perdita di una parte di essi e rallentando l'intero sistema. Nel progetto

viene quindi preferita una struttura multi-thread che possa soddisfare questo bisogno di molteplici operazioni contemporaneamente.

Sono state scartate le operazioni che coinvolgono l'interrogazione frequente del File System, che notoriamente comporta lunghe attese nei processi, a favore dell'implementazione di una cache temporanea che possa soddisfare le validazioni in modo molto più rapido. Questa scelta, in particolare nel contesto della validazione della trama, potrebbe comportare un rischio di sicurezza che verrà approfondito in seguito.

Data la dinamicità delle informazioni da memorizzare ed elaborare, in particolare il numero di pacchetti acquisiti dalle interfacce di rete, risulta impraticabile utilizzare strutture dati statiche che necessitano di una dichiarazione anticipata dello spazio di memoria da allocare. Viene implementata l'allocazione dinamica delle strutture dati, grazie alla quale si ha un controllo più minuzioso dell'utilizzo della memoria e una flessibilità incomparabile.

5.2 Architettura Multi-Thread e Flusso Dati

L'architettura multi-thread è comunemente identificata con la programmazione concorrente. Questo tipo di processi ad elevato parallelismo esecutivo richiede interazioni tra i thread che devono essere gestite correttamente. Nel contesto della Dynamic ARP Inspection viene impiegato un primo tipo di *thread ricevitori* per l'acquisizione delle trame a livello Data Link, una struttura dati condivisa tra i thread per memorizzare le associazioni complete, ovvero la tripla $\langle$IP, MAC, Sender MAC$\rangle$, e un numero variabile di *thread analizzatori* che estrapolano le associazioni dalla memoria e le validano.

Queste interazioni danno luogo a un pattern architetturale [19] conosciuto come modello Produttore-Consumatore. La Figura 4 schematizza il modello Produttore-Consumatore adottato per disaccoppiare la rapidità della ricezione di pacchetti (il Produttore) dalla latenza insita nel processo di analisi e validazione (il Consumatore).

Per garantire l'integrità dei dati scambiati tra i due thread, il sistema richiede l'implementazione di robusti meccanismi di sincronizzazione e l'applicazione di regole ferree che regolano le due condizioni critiche: la disponibilità di spazio per l'accodamento e la presenza effettiva di un elemento da analizzare nella coda. Verranno approfonditi tali argomenti in seguito.

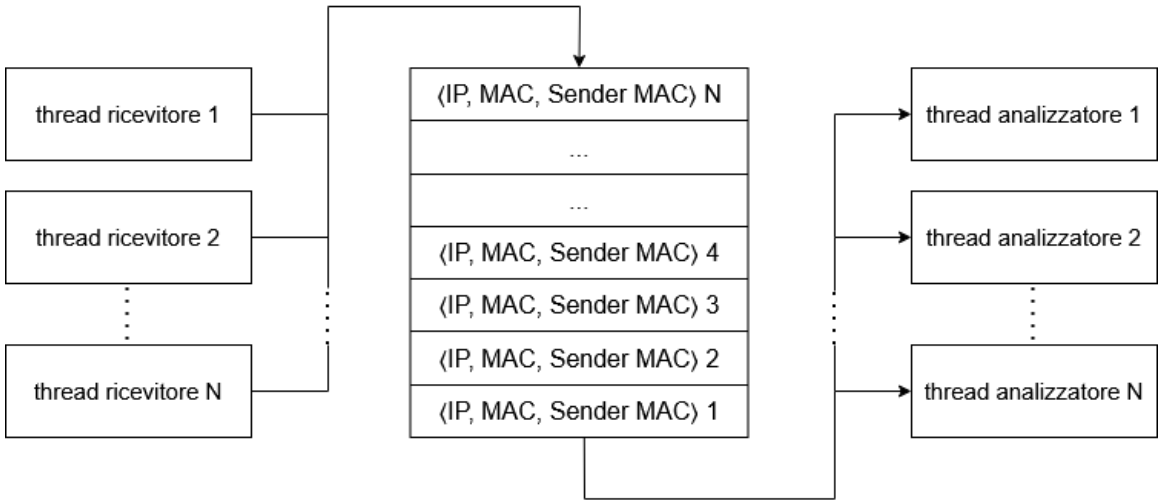

Figura 4: Diagramma Architettura Produttore Consumatore

Dopo aver definito l'architettura, è necessario delineare il ciclo di vita di un'associazione ⟨IP, MAC⟩ all'interno del sistema, il quale richiede l'interazione con componenti accessori oltre ai thread principali. Il Produttore (il thread ricevente) intercetta le trame a livello Data Link e, dopo una prima fase di parsing degli header essenziali, incapsula l'associazione in una struttura dati destinata alla coda. Da qui, l'associazione viene estratta dal Consumatore (il thread analizzatore) per la fase di validazione.

Cruciale per la correttezza del processo è la disponibilità di una base dati affidabile contenente le associazioni note e legittime. Pertanto, l'architettura include un terzo thread dedicato all'acquisizione e al mantenimento della cache delle lease DHCP. Questo thread gestisce l'accesso al File System e aggiorna la cache in memoria, isolando le lente operazioni di I/O dal percorso critico di analisi dei pacchetti. L'introduzione di questa base dati e del relativo thread di gestione completa il modello architetturale, garantendo che l'analizzatore operi sempre con informazioni aggiornate e coerenti.

5.3 Modulo di Acquisizione delle Trame e Utilizzo di Libpcap

Il Modulo di Acquisizione delle Trame, implementato nel thread ricevente e contenuto nel file receiver.c, costituisce il punto d'ingresso del traffico di rete nel daemon di ispezione. La sua funzione è cruciale: agire come Produttore nel modello architetturale, intercettando selettivamente le trame ARP Reply e incapsulandole rapidamente in una struttura dati per l'analisi successiva.

Per realizzare un'acquisizione efficiente, l'implementazione si avvale della libreria Libpcap [20], che opera direttamente a livello Data Link e accede alla socket di rete con un overhead minimo. La scelta di libpcap è strategica, in quanto consente di scaricare gran parte del lavoro di scrematura dei pacchetti sul kernel del sistema operativo. Questo trasferimento di carico è ottenuto tramite l'applicazione di un filtro BPF (Berkeley Packet Filter) rigoroso, definito da: `arp and arp[7] == 2`. Tale filtro garantisce che il daemon riceva nello user space esclusivamente le trame di ARP Reply (Opcode=2), scartando tutto il resto del traffico,inclusi gli ARP Request e i pacchetti IP/TCP, prima che venga elaborato dal thread ricevente. Questa preselezione è fondamentale per mantenere elevato il throughput e prevenire la perdita di pacchetti.

Una volta intercettata la trama, il thread esegue un parsing essenziale. Oltre a estrarre i dati che compongono l'associazione base ⟨IP, MAC⟩ dall'intestazione ARP, il modulo acquisisce un elemento di sicurezza critico: l'indirizzo MAC del mittente ⟨Sender MAC⟩ direttamente dall'intestazione Ethernet. Questo dato è fondamentale poiché, in caso di rilevamento di un'associazione ⟨IP, MAC⟩ non valida (cioè, un MAC falso) all'interno del payload ARP, l'indirizzo MAC contenuto nell'Intestazione Ethernet corrisponde al dispositivo che ha effettivamente trasmesso il pacchetto. L'estrazione e la memorizzazione di entrambi i valori, il MAC dichiarato nel payload e il MAC mittente nel frame Ethernet, garantiscono che l'analizzatore non solo possa stabilire l'invalidità dell'associazione, ma possa anche identificare in modo tracciabile l'origine fisica dell'attacco di ARP Cache Poisoning.

Infine, l'associazione completa ⟨IP, MAC, Sender MAC⟩ viene accodata tramite la funzione `enqueue_arp_association`, completando il ruolo del Produttore e assicurando che la fase di ricezione non sia mai il collo di bottiglia dell'intero applicativo.

5.4 Implementazione della Coda Condivisa e Sincronizzazione

Il modulo `queue.c` implementa la coda centrale condivisa, elemento fondamentale per garantire l'efficacia del modello Produttore-Consumatore in un ambiente di ispezione di rete ad alta velocità. Questa struttura dati è realizzata come un buffer circolare (Ring Buffer) a capacità fissa, definita dalla costante `ARP_QUEUE_SIZE`. La scelta di un buffer circolare è dettata dall'esigenza di massimizzare l'efficienza della memoria, consentendo il riutilizzo dinamico dello spazio liberato senza richiedere costose operazioni di spostamento dei dati.

Il meccanismo si basa sull'utilizzo di due puntatori interi, `head` e `tail`, e di un contatore `count` che traccia il numero attuale di elementi presenti. L'indice `tail` è gestito dal Produttore e indica la prossima posizione libera nel buffer, mentre l'indice `head` è gestito dal Consumatore e punta all'elemento più vecchio in attesa di essere processato. L'avanzamento di entrambi gli indici è controllato con l'operazione modulare (`% ARP_QUEUE_SIZE`),

che realizza in modo efficiente la proprietà First-In, First-Out (FIFO) della coda e rende effettiva la circolarità di quest'ultima.

L'implementazione della concorrenza e di tutte le primitive di sincronizzazione del modello Produttore-Consumatore è stata realizzata tramite l'API POSIX Threads (pthread) [21], la libreria standard per la gestione dei thread nell'ambiente C/Linux. L'uso di `pthread` consente di sfruttare i meccanismi di gestione della memoria condivisa e dello scheduling del sistema operativo in modo nativo ed efficiente. Il ricorso a questa libreria è essenziale per implementare le sezioni critiche e prevenire le race condition, garantendo che i thread di ricezione e di analisi possano accedere alla coda in modo sicuro e coordinato.

Per garantire l'integrità dei dati in un ambiente multi-thread, dove entrambi i thread accedono e modificano simultaneamente gli indici e il contatore, tutte le operazioni critiche sono protette da un Mutex (`pthread_mutex_t mutex`). L'acquisizione del lock all'inizio di `enqueue_arp_association` e `dequeue_arp_association` impedisce le race condition e garantisce che ogni singola operazione di accesso o modifica alla coda sia trattata come atomica.

A completamento del meccanismo di sincronizzazione, vengono utilizzate due Variabili di Condizione (`pthread_cond_t`), essenziali per la gestione delle condizioni limite del buffer. La variabile `not_full` entra in gioco per gestire l'eccesso di traffico, bloccando il Produttore in attesa (tramite `pthread_cond_wait`) quando la coda raggiunge la sua capacità massima. Parallelamente, la variabile `not_empty` blocca il Consumatore quando la coda è vuota, prevenendo letture non valide. L'uso congiunto di Mutex e Variabili di Condizione è fondamentale per l'efficienza: permette ai thread di dormire senza sprecare cicli di CPU in cicli di continua attesa e di essere risvegliati immediatamente tramite `pthread_cond_signal` non appena l'altro thread modifica lo stato della coda.

All'interno della funzione `enqueue_arp_association`, l'implementazione ricorre all'allocazione dinamica di memoria per ogni singola associazione tramite `malloc` e al successivo copy bit a bit della struttura. Sebbene l'allocazione dinamica garantisca flessibilità, introduce un overhead leggermente superiore rispetto a una pre-allocazione statica; tuttavia, la strategia di delegare al thread analizzatore l'onere della deallocazione tramite `free_arp_association`, una volta che l'associazione è stata processata, isola l'operazione di liberazione di memoria dal thread ricevente, proteggendo il percorso critico di acquisizione dei pacchetti.

Meccanismo	**Tipo**	**Descrizione e Scopo**
`pthread_mutex_t mutex`	Mutua Esclusione	Garantisce che un solo thread alla volta acceda alla coda, prevenendo la manipolazione concorrente di `head`, `tail` e `count`.
`pthread_cond_t not_empty`	Variabile di Condizione	Sospende il thread Consumatore (`dequeue`) quando la coda è vuota, in attesa di un segnale dal Produttore.
`pthread_cond_t not_full`	Variabile di Condizione	Sospende il thread Produttore (`enqueue`) quando la coda è piena, in attesa di un segnale dal Consumatore.
Condizione Evitata	**Definizione**	**Impatto sull'Applicazione di Rete**
Race Condition	Accesso concorrente ai dati condivisi.	Corruzione della coda (ad es., `head` o `tail` che puntano a valori errati), perdita di pacchetti ARP.
Active Waiting (Polling)	Thread che controlla continuamente una condizione.	Spreco di cicli CPU e aumento dell'overhead del sistema operativo, a discapito dell'efficienza.

Tabella 9: Riepilogo dei Meccanismi di Sincronizzazione e delle Condizioni

5.5 Modulo di Validazione (Analyzer)

Il Modulo di Validazione, la cui implementazione risiede nel file `analyzer.c`, costituisce il fulcro decisionale del sistema e opera in veste di Consumatore all'interno dell'architettura a thread concorrenti. Esso è primariamente responsabile dell'estrazione asincrona e dell'analisi della tripla associazione $\langle$IP, MAC, Sender MAC$\rangle$ contenuta nella coda condivisa. L'attivazione di ciascun thread analizzatore tramite la routine `analyzer_thread` conferisce al sistema la necessaria scalabilità orizzontale, permettendo l'elaborazione simultanea di molteplici associazioni e mitigando così il rischio di backlog durante i picchi di traffico ARP.

La routine del thread analizzatore è strutturata in un ciclo continuo che si avvia con la chiamata alla funzione `dequeue_arp_association`. In virtù delle primitive di

sincronizzazione POSIX Threads impiegate nella coda, il thread Consumatore rimane in uno stato di sospensione senza consumare cicli di CPU in cicli di attesa attiva, venendo risvegliato esclusivamente quando il Produttore segnala la disponibilità di un nuovo elemento. Una volta estratta l'associazione, il thread ne assume la piena proprietà e la indirizza alla la validazione.

La funzione `analyze_arp_association` incarna la logica di verifica, realizzando l'applicazione della Deny Policy del sistema attraverso l'interrogazione della Cache delle Lease DHCP. Infatti, con Deny Policy, si intende che ogni associazione ⟨IP, MAC⟩ ricevuta da un pacchetto ARP Reply è considerata non valida e non attendibile, fino a prova contraria. Il processo di validazione è articolato nel seguente modo.

Il thread riceve un puntatore alla struttura `arp_association_t`. Successivamente, gli indirizzi MAC e IP vengono convertiti dal formato binario a quello di stringa, un'operazione necessaria per facilitare il lookup nella struttura dati della cache. La verifica è delegata alla funzione `lease_cache_check`, che interroga in tempo reale la cache residente in memoria. L'obiettivo è determinare se la coppia ⟨IP, MAC⟩ (ovvero, l'associazione dichiarata nel payload ARP) è stata precedentemente registrata come legittima dal server DHCP.

Qualora la coppia ⟨IP, MAC⟩ sia riscontrata nella cache, l'associazione è ritenuta legittima e il processo analitico si conclude. Qualora si manifesti una discrepanza tra il MAC dichiarato nel payload ARP e quello atteso, si solleva un'eccezione di sicurezza, indicativa di un tentativo di ARP Cache Poisoning. In questo scenario, l'analizzatore sfrutta l'indirizzo Sender MAC, ottenuto dall'intestazione Ethernet, per registrare un alert e identificare in modo tracciabile l'origine fisica del pacchetto maligno.

Al termine dell'analisi, il thread analizzatore adempie alla sua responsabilità di gestione della memoria, eseguendo la funzione `free_arp_association`. Tale azione dealloca la memoria precedentemente allocata dal Produttore con `malloc`, assicurando che la cooperazione nella gestione della memoria prevenga le perdite (*memory leak*) e garantisca la stabilità del daemon durante l'esecuzione a lungo termine.

5.6 Validazione tramite File di Lease DHCP

Il modulo `lease_t.c` gestisce la componente statica di riferimento del sistema di Dynamic ARP Inspection, ovvero la Cache delle Lease DHCP. Questa cache rappresenta l'unica fonte attendibile per la verifica della legittimità delle associazioni ⟨IP, MAC⟩, garantendo dati autenticati. Per assolvere a questo compito, l'implementazione prevede un terzo thread dedicato (`lease_updater_thread`), il cui unico compito è isolare l'operazione lenta di I/O dal percorso critico di analisi dei pacchetti.

Il thread updater opera a intervalli regolari, definiti in fase di configurazione, per leggere il file di lease DHCP del server, nel nostro caso `/var/lib/misc/dnsmasq.leases`, e aggiornare la struttura `lease_cache_t` allocata in memoria. Questo approccio previene che il thread analizzatore debba mai attendere operazioni di accesso al file system.

La funzione `lease_cache_update` è il cuore del processo di I/O. Essa esegue l'apertura del file di lease e lo analizza riga per riga utilizzando `sscanf` all'interno della funzione statica `parse_lease_line`. Il parser estrae selettivamente gli indirizzi IP e MAC, ignorando i campi non essenziali per la validazione ARP, come il timestamp e l'hostname. Durante questo processo, viene allocata dinamicamente una nuova struttura dati `new_entries` con capacità variabile attraverso `realloc`, permettendo alla cache di adattarsi in modo flessibile al numero di client attivi in rete.

Cruciale per la robustezza del sistema è l'inclusione di una entry fissa per l'associazione $\langle$IP, MAC$\rangle$ del router all'inizio della nuova cache. Questa misura garantisce che la validazione dei pacchetti ARP inviati dal gateway di rete sia sempre effettuata, indipendentemente dalle dinamiche del servizio DHCP. Questa entry viene inizializzata in fase di avvio del daemon prima di caricare la cache dal file delle lease DHCP.

Una volta completata la lettura e l'inserimento di tutte le lease valide nella struttura temporanea, avviene la fase di swap atomico. Il thread di aggiornamento acquisisce il Mutex dedicato alla cache (`cache->mutex`), libera la memoria della vecchia cache (`free(cache->entries)`), e riassegna il puntatore `cache->entries` alla nuova struttura. Il lock sulla cache è mantenuto solo per la brevissima durata di questa operazione di scambio, riducendo al minimo l'impatto sulla disponibilità dei dati per i thread analizzatori.

La funzione `lease_cache_check` è l'unico punto di interazione tra il Modulo di Validazione e la Cache ed è progettata per la massima velocità. Il thread analizzatore invoca questa funzione passando l'IP e il MAC estratti dal pacchetto, e la funzione esegue una ricerca sequenziale diretta (`strcmp` e `strcasecmp`) attraverso la struttura dati in memoria.

Sebbene l'accesso esclusivo alla memoria RAM garantito dall'isolamento dell'I/O renda l'operazione intrinsecamente rapida, l'algoritmo di ricerca implementato è di tipo lineare. La complessità temporale di questa ricerca sequenziale, nel caso peggiore o nel caso medio in cui la lease non è trovata, è $O(n)$, dove n è il numero totale di lease attive nella cache. Questa complessità, pur accettabile in ambienti di rete locali con un numero contenuto di host (generalmente $n < 1000$), diventerebbe un fattore limitante in data center o reti aziendali di grandi dimensioni. In tali contesti, la costante di proporzionalità n potrebbe compromettere i requisiti di latenza in tempo reale del sistema.

È tuttavia doveroso notare che, nelle reti aziendali, un elevato numero di host (con n superiore alla soglia critica) è più frequente negli ambienti wireless Wi-Fi, dove la densità di utenti è concentrata. Nelle configurazioni cablate tradizionali, il numero di host per segmento di rete o VLAN è tipicamente inferiore e più gestibile. Negli ambienti Wi-Fi ad alta densità, è prassi comune segmentare la rete con punti di accesso (AP) multipli che operano su VLAN separate. Questo approccio divide il carico, mantenendo il valore di n, il numero di lease per specifica VLAN che il daemon monitora, al di sotto del limite critico che renderebbe la ricerca $O(n)$ impraticabile.

È fondamentale che anche la funzione di lookup acquisisca il Mutex della cache. Sebbene il thread analizzatore non modifichi la cache, l'acquisizione del Mutex impedisce che

l'operazione di lookup avvenga simultaneamente all'operazione di swap da parte del thread updater. Tale sincronizzazione garantisce che ogni ricerca avvenga su una struttura dati integra e coerente. Il risultato (TRUE o FALSE) viene quindi restituito al thread analizzatore per la decisione finale di sicurezza, come discusso nel capitolo precedente.

5.7 Dettagli Implementativi e Struttura del Codice Sorgente

Nella presente sezione si presenta il cuore dei file sorgenti che compongono il programma. Ovviamente, tenuto conto della lunghezza, verranno presentati degli estratti delle principali funzionalità con relative descrizioni.

5.7.1 Implementazione del Receiver

Di seguito viene riportata la definizione della struttura dati utilizzata per passare gli argomenti al thread di ricezione:

```
typedef struct {
    int num;
    pthread_mutex_t* stdout_mutex;
    char *interface;
    arp_association_queue_t *queue;
} receiver_t_args;
```

I campi della struttura `receiver_t_args` svolgono le seguenti funzioni:

1. **`stdout_mutex`** Puntatore al mutex per sincronizzare la stampa su terminale (debug).
2. **`interface`** Stringa contenente il nome dell'interfaccia di rete (es. `enp0s8`).
3. **`queue`** Puntatore alla coda thread-safe dove verranno inseriti i pacchetti ARP parsati.

Successivamente, il ciclo principale di acquisizione viene eseguito dalla funzione `receiver_thread`. La funzione inizializza l'handle `pcap`, applica il filtro BPF per intercettare solo le ARP Reply ed entra in un loop infinito di ascolto:

```
void *receiver_thread(void *args) {
    receiver_t_args* t_args = (receiver_t_args *) args;
    pcap_t *handle;
    struct bpf_program fp;
    bpf_u_int32 net = 0;
```

```
            // ... setup variabili ...

            handle = pcap_open_live(t_args->interface, BUFSIZ, 1, 1000, errbuf);
            const char *arp_filter = "arp and arp[7] == 2"; // Filtro per ARP Reply
            pcap_compile(handle, &fp, arp_filter, 1, net);
            pcap_setfilter(handle, &fp);

            struct pcap_pkthdr header;
            const unsigned char *packet;

            while (1) {
                packet = pcap_next(handle, &header);
                if (packet) {
                    // Check lunghezza minima header Ethernet + ARP
                    if (header.caplen >= 14 + 28) {
                        const unsigned char *mac_sender = packet + 6;
                        const unsigned char *arp_header = packet + 14;
                        const unsigned char *mac_bind = arp_header + 8;
                        const unsigned char *ip_bind = arp_header + 14;

                        arp_association_t association;
                        memcpy(association.mac_addr, mac_bind, ETH_ALEN);
                        memcpy(&association.ip_addr, ip_bind, 4);
                        memcpy(association.mac_addr_sender, mac_sender, ETH_ALEN);

                        // ... setup timestamp ...

                        arp_association_queue_t *arp_queue = t_args->queue;
                        enqueue_arp_association(arp_queue, association);
                    }
            }
            }
        }
```

5.7.2 Struttura della Coda e Gestione della Concorrenza

Per disaccoppiare la fase di cattura (ad alta frequenza) dalla fase di analisi (potenzialmente più lenta), è stata implementata una coda circolare (Circular Buffer) thread-safe. Questa struttura agisce come buffer intermedio secondo il pattern Produttore-Consumatore.

Di seguito vengono riportate le strutture dati fondamentali definite nell'header file della coda:

```
        #define ARP_QUEUE_SIZE 1000

        // Struct dell'associazione MAC - IP - MAC Sender ricevuta dal receiver
        typedef struct {
```

```
        unsigned char mac_addr[ETH_ALEN];
        struct in_addr ip_addr;
        unsigned char mac_addr_sender[ETH_ALEN];
    } arp_association_t;

    // Coda Circolare Thread-Safe
    typedef struct {
        arp_association_t *buffer[ARP_QUEUE_SIZE];
        int head;
        int tail;
        int count;
        pthread_mutex_t mutex;          // Primitive di sincronizzazione
        pthread_cond_t not_full;
        pthread_cond_t not_empty;
        pthread_mutex_t *stdout_mutex;
    } arp_association_queue_t;

    // Interfaccia pubblica della coda
    void arp_queue_init(arp_association_queue_t *queue,
                        pthread_mutex_t *stdout_mutex);
    void enqueue_arp_association(arp_association_queue_t *queue,
                                 arp_association_t association);
    arp_association_t *dequeue_arp_association(arp_association_queue_t *queue);
```

La struttura `arp_association_queue_t` integra le primitive di sincronizzazione POSIX (`pthread_mutex` e `pthread_cond`) necessarie per garantire l'accesso esclusivo al buffer condiviso. In particolare, `not_full` e `not_empty` permettono di sospendere rispettivamente il thread *Receiver* quando la coda è piena e il thread *Analyzer* quando la coda è vuota, ottimizzando l'uso della CPU ed evitando attese attive (spin-lock).

5.7.3 Implementazione dell'Analyzer

Il componente *Analyzer* agisce come consumatore all'interno dell'architettura. Il suo compito è prelevare le associazioni dalla coda e verificarne la legittimità confrontandole con il database delle assegnazioni DHCP (Lease Cache).

La struttura dati passata come argomento ai thread di analisi contiene i riferimenti alle risorse condivise necessarie per l'operazione:

```
    typedef struct {
        int num;
        pthread_mutex_t* stdout_mutex;
        lease_cache_t *lease_cache;
        arp_association_queue_t *queue;
    } analyzer_t_args;
```

I campi principali sono:

lease_cache Puntatore alla struttura dati che mantiene lo stato corrente delle assegnazioni IP-MAC (DHCP Snooping database).

queue Riferimento alla coda condivisa popolata dal Receiver.

Il ciclo di vita del thread è definito nella funzione `analyzer_thread`. Questa implementa un loop continuo che esegue l'operazione di dequeue: se la coda restituisce un elemento valido, viene invocata la logica di analisi, altrimenti il thread attende (tramite le condition variable interne alla coda).

```
void *analyzer_thread(void *args) {
    analyzer_t_args *t_args = (analyzer_t_args *) args;
    arp_association_queue_t *arp_queue = t_args->queue;

    while (1) {
        // Prelievo bloccante dalla coda thread-safe
        arp_association_t *association = dequeue_arp_association(arp_queue);

        if (association) {
            // Avvio analisi logica
            analyze_arp_association(association, t_args->lease_cache);

            // Rilascio della memoria allocata per l'associazione
            free_arp_association(association);
        }
    }
    pthread_exit(NULL);
}
```

La funzione `analyze_arp_association` si occupa di trasformare i dati grezzi di rete (byte array) in stringhe leggibili e di interrogare la cache.

Si noti che la logica di validazione effettiva è delegata al modulo *Lease Cache*, garantendo una separazione delle responsabilità.

```
void analyze_arp_association(arp_association_t *association,
                             lease_cache_t *lease_cache) {

    char mac_str[18], mac_sender[18], ip_addr[INET_ADDRSTRLEN];

    // Conversione da formato binario (network byte order) a stringa
    mac_to_str(association->mac_addr, mac_str);
    mac_to_str(association->mac_addr_sender, mac_sender);
    inet_ntop(AF_INET, &association->ip_addr, ip_addr, INET_ADDRSTRLEN);
```

```
        // Verifica della corrispondenza IP-MAC nel database DHCP
        int is_valid = lease_cache_check(lease_cache, ip_addr, mac_str);

        // ... gestione dell'esito (Log / Alert / ...)
    }
```

Come mostrato nell'ultimo frammento, la decisione finale sulla legittimità del pacchetto ARP dipende dal valore restituito da `lease_cache_check`, la cui implementazione interna verrà approfondita nella sezione successiva dedicata alla gestione dei Lease DHCP.

5.7.4 Gestione dei Lease DHCP e Validazione

Il componente *Lease Cache* rappresenta la fonte di verità per il sistema DAI. Esso mantiene in memoria una lista aggiornata delle associazioni IP-MAC valide, ottenute parsando il file dei lease del server DHCP (es. `dnsmasq.leases`).

Per questa implementazione prototipale, le costanti relative agli intervalli di aggiornamento e, in particolare, agli indirizzi fisici e logici del Router (Gateway) sono state definite staticamente tramite macro del preprocessore.

Si sottolinea che l'uso di indirizzi hardcoded (`ROUTER_IP_STR` e `ROUTER_MAC_STR`) costituisce una limitazione nota dell'attuale versione, introdotta per semplificare l'ambiente di test. Una gestione dinamica della configurazione del gateway è prevista e sarà discussa nel capitolo 7 conclusivo relativo agli sviluppi futuri.

Di seguito le strutture dati e le definizioni principali:

```
    #define UPDATE_INTERVAL 5
    #define MAC_LEN 17
    #define IP_LEN INET_ADDRSTRLEN

    // Indirizzi statici del Router per le due sottoreti (limitazione attuale)
    #define ROUTER_IP_STR    "192.168.10.1"
    #define ROUTER_MAC_STR   "08:00:27:68:55:09"
    #define ROUTER_IP_STR2   "192.168.20.1"
    #define ROUTER_MAC_STR2  "08:00:27:02:39:b1"

    // Associazione IP-MAC singola
    typedef struct {
        char mac[MAC_LEN + 1]; // +1 per terminatore \0
        char ip[IP_LEN];
    } lease_entry_t;

    // Struttura per la cache globale dei lease (Thread-Safe)
    typedef struct {
        lease_entry_t *entries;  // array dinamico di lease
```

```
    size_t count;            // numero attuale di lease
    pthread_mutex_t *mutex;  // protezione accesso concorrente
    pthread_mutex_t *stdout_mutex;
} lease_cache_t;

// Argomenti per il thread di aggiornamento periodico
typedef struct {
    lease_cache_t *cache;
    int update_interval_sec;
} lease_updater_t_args;
```

La funzione cuore della validazione è `lease_cache_check`. Questa funzione viene invocata dall'Analyzer per ogni pacchetto ARP ricevuto. Il controllo avviene in due fasi: prima si verifica se la coppia IP-MAC è presente nel database dei lease DHCP; se la ricerca fallisce, si controlla se l'associazione corrisponde a uno degli indirizzi legittimi del Router (Gateway), permettendo così il traffico infrastrutturale essenziale.

```
int lease_cache_check(lease_cache_t *cache,
                      const char *search_ip_str,
                      const char *search_mac_str) {
    int found = 0;

    // Accesso in lettura protetto da Mutex
    pthread_mutex_lock(cache->mutex);

    for (size_t i = 0; i < cache->count; i++) {
        // Confronto stringhe case-insensitive per il MAC
        if (strcmp(cache->entries[i].ip, search_ip_str) == 0 &&
            strcasecmp(cache->entries[i].mac, search_mac_str) == 0) {
            found = 1;
            break;
        }

       // Se non trovato nei lease, verifica se è un Router legittimo con test-case
        if (!found) {
            int is_router_mac = (strcasecmp(search_mac_str, ROUTER_MAC_STR) == 0 ||
                                 strcasecmp(search_mac_str, ROUTER_MAC_STR2) == 0);
            int is_router_ip  = (strcmp(search_ip_str, ROUTER_IP_STR) == 0 ||
                                 strcmp(search_ip_str, ROUTER_IP_STR2) == 0);

            if (is_router_mac && is_router_ip) {
                found = 1;
                break;
            }
        }
    }
```

```
    pthread_mutex_unlock(cache->mutex);
    return found;
}
```

5.7.5 Librerie e Struttura dei File Sorgente

Lo sviluppo del demone è stato realizzato interamente in linguaggio C, sfruttando un insieme selezionato di librerie standard e specifiche per il networking al fine di garantire performance e controllo a basso livello.

Le dipendenze principali del progetto sono:

- **Libpcap**: Libreria standard de facto per la cattura di pacchetti di rete. Utilizzata nel modulo *Receiver* per l'interfacciamento con il device driver della scheda di rete in modalità promiscua e per il filtraggio efficiente tramite BPF (Berkeley Packet Filter).
- **POSIX Threads (Pthread)**: Libreria conforme allo standard IEEE 1003.1c per la gestione della programmazione concorrente. Essenziale per l'orchestrazione dei thread *Receiver*, *Analyzer* e *Lease Updater*, nonché per le primitive di sincronizzazione (Mutex e Condition Variables) necessarie alla gestione della coda condivisa.
- **Standard C Library (glibc)**: Utilizzata per la gestione della memoria, manipolazione delle stringhe e operazioni di I/O, incluse le socket API (`sys/socket.h`, `arpa/inet.h`) per la gestione degli indirizzi IP.

La tabella 10 riassume l'organizzazione dei file sorgente del progetto, evidenziando le responsabilità di ciascun modulo.

File Sorgente	Descrizione Funzionale
`main.c`	Entry point dell'applicazione. Gestisce il parsing degli argomenti, l'inizializzazione delle strutture dati globali, la gestione dei segnali di sistema (es. `SIGINT` per graceful shutdown) e l'avvio dei thread principali.
`queue.c`	Implementazione della struttura dati *Ring Buffer* (Coda Circolare). Incapsula le logiche di inserimento e prelievo thread-safe, gestendo la sincronizzazione tra Produttore e Consumatore.
`receiver.c`	Modulo di interfaccia di rete. Configura la sessione di cattura `pcap`, compila e applica i filtri BPF per isolare il traffico ARP e popola la coda condivisa con i pacchetti grezzi.
`analyzer.c`	Core logic del sistema DAI. Preleva i pacchetti dalla coda, effettua il parsing dei protocolli Ethernet/ARP e convalida la tripla ⟨IP, MAC, Sender MAC⟩ interrogando la cache.
`lease_cache.c`	Gestione della "Source of Truth". Implementa il thread periodico che legge il file di lease del server DHCP (I/O su disco), aggiorna la struttura dati in memoria e fornisce l'interfaccia di ricerca sicura (`thread-safe read`).

Tabella 10: Sommario dell'organizzazione dei file sorgente del progetto

Capitolo 6

Testing ed Evidenze

6.1 Criteri ed Obbiettivi

La validazione di un sistema di sicurezza di rete come il *Dynamic ARP Inspection* (DAI) non può limitarsi alla sola verifica funzionale della correttezza logica (i.e., il blocco di un pacchetto malevolo), ma deve necessariamente includere un'analisi quantitativa delle prestazioni. Essendo il DAI un componente che opera in linea sul traffico di rete a Livello 2, qualsiasi inefficienza nel codice si tradurrebbe in latenza per l'utente finale o, nel caso peggiore, in perdita di pacchetti legittimi (packet loss) e interruzione del servizio.

Per valutare l'efficacia e l'efficienza della soluzione Open Source proposta, sono state definite quattro metriche fondamentali. Queste misurazioni ci permettono di quantificare il comportamento del sistema sotto diversi profili di carico e di confrontare oggettivamente le diverse configurazioni architetturali (Single-Thread vs Multi-Thread).

Di seguito vengono formalizzate le metriche utilizzate e il loro significato nel contesto del progetto.

6.1.1 Latenza di Elaborazione (Processing Latency)

La latenza di elaborazione, misurata in microsecondi (μs), rappresenta l'intervallo di tempo che intercorre tra l'istante in cui un pacchetto ARP viene catturato dal thread ricevitore e l'istante in cui il sistema emette un verdetto (legittimo o malevolo) al termine dell'analisi.

Formalmente, per ogni pacchetto p, la latenza L_p è calcolata come:

$$L_p = T_{out} - T_{in}$$

dove T_{in} è il timestamp acquisito tramite *Monotonic Clock* al momento dell'inserimento nella coda condivisa (enqueue), e T_{out} è il timestamp acquisito al termine della funzione di validazione.

Significato nel contesto: Questa metrica è il principale indicatore della reattività del sistema ("Real-Time"). Poiché il protocollo ARP è bloccante per la comunicazione IP successiva, un valore elevato di latenza introdurrebbe ritardi nell'instaurazione delle connessioni di rete. L'obiettivo è minimizzare L_p affinché l'ispezione risulti trasparente all'utente. Nei grafici successivi, analizzeremo sia la latenza media che i picchi di latenza per valutare la stabilità sotto stress.

6.1.2 Throughput Sostenibile (Packets Per Second - PPS)

Il Throughput indica la capacità del sistema di processare una data mole di traffico nell'unità di tempo. Viene misurato in Pacchetti Per Secondo (PPS).

Questa metrica viene calcolata dal modulo di monitoraggio campionando il numero di pacchetti processati ΔN in un intervallo di tempo Δt:

$$PPS = \frac{\Delta N}{\Delta t}$$

Significato nel contesto: Questa metrica misura la "potenza bruta" del motore di ispezione ed è fondamentale per valutare la scalabilità. Durante un attacco di tipo *ARP Flood* o in presenza di picchi di traffico broadcast legittimo (es. tempeste di broadcast), il sistema deve essere in grado di sostenere un throughput elevato senza collassare. I test di carico mirano a individuare il punto di saturazione del sistema, ovvero il numero massimo di PPS gestibili prima che si verifichi un degrado delle prestazioni.

6.1.3 Occupazione della Coda (Queue Load Saturation)

Questa metrica percentuale misura il livello di riempimento del buffer circolare condiviso tra il thread produttore (Receiver) e i thread consumatori (Analyzers). Dato che il campionamento periodico potrebbe non catturare picchi istantanei di riempimento, il software è stato strumentato per registrare il **Picco di Occupazione** (*Peak Queue Load*) raggiunto nell'intervallo di campionamento.

$$Q_{\%} = \left(\frac{Count_{max}}{Size_{queue}} \right) \times 100$$

Significato nel contesto: Questa è la metrica diagnostica principale per identificare i "colli di bottiglia" nell'architettura Produttore-Consumatore.

- Un valore vicino allo **0%** indica che i consumatori sono più veloci del produttore: la coda viene smaltita istantaneamente (situazione ideale).

- Un valore del **100%** indica saturazione: i consumatori non riescono a smaltire il carico, la coda si riempie e il produttore viene bloccato, causando potenziale perdita di pacchetti di rete a livello di kernel.

L'analisi di questa metrica sarà determinante per dimostrare i vantaggi del passaggio da un'architettura Single-Thread a una Multi-Thread.

6.1.4 Accuratezza del Rilevamento (Detection Rate)

Rappresenta la capacità funzionale del sistema di distinguere correttamente tra traffico lecito e illecito. Viene misurata confrontando il numero di attacchi iniettati durante la simulazione con il numero di alert registrati dal sistema nel contatore `AttacksTotal`.

Significato nel contesto: Oltre alle prestazioni velocistiche, è imperativo che il sistema garantisca la sicurezza. In scenari di stress (es. Denial of Service), è cruciale verificare che il sistema non vada in "fail-open" (lasciando passare attacchi) né scarti traffico legittimo a causa del sovraccarico. Questa metrica confermerà la robustezza della logica di validazione anche in condizioni critiche.

6.2 Strumentazione Integrata nel Software per il Testing

Per ottenere le metriche definite nella sezione 6.1 senza alterare significativamente le prestazioni del sistema (riducendo al minimo l'effetto osservatore), è stato necessario "attrezzare" il codice sorgente. L'approccio adottato consiste nell'introduzione di strumenti di misurazione a basso livello e nell'implementazione di un thread dedicato esclusivamente alla raccolta e alla persistenza dei dati statistici su file CSV. Viene prediletto tale formato di file per la raccolta di dati in un formato facilmente integrabile in diagrammi e infografica.

Di seguito vengono descritti i moduli implementativi introdotti per il testing.

6.2.1 Estensione delle Strutture Dati

Il primo passo ha riguardato l'estensione delle strutture dati fondamentali definite in `arp_queue.h`. È stato introdotto un campo `reception_time` all'interno della struttura del pacchetto ARP per tracciare l'istante esatto di ingresso nel sistema. Inoltre, è stata definita una struttura `dai_metrics_t` per aggregare i contatori globali in modo thread-safe e aggiunto un campo `peak_count` alla coda per rilevare i picchi di utilizzo istantanei che sfuggirebbero a un semplice campionamento periodico.

```
        // Estensione della struttura pacchetto con timestamp
        typedef struct {
            unsigned char mac_addr[ETH_ALEN];
```

```
        struct in_addr ip_addr;
        unsigned char mac_addr_sender[ETH_ALEN];
        struct timespec reception_time; // Timestamp (CLOCK_MONOTONIC)
    } arp_association_t;

    // Struttura per le metriche globali condivise
    typedef struct {
        unsigned long total_processed;  // Totale pacchetti analizzati
        unsigned long attacks_detected; // Totale attacchi rilevati
        double total_latency_accum;     // Accumulatore latenza (per calcolo media)
        pthread_mutex_t mutex;          // Mutex per accesso concorrente
    } dai_metrics_t;

    // Aggiunta del Peak Meter nella Coda
    typedef struct {
        // ... altri campi (buffer, head, tail) ...
        int count;      // Elementi attuali
        int peak_count; // Massimo picco raggiunto nell'intervallo
        // ... mutex e condition variables ...
    } arp_association_queue_t;
```

6.2.2 Misurazione della Latenza e Peak Detection

Per calcolare la latenza di elaborazione (L_p), il modulo `receiver.c` è stato modificato per acquisire il timestamp tramite `clock_gettime(CLOCK_MONOTONIC)` immediatamente prima dell'inserimento in coda. L'uso del clock monotonico è essenziale per garantire la precisione indipendentemente da eventuali cambi dell'orario di sistema.

Parallelamente, nel modulo `queue.c`, è stata implementata la logica di *Peak Hold* all'interno della funzione di enqueue. Ogni volta che un pacchetto viene inserito, si verifica se il riempimento attuale supera il massimo registrato.

```
    // In queue.c - Logica Peak Hold durante l'enqueue
    queue->count++;

    // Se il conteggio attuale supera il picco storico dell'intervallo, aggiorniamo
    if (queue->count > queue->peak_count) {
        queue->peak_count = queue->count;
    }
```

Nel modulo `analyzer.c`, al termine della validazione, viene acquisito il timestamp finale e calcolata la differenza.

```
    // In analyzer.c - Calcolo Latenza
    struct timespec end_time;
```

```
        clock_gettime(CLOCK_MONOTONIC, &end_time);
        double latency_us = time_diff_micros(association->reception_time, end_time);

        // Aggiornamento atomico delle metriche globali
        pthread_mutex_lock(&metrics->mutex);
        metrics->total_processed++;
        metrics->total_latency_accum += latency_us;
        if (!is_valid) {
            metrics->attacks_detected++;
        }
        pthread_mutex_unlock(&metrics->mutex);
```

6.2.3 Modulo di Monitoraggio e Logging (Monitor Thread)

Per garantire che l'attività di misurazione non rallentasse il processo critico di ispezione (come evidenziato dai test preliminari sull'impatto dell'I/O), è stato implementato un nuovo modulo `monitor_t.c`. Questo thread opera in modo asincrono rispetto al flusso dei pacchetti: si risveglia a intervalli regolari di 0.1 secondi (100ms), preleva un'istantanea atomica dei contatori e del picco della coda, calcola i delta (PPS e Latenza Media dell'intervallo) e scrive i risultati su un file CSV ottimizzato per la successiva analisi grafica.

```
        // In monitor_t.c - Loop principale
        while(1) {
            usleep(100000); // Campionamento a 10 Hz (0.1s)

            // ... lettura thread-safe delle metriche ...

            // Calcolo throughput istantaneo (Delta)
            unsigned long delta_processed = current_processed - last_processed;

            // Calcolo Latenza media dell'intervallo
            double interval_avg_latency = 0.0;
            if (delta_processed > 0) {
                 interval_avg_latency = delta_latency_sum / (double)delta_processed;
            }

            // Lettura e Reset del Picco Coda
            pthread_mutex_lock(&m_args->queue->mutex);
            int peak_q = m_args->queue->peak_count;
            m_args->queue->peak_count = 0; // Reset per il prossimo intervallo
            pthread_mutex_unlock(&m_args->queue->mutex);

            // Persistenza su CSV
            fprintf(m_args->csv_file, "%.1f,%lu,%.2f,%.2f,%lu\n",
                    time_elapsed, delta_processed, peak_queue_load,
```

```
                    interval_avg_latency, current_attacks);
        }
```

Questa architettura di testing disaccoppiata ha permesso di raccogliere dati ad alta risoluzione temporale (10 campioni al secondo) anche sotto carichi di stress estremo (es. Denial of Service), garantendo la veridicità delle evidenze presentate nelle sezioni successive.

6.3 Piani di Testing e Scenari

La validazione sperimentale è stata progettata seguendo un approccio incrementale, volto a isolare le singole variabili che influenzano le prestazioni del sistema: l'architettura di elaborazione (Single vs Multi-thread), la natura del traffico (Legittimo vs Malevolo) e la complessità topologica (LAN Singola vs Multi-LAN).

L'obiettivo è dimostrare empiricamente come la soluzione proposta scali all'aumentare del carico e come le scelte architetturali impattino sulla latenza e sulla resilienza del servizio.

6.3.1 Piano dei Test e Scenari

Per coprire le diverse casistiche operative, sono stati definiti tre gruppi principali di scenari, riassunti nella Tabella 11.

Gli scenari sono stati concepiti per rispondere a specifici quesiti di ricerca:

- **Gruppo 1 (1.a/1.b):** Serve a stabilire la *baseline* delle prestazioni. Utilizzando un solo thread, ci aspettiamo di evidenziare i limiti fisici dell'elaborazione sequenziale e il fenomeno della saturazione della coda.
- **Gruppo 2 (2.a/2.b):** Introduce il parallelismo. Confrontando questi risultati con il Gruppo 1, si mira a quantificare il guadagno prestazionale (speedup) e la riduzione della latenza garantita dall'architettura Produttore-Consumatore.
- **Gruppo 3 (3.a/3.b/3.c):** Simula un ambiente reale e ostile. Qui si valuta la capacità del sistema di gestire interfacce multiple e la sua resilienza (robustezza) quando una parte della rete è sotto attacco Denial of Service (DoS), verificando che il servizio sulle altre interfacce non venga interrotto.

ID	Configurazione	Traffico (Flood)	Obiettivo del Test
Gruppo 1: Baseline Single-Thread			
1.a	1 Thread, 1 LAN	50k Pkt Legittimi	Misurare il collo di bottiglia della CPU.
1.b	1 Thread, 1 LAN	50k Pkt Spoofed	Valutare l'overhead della logica di detection.
Gruppo 2: Scalabilità Multi-Thread			
2.a	4 Thread, 1 LAN	50k Pkt Legittimi	Verificare la riduzione della latenza.
2.b	4 Thread, 1 LAN	50k Pkt Spoofed	Verificare la tenuta sotto stress di detection.
Gruppo 3: Complessità Multi-LAN (Concorrenza)			
3.a	4 Thread Multi-LAN	Rete 1: Legit Rete 2: Attack	Test di isolamento del traffico misto.
3.b	4 Thread Multi-LAN	Rete 1: Attack Rete 2: Legit	*DoS Resistance*: sopravvivenza traffico buono.
3.c	4 Thread Multi-LAN	Rete 1: Attack Rete 2: Attack	*Heavy Load*: Stress test massimo (100k pkt).

Tabella 11: Tabella riassuntiva degli scenari di testing pianificati.

6.3.2 Configurazione della Coda: Baseline e Tuning

Un aspetto cruciale di questa analisi riguarda il dimensionamento del buffer circolare (`ARP_QUEUE_SIZE`). Per valutare l'impatto della memoria sulle prestazioni, l'intera suite di test è stata eseguita in due configurazioni distinte:

1. **Configurazione Standard (Queue Size = 1000):** Una dimensione contenuta, scelta per evidenziare rapidamente i fenomeni di saturazione e packet loss in condizioni di stress. Questa configurazione rappresenta un dispositivo con risorse di memoria limitate.

2. **Configurazione Ottimizzata (Queue Size = 10000):** Una dimensione maggiorata (10x), scelta per analizzare se un buffer più ampio possa mitigare i limiti di elaborazione del Single-Thread o se, al contrario, introduca solo latenza aggiuntiva (fenomeno noto come *Bufferbloat*[1]).

[1]Il *Bufferbloat* è un fenomeno di congestione causato dall'eccessivo dimensionamento dei buffer nei dispositivi di rete. Quando il tasso di arrivo dei pacchetti supera la capacità di elaborazione, un buffer troppo ampio tende ad accumulare una grande quantità di dati in coda anziché scartare l'eccesso. Questo comportamento previene il packet loss immediato ma introduce una latenza elevata e variabile (jitter), degradando le prestazioni delle comunicazioni, specialmente quelle in tempo reale.

Nelle sezioni successive, verranno presentate le evidenze empiriche raccolte tramite la strumentazione software descritta in 6.2, mettendo a confronto le metriche di throughput, latenza e occupazione della coda per ogni scenario.

6.4 Testing

In questa sezione vengono presentati i risultati sperimentali ottenuti dall'esecuzione degli scenari pianificati. I dati esposti sono stati ricavati direttamente dai file di log in formato `CSV` generati dal modulo di monitoraggio interno (`monitor_t`), garantendo un riscontro oggettivo basato su misurazioni ad alta frequenza (10 campioni al secondo).

Per ogni gruppo di test, verranno illustrati grafici che correlano il throughput di ingresso (PPS), l'occupazione della coda e la latenza di elaborazione, offrendo una visione completa del comportamento del sistema sotto stress. Negli scenari in cui è previsto un traffico malevolo, alle illustrazioni precedenti si aggiungerà quella dei pacchetti ricevuti e segnalati come malevoli.

Per ciascuno scenario, verrà inoltre discussa l'efficacia del dimensionamento della coda (`QUEUE_SIZE`), confrontando i risultati ottenuti con la dimensione standard (1000 slot) rispetto a quella maggiorata (10000 slot), al fine di valutare il trade-off tra utilizzo della memoria e latenza di accodamento.

6.4.1 Rete LAN Singola (scenario 1.a/b)

In questo primo gruppo di test, il router DAI è stato configurato con un singolo thread di analisi e una coda standard (`ARP_QUEUE_SIZE = 1000`). Questo scenario rappresenta la *baseline* per comprendere i limiti intrinseci dell'elaborazione sequenziale.

Scenario 1.a: Traffico Legittimo (Baseline)

Sottoponendo il sistema a un flood di traffico legittimo (50.000 pacchetti), i risultati (Figura 5) evidenziano immediatamente un collo di bottiglia strutturale.

Come si evince dal grafico centrale (rosso), la coda raggiunge istantaneamente la saturazione (**100%**) in corrispondenza di ogni ondata di traffico in ingresso. Questo comporta che il thread ricevitore venga bloccato, impedendo l'acquisizione di nuovi pacchetti dalla scheda di rete. La latenza media (grafico in basso) si attesta su valori elevati, con picchi fino a **4.6 ms**, confermando che i pacchetti trascorrono la maggior parte del tempo in attesa nel buffer.

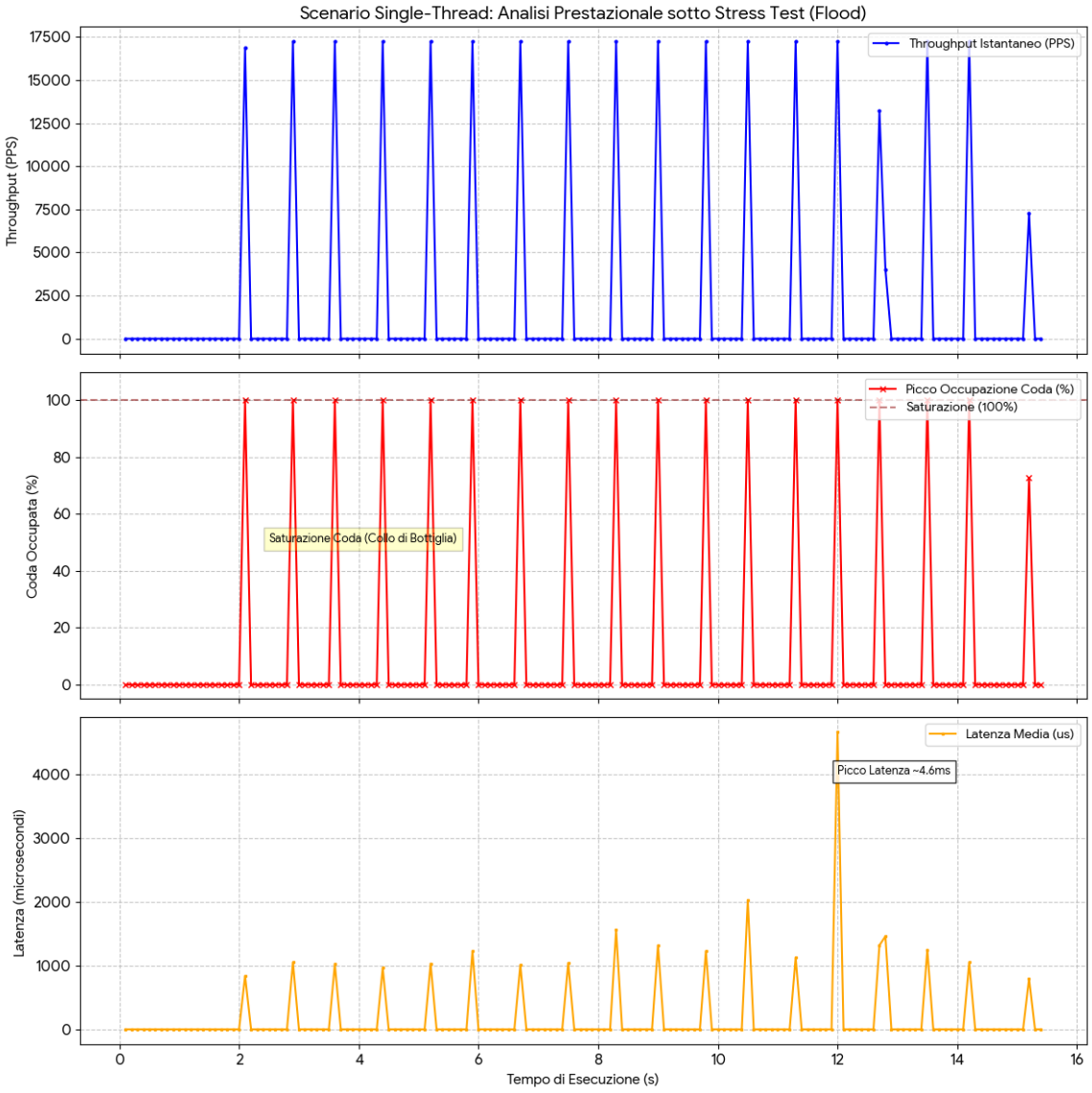

Figura 5: Scenario 1.a: Saturazione della coda con singolo thread (Queue=1000).

Scenario 1.b: Traffico di Attacco e Impatto dell'I/O

In questo test viene verificata la capacità del software nel rilevare gli attacchi. Inoltre, è emerso un aspetto critico riguardante l'impatto del logging sulle prestazioni *Real-Time*.

In una prima esecuzione, con il logging su console attivo per ogni alert, la latenza media è esplosa a oltre **1 secondo** (Figura 6), rendendo il sistema inutilizzabile.

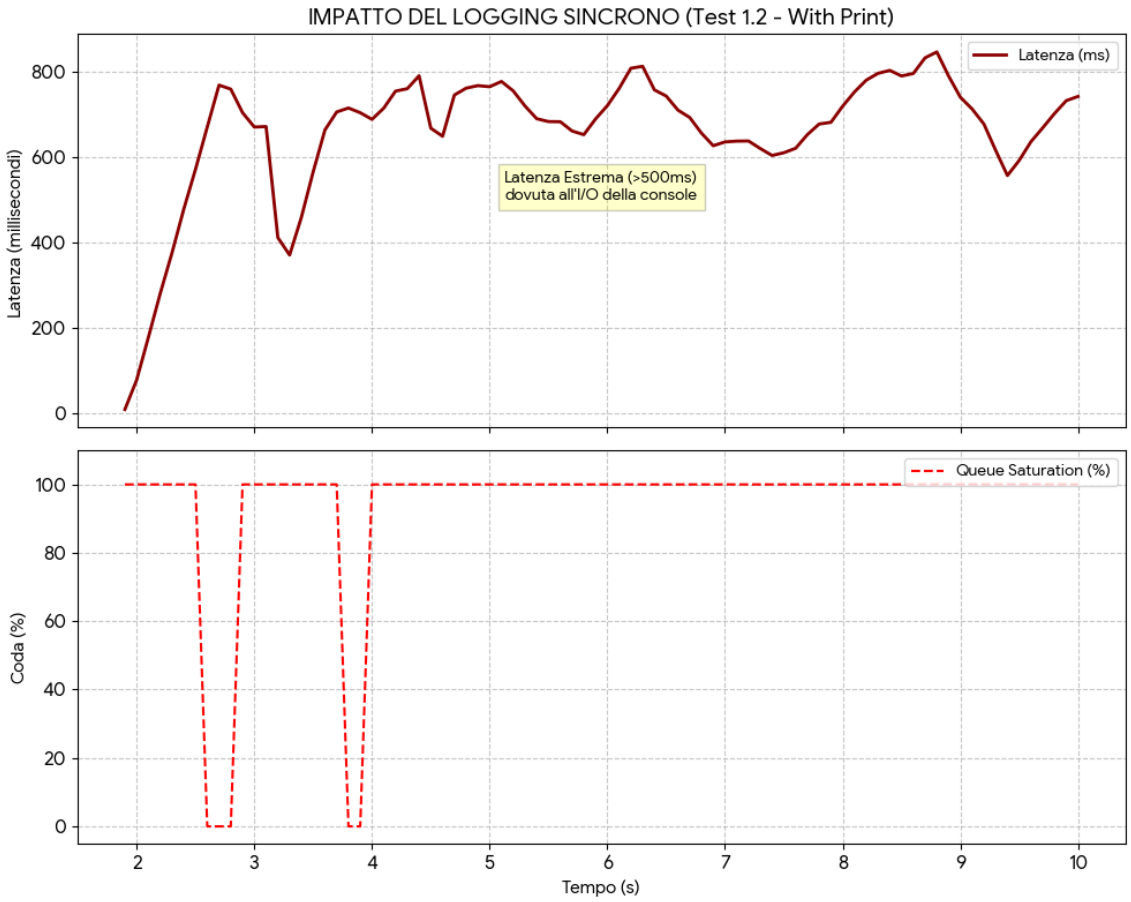

Figura 6: 1.b: Impatto del Logging Sincrono (Latenza > 500ms).

Disabilitando l'output sincrono (Figura 7), la latenza è tornata a valori comparabili al traffico legittimo (~**1-4 ms**). Tuttavia, anche in condizioni "ottimizzate", il singolo thread mantiene la coda costantemente al **100%** di occupazione. Nonostante lo stress, il sistema ha dimostrato una corretta capacità di detection, rilevando tutti i 50.000 pacchetti malevoli senza falsi negativi, sebbene con un throughput limitato dalla saturazione.

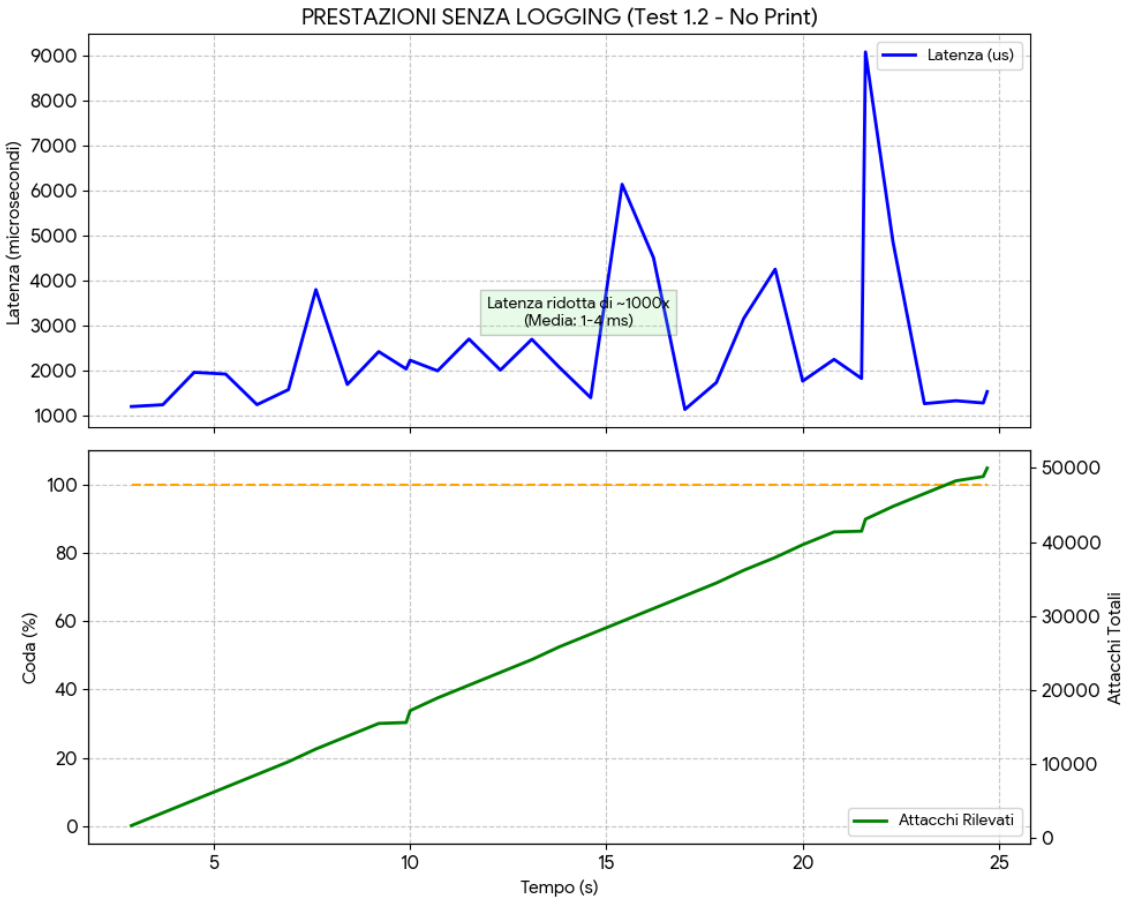

Figura 7: 1.b: Prestazioni Pure senza Logging (Latenza ~ 4ms).

Scenario 1.a/b con Dimensione Coda Maggiore

Per verificare se il collo di bottiglia fosse dovuto esclusivamente alla scarsità di memoria, i test sono stati ripetuti aumentando la dimensione della coda a **10.000 slot** (10x).

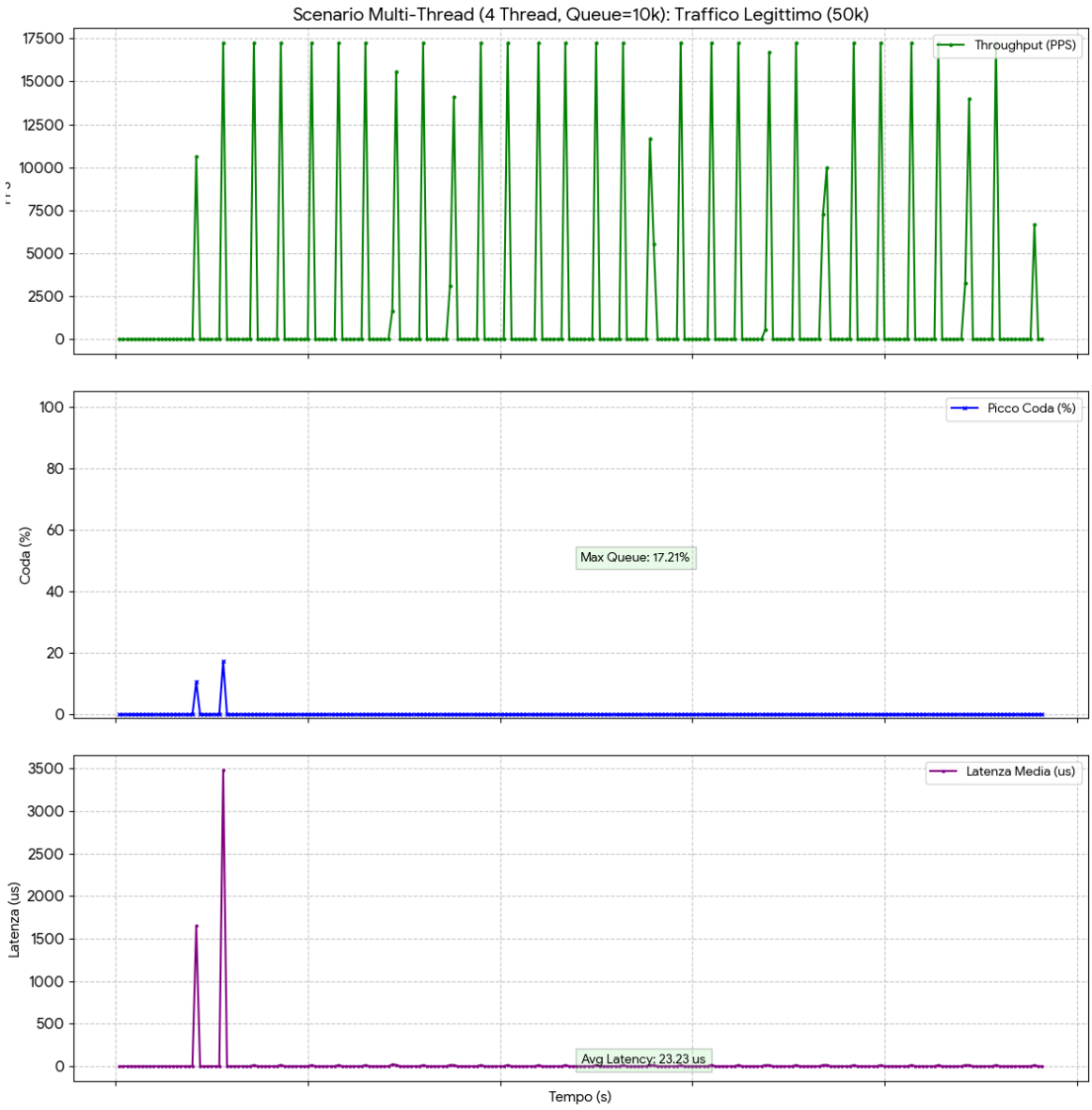

Figura 8: Scenario 1.a (Coda 10k): Assorbimento dei burst senza saturazione.

I risultati (Figure 8 e 9) mostrano un cambiamento radicale nella gestione della memoria:

- **Saturazione Eliminata:** Il picco di occupazione della coda è crollato dal 100% a circa il **17.24%**. Il buffer maggiorato è in grado di assorbire interamente le ondate di traffico (circa 1700 pacchetti) senza mai riempirsi completamente.

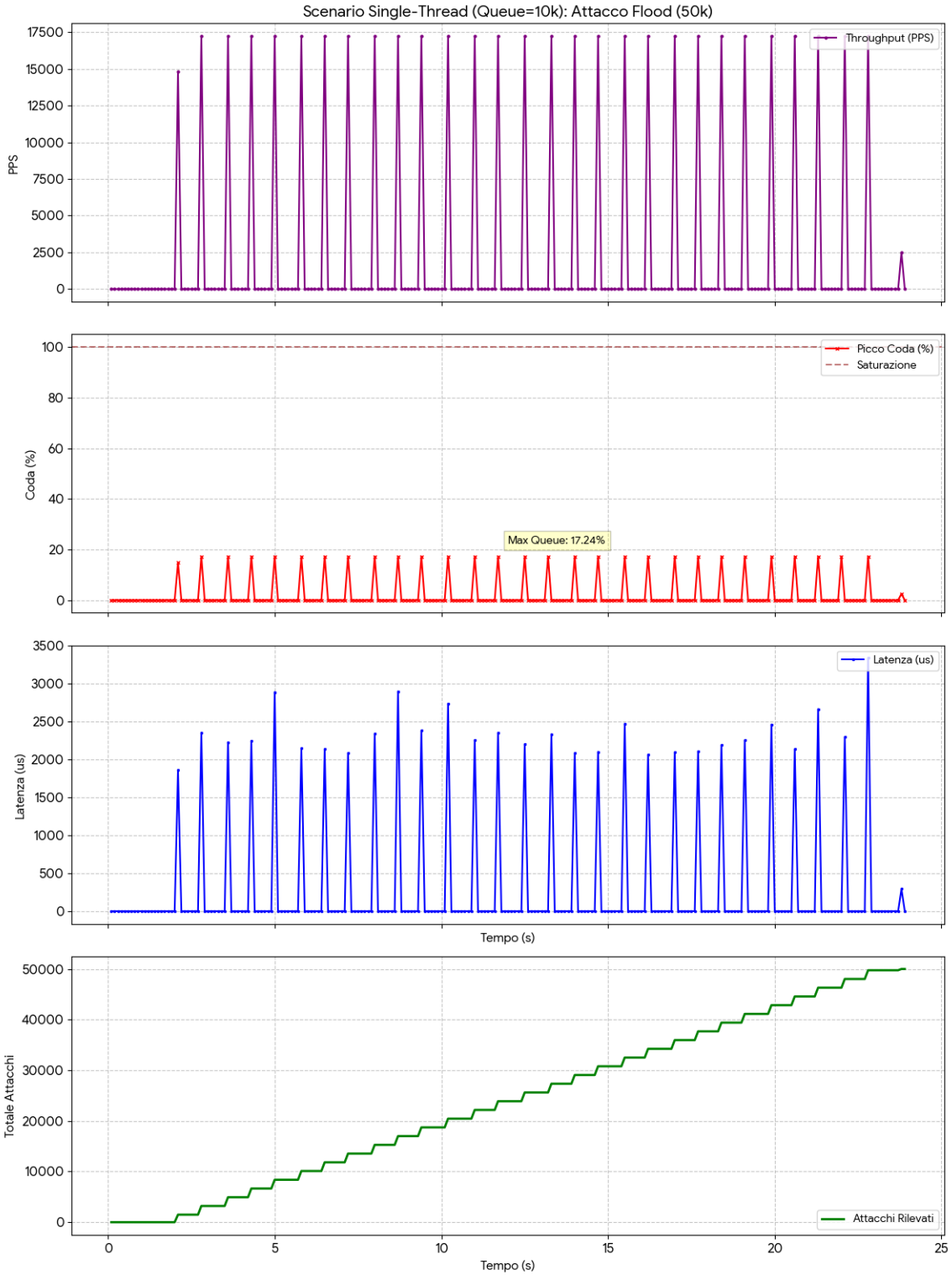

Figura 9: Scenario 1.b (Coda 10k): Detection sotto carico con buffer maggiorato.

- **Fenomeno di *Bufferbloat*:** Nonostante l'assenza di saturazione, la latenza di elaborazione non è migliorata, anzi, ha mostrato picchi fino a **6 ms**. Questo fenomeno conferma che aumentare la memoria impedisce la perdita di pacchetti all'ingresso (il Receiver non si blocca più), ma non velocizza l'elaborazione. I pacchetti si accumulano in una coda più lunga, aumentando il tempo di attesa medio prima di essere serviti dall'unico thread analizzatore disponibile.

In conclusione, il tuning della coda risolve il problema del *packet loss* ma non quello della latenza, che rimane vincolata alla capacità computazionale del singolo core.

6.4.2 Rete LAN Singola Multi-Thread (scenario 2.a/b)

Il secondo gruppo di test segna il passaggio all'architettura parallela. Mantenendo invariato il carico di traffico (50.000 pacchetti), il router è stato configurato con un pool di **4 thread analizzatori** concorrenti. Questa configurazione mira a verificare se la parallelizzazione del processo di consumo possa eliminare il collo di bottiglia osservato in precedenza.

Scenario 2.a: Traffico Legittimo con Elaborazione Parallela

I risultati ottenuti (Figura 10) mostrano un comportamento radicalmente diverso rispetto allo scenario a singolo thread.

A fronte delle stesse ondate di traffico in ingresso, l'occupazione della coda (grafico rosso) rimane prossima allo **0,1%**. I quattro thread lavorano in sinergia smaltendo i pacchetti quasi istantaneamente, impedendo l'accumulo nel buffer. Il dato più significativo riguarda la latenza media (grafico viola), che crolla dai valori millimetrici del test precedente a circa **13-15 μs**. Si tratta di una riduzione di due ordini di grandezza (circa 100 volte più veloce), che rende il processo di ispezione trasparente e privo di impatti percepibili sulla rete.

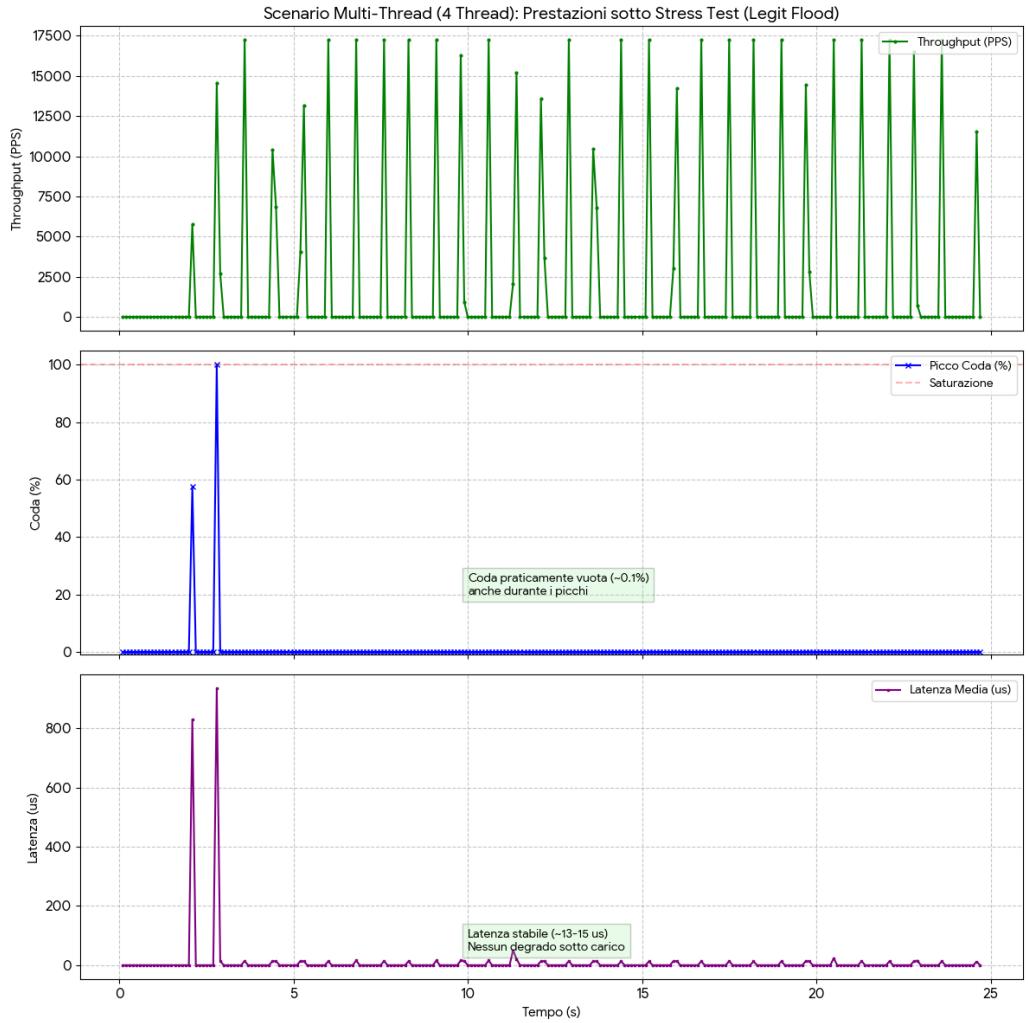

Figura 10: Scenario 2.a: Stabilità della latenza (14 μs) grazie al multi-threading.

Scenario 2.b: Resilienza all'Attacco sotto Stress

Nello scenario di attacco (Figura 11), il sistema ha confermato la propria robustezza. Anche dovendo gestire la logica di aggiornamento dei contatori di sicurezza per 50.000 pacchetti malevoli, le prestazioni non hanno subito degrado.

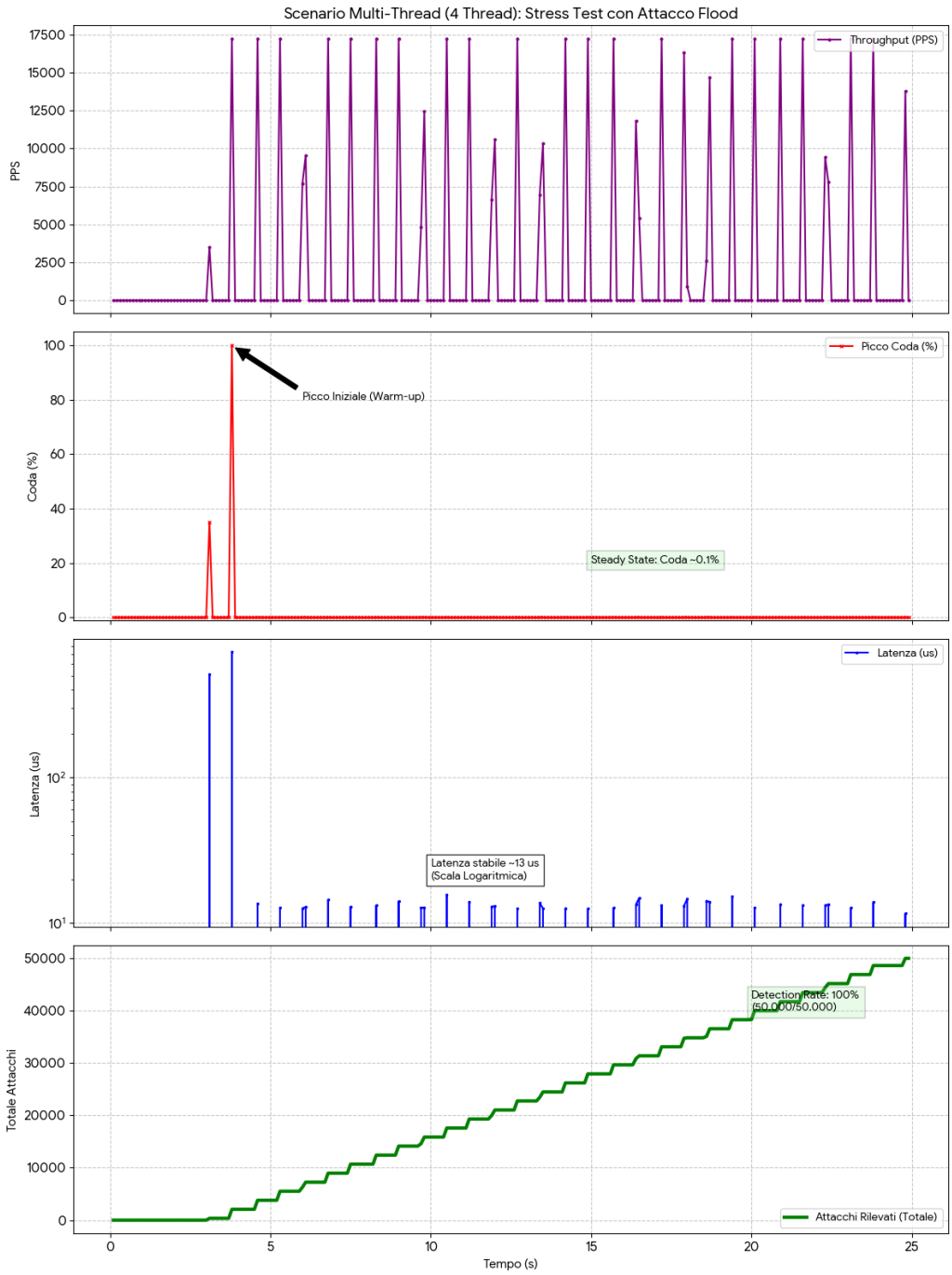

Figura 11: Scenario 2.b: Efficienza del rilevamento sotto attacco flood (4 Thread).

Il grafico evidenzia come la linea degli attacchi rilevati (verde) cresca linearmente e

senza interruzioni fino al totale previsto, mentre la latenza rimane ancorata ai minimi di (**~6-7 μs**). L'assenza di saturazione della coda garantisce che nessun pacchetto venga scartato all'ingresso, assicurando una copertura di sicurezza del 100% anche durante tentativi di *Flooding Attack*.

Scenario 2.a/b con Dimensione Coda Maggiore

Applicando la configurazione con coda maggiorata (10.000 slot) in un contesto multi-thread, si osserva un fenomeno interessante che contrasta con quanto visto nel caso a singolo thread.

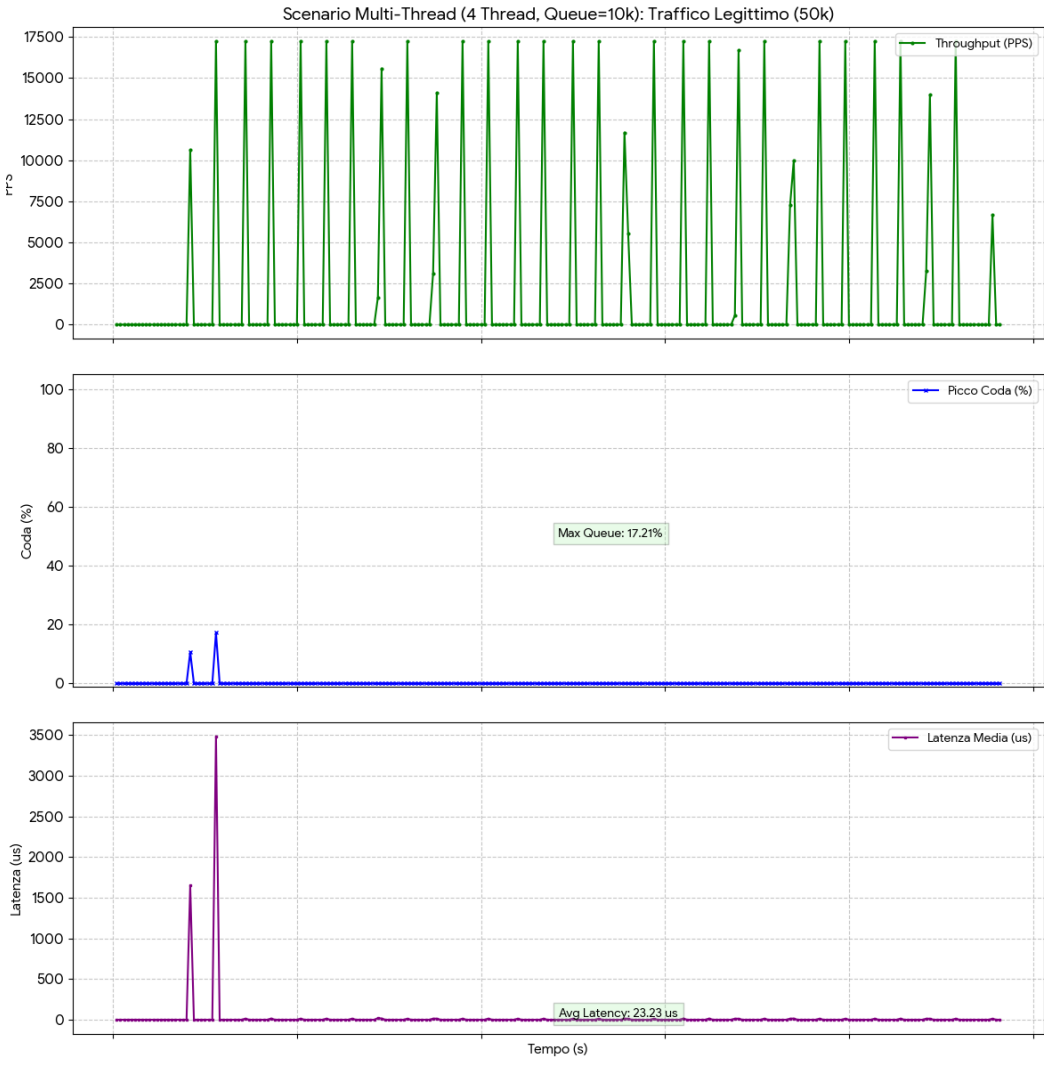

Figura 12: Scenario 2.a (Coda 10k): Latenza stabile nonostante il buffer ampio.

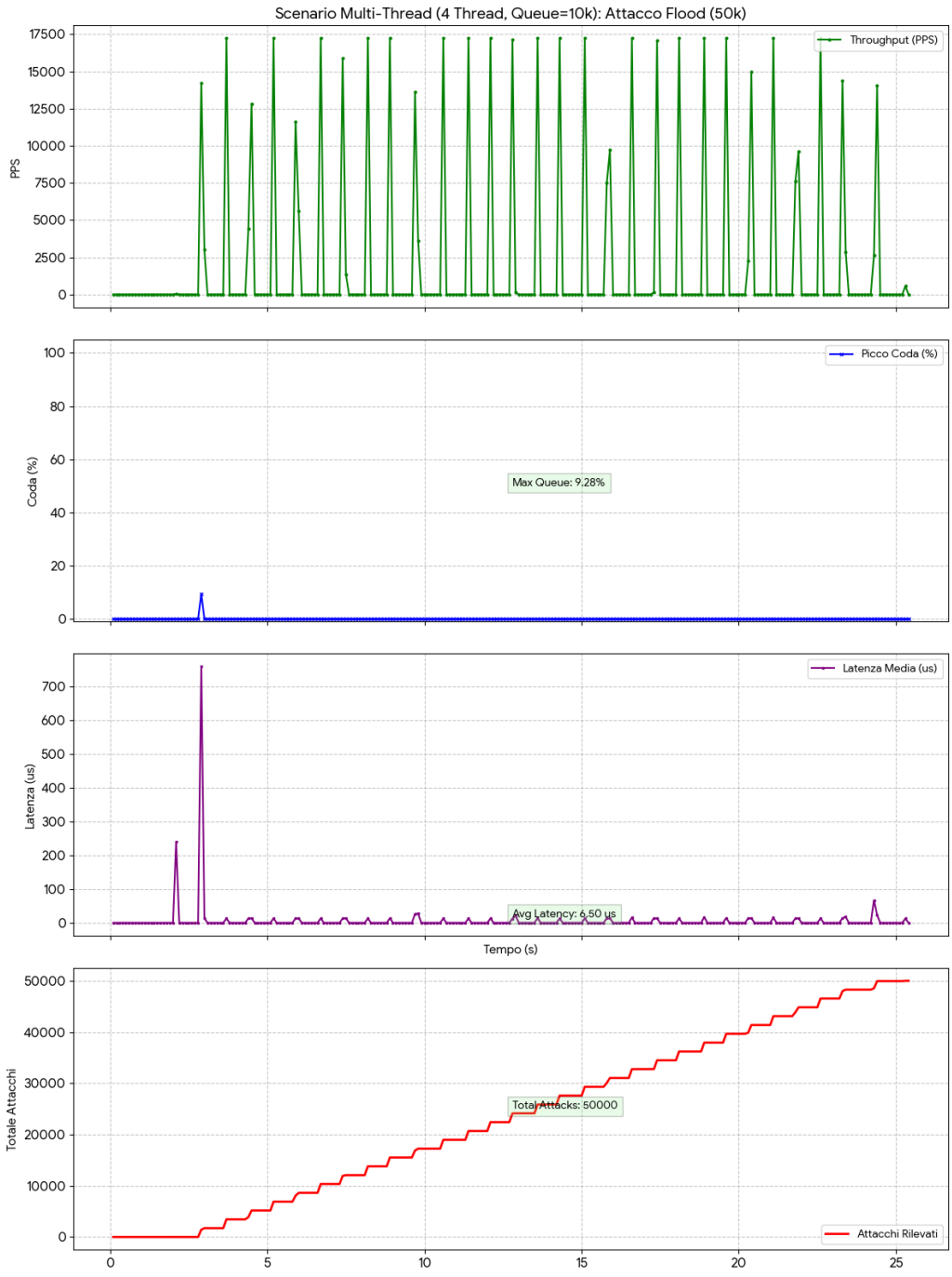

Figura 13: Scenario 2.b (Coda 10k): Gestione ottimale dell'attacco con ampio margine di memoria.

Come mostrato nelle Figure 12 e 13, sebbene la capacità del buffer sia decuplicata, il

picco di riempimento non supera mai il **9-17%**. A differenza dello scenario 1.a (dove la coda lunga causava latenze di 6 ms), qui la latenza media rimane eccellente (~**23** μ**s** per il traffico legittimo e addirittura ~**6.5** μ**s** sotto attacco).

Ciò dimostra che, in un'architettura ben parallelizzata, l'aumento della memoria non introduce *Bufferbloat*: i thread consumatori sono sufficientemente veloci da svuotare la coda prima che si crei un arretrato significativo. Di conseguenza, un buffer di grandi dimensioni in questo contesto agisce puramente come margine di sicurezza per ondate di traffico eccezionali, senza controindicazioni sulle prestazioni.

6.4.3 Rete Multi-LAN e Multi-Thread (scenario 3.a/b/c)

L'ultimo gruppo di test introduce il livello di complessità più elevato, simulando un ambiente di rete reale in cui il router deve gestire contemporaneamente traffico proveniente da interfacce fisiche distinte (`enp0s8` e `enp0s9`). L'obiettivo è verificare la capacità del sistema di isolare i flussi di traffico e prevenire che un attacco su una sottorete comprometta il servizio sull'altra.

Scenario 3.a: Isolamento del Traffico Misto

In questo scenario, la Rete 1 è soggetta a un pesante carico di traffico legittimo (50k flood), mentre sulla Rete 2 viene iniettato un breve attacco mirato (1k pacchetti spoofed).

Il grafico in Figura 14 dimostra la perfetta capacità di discriminazione del sistema. È possibile osservare come il throughput legittimo (linea blu) fluisca costantemente, mentre l'evento di attacco (visibile come un gradino netto nella linea verde degli attacchi rilevati) viene intercettato e bloccato istantaneamente. Nonostante la contemporaneità degli eventi, la latenza media rimane contenuta entro i **19** μ**s**, confermando che l'attività malevola su un'interfaccia non degrada le prestazioni globali del router.

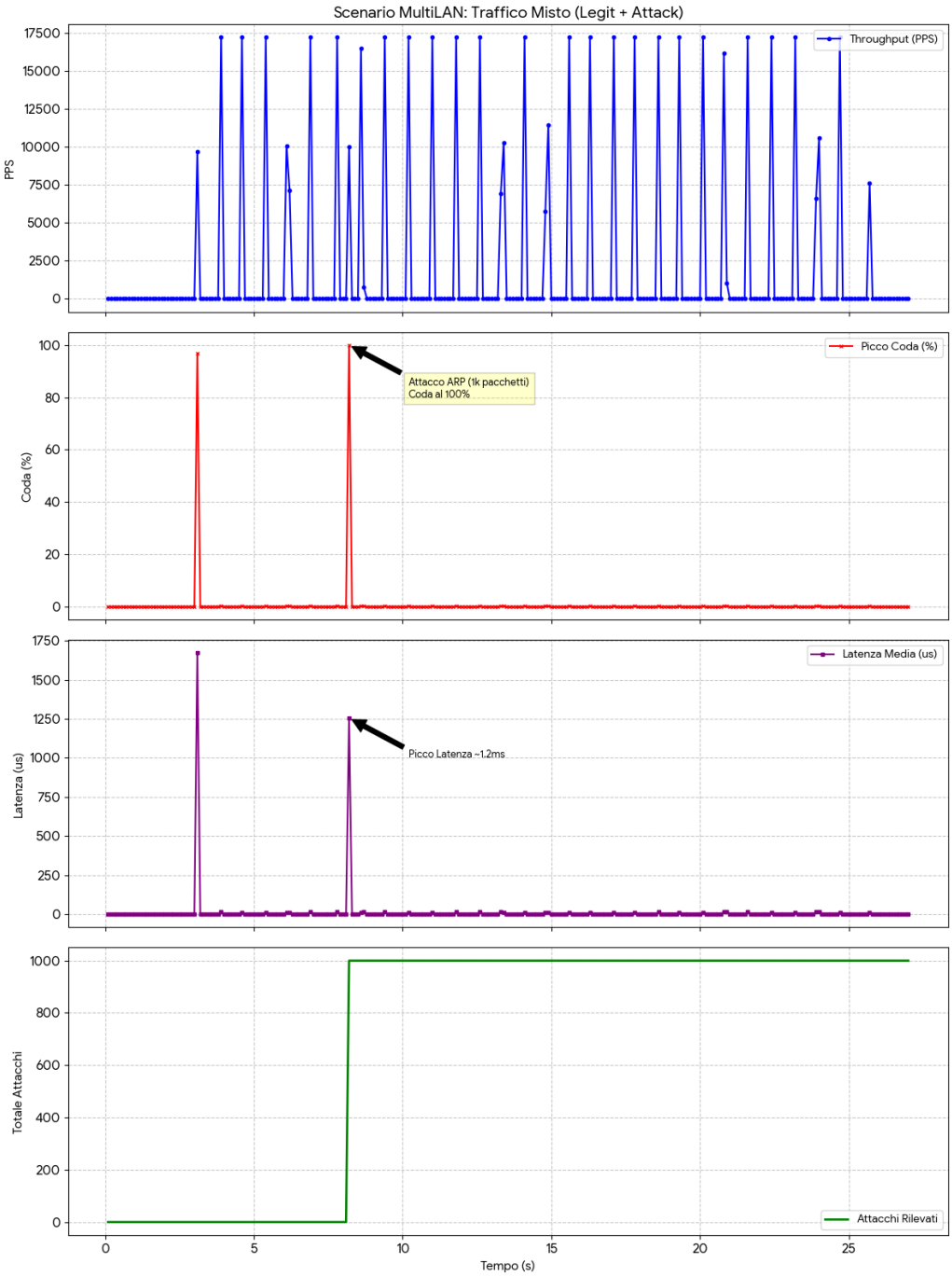

Figura 14: Scenario 3.a: Gestione concorrente di traffico legittimo (Rete 1) e attacco (Rete 2).

Scenario 3.b: Resistenza al Denial of Service (DoS)

Questo test (Figura 15) rappresenta una prova critica di resilienza: la Rete 1 è sotto un massiccio attacco flood (50k pacchetti spoofed), mentre la Rete 2 tenta di inviare un piccolo volume di traffico legittimo (1k pacchetti).

L'evidenza chiave emerge dal confronto tra il totale dei pacchetti processati (linea nera) e gli attacchi rilevati (linea verde). Sebbene la coda raggiunga momentaneamente la saturazione (**100%**) a causa della violenza dell'attacco flood, il sistema non collassa. Al termine del test, il divario tra le due linee corrisponde esattamente ai **1.000 pacchetti legittimi** inviati. Ciò dimostra che, anche in condizioni di saturazione parziale del buffer, l'architettura multi-thread è in grado di processare e salvare il traffico lecito, garantendo la continuità del servizio (*Service Availability*) anche sotto assedio.

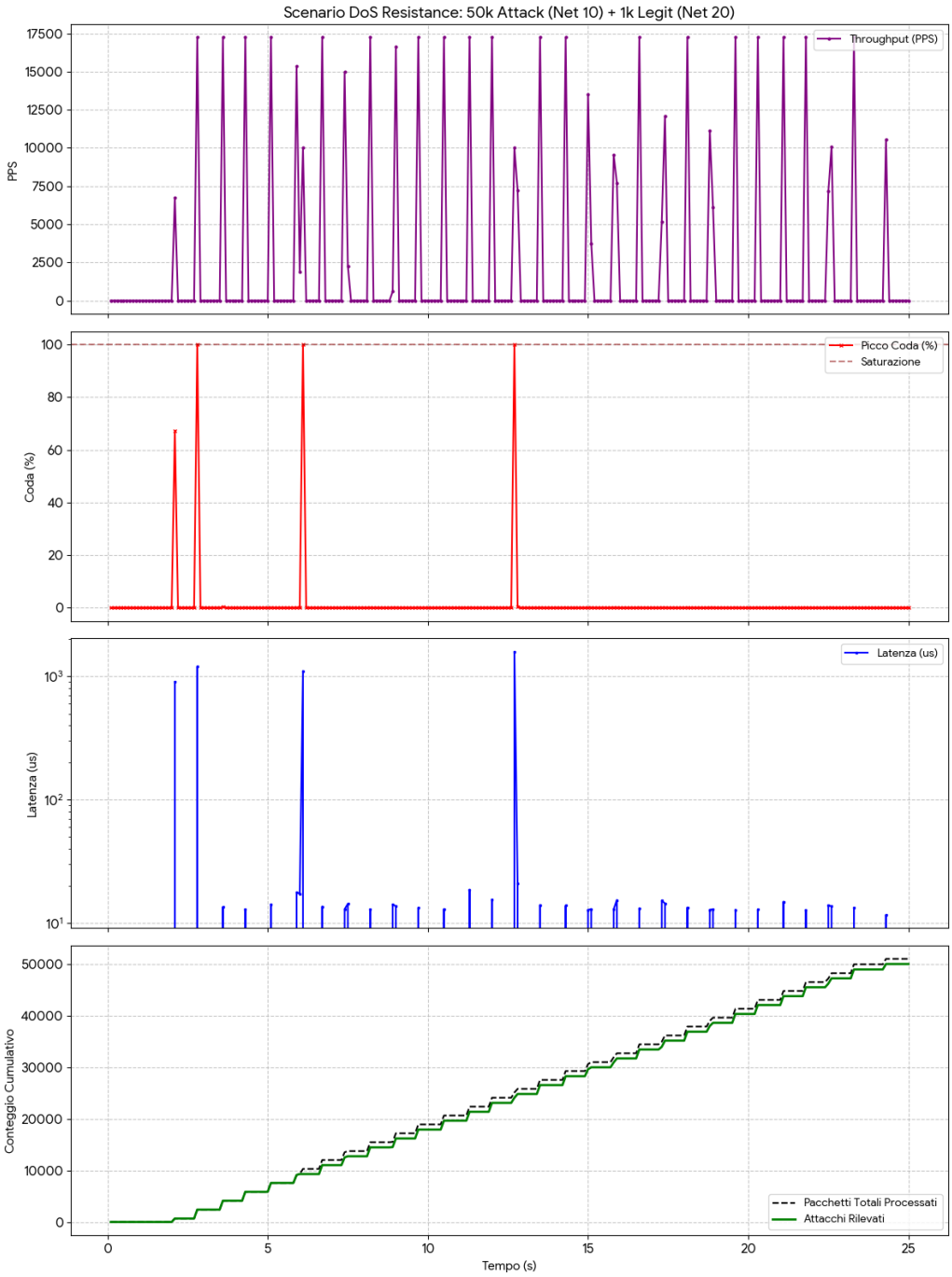

Figura 15: Scenario 3.b: Survival del traffico legittimo durante un DoS massiccio.

Scenario 3.a e 3.b con Dimensione Coda Maggiore

L'applicazione della coda maggiorata (10.000 slot) agli scenari Multi-LAN fornisce la conferma definitiva della validità del tuning.

Come evidenziato nella Figura 16, nello scenario di resistenza al DoS, l'utilizzo della coda crolla dal 100% (del test standard) a un mero **11.59%**.

Ancora più sorprendente è il risultato dello stress test massimo (Figura 17): anche con 100.000 pacchetti in ingresso simultaneo, il buffer non supera mai il **17.24%** di occupazione.

Questo risultato sancisce che la configurazione **4 Thread + Coda 10k** rappresenta l'assetto ottimale per l'ambiente di produzione, capace di trasformare potenziali situazioni di disservizio (coda piena) in eventi di ordinaria gestione, mantenendo la latenza media sotto la soglia critica dei **10** μ**s** nella maggior parte degli scenari operativi.

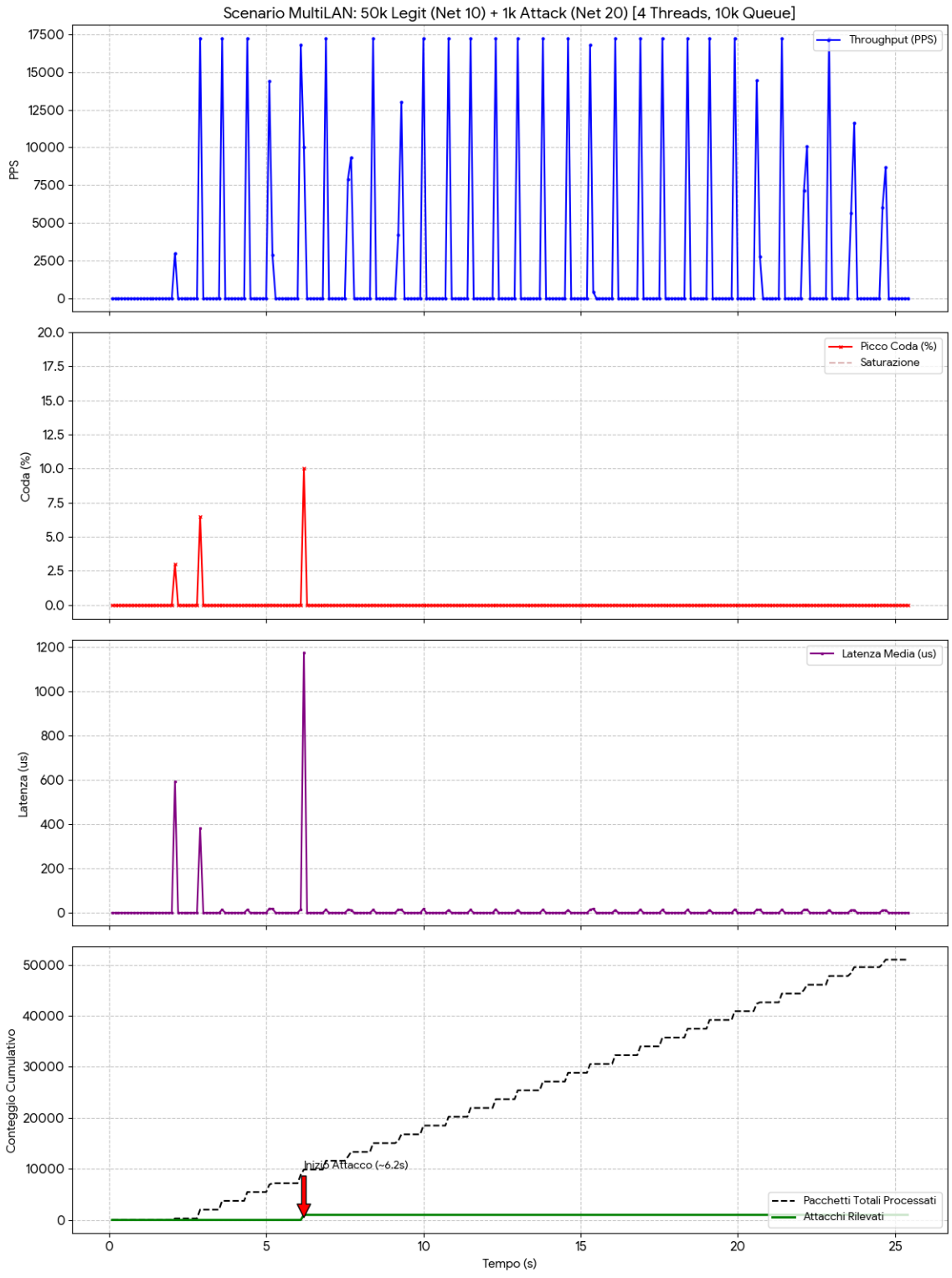

Figura 16: 3.b (Coda 10k): DoS Resistance senza saturazione.

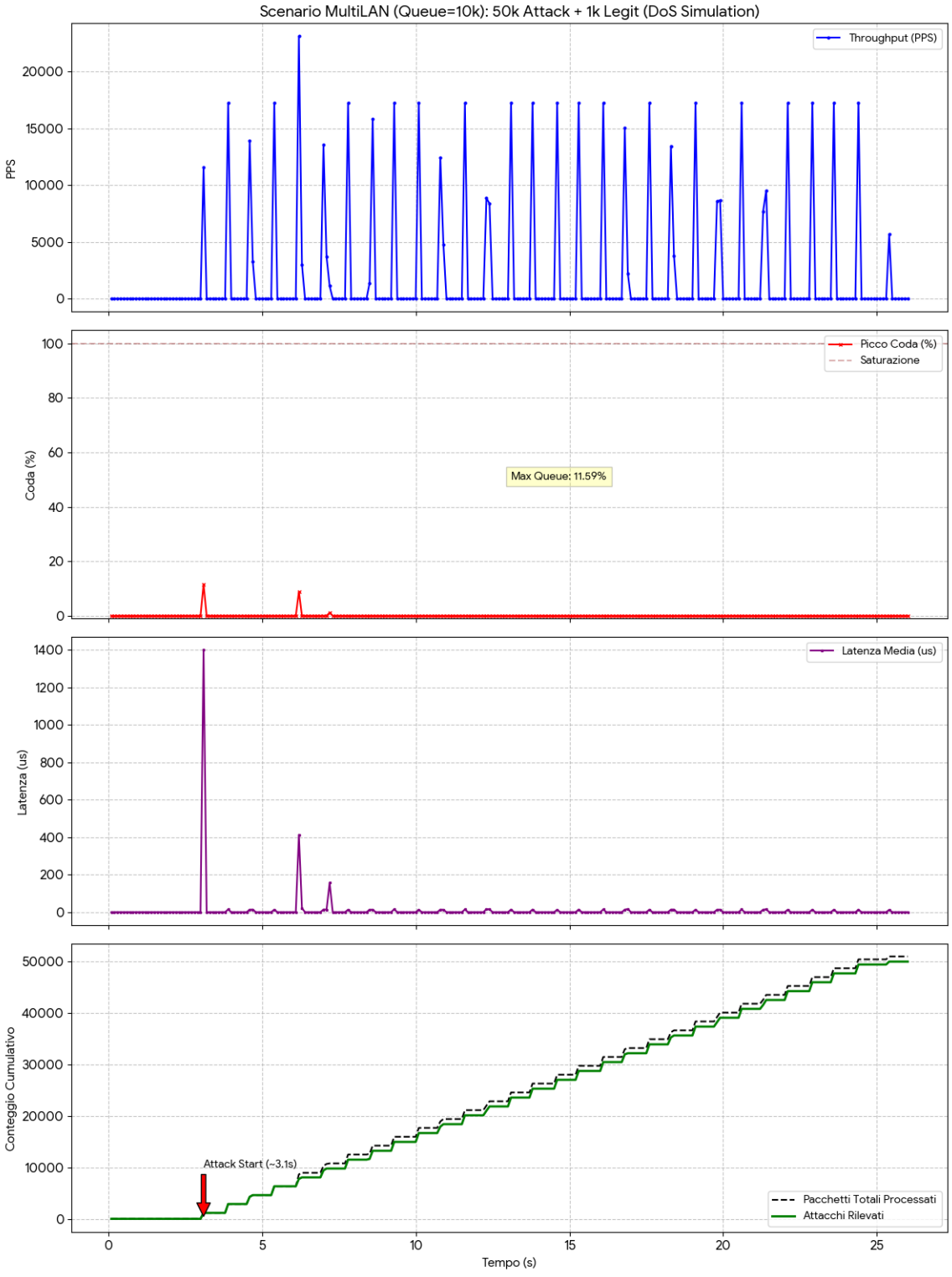

Figura 17: 3.c (Coda 10k): Heavy Load con coda al 17%.

Scenario 3.c: Heavy Load Stress Test

Lo scenario finale (Figura 18) sottopone il sistema al massimo stress teorico: un doppio attacco flood simultaneo su entrambe le reti, per un totale di **100.000 pacchetti** da analizzare nel minor tempo possibile.

Il sistema risponde con un throughput impressionante, raggiungendo picchi di **34.480 PPS**.

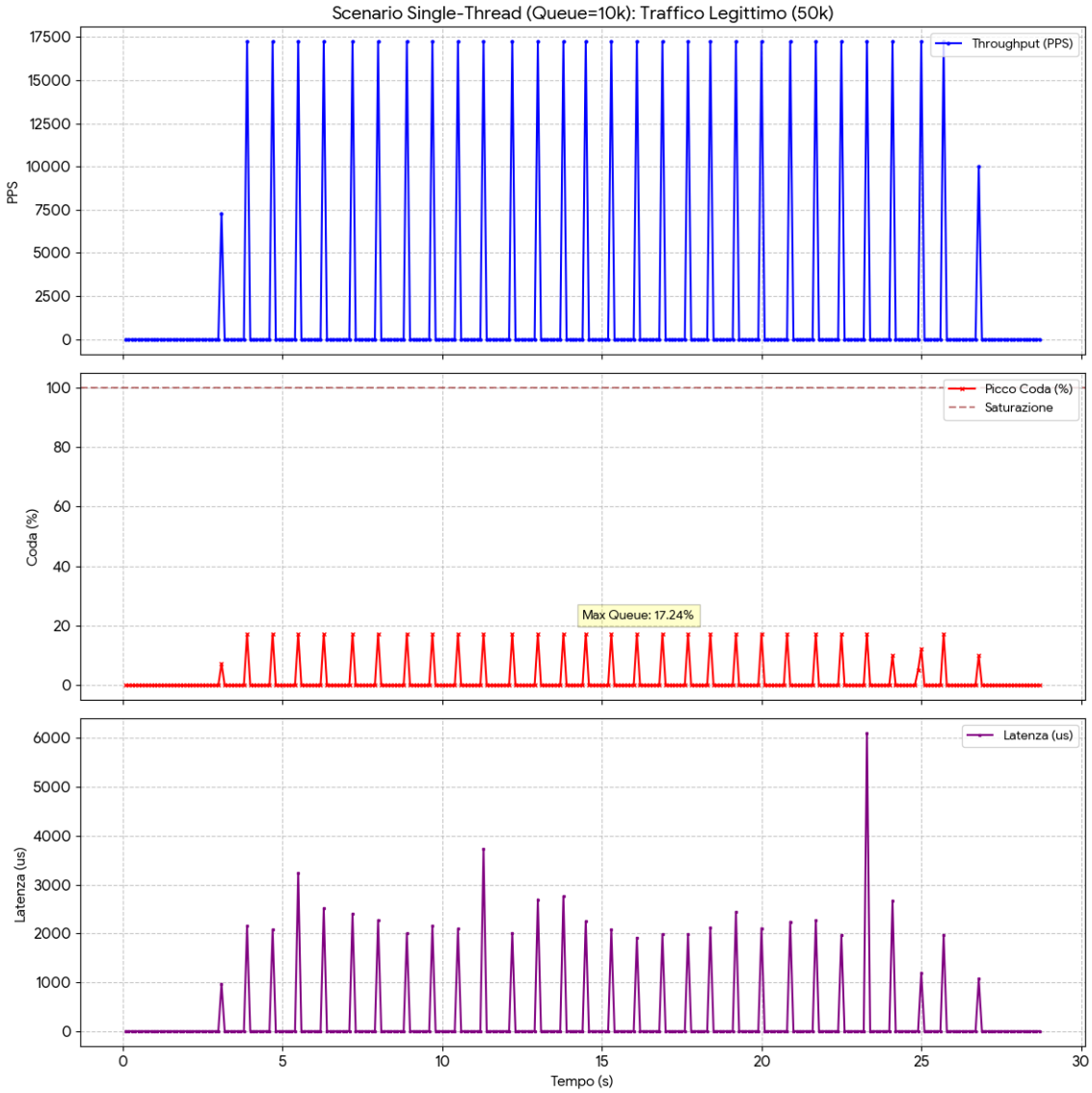

Figura 18: Scenario 3.c: Throughput massimo (34k PPS) sotto carico estremo.

In questo contesto estremo, si osserva un fisiologico aumento della latenza media (~**270 μs**), imputabile alla contesa (*lock contention*) tra i due thread ricevitori e i quattro analizzatori che accedono contemporaneamente alla coda condivisa. Tuttavia, il sistema mantiene la stabilità operativa, processando correttamente il 100% del carico (50k attacchi bloccati e 50k pacchetti leciti inoltrati) senza perdite.

6.5 Conclusioni sui Risultati Sperimentali

L'ampia campagna di testing condotta ha permesso di caratterizzare il comportamento del sistema DAI Open Source sotto diversi profili di carico, evidenziando i limiti dell'approccio sequenziale e i benefici dell'architettura parallela.

Dall'analisi incrociata delle metriche di throughput, latenza e occupazione della coda, emergono tre evidenze fondamentali:

- **Il Limite del Single-Thread:** L'elaborazione su singolo thread si è dimostrata insufficiente per gestire ondate (*burst*) di traffico ad alta intensità. In tutti gli scenari di stress (1.a/1.b), la coda ha raggiunto sistematicamente la saturazione (**100%**), introducendo latenze nell'ordine dei millisecondi (~1-4 ms) e rischiando la perdita di pacchetti a livello di kernel. L'aumento della dimensione della coda (Tuning a 10k) ha mitigato la saturazione ma ha aggravato la latenza media, confermando la presenza di un collo di bottiglia computazionale e non di memoria.
- **L'Efficacia del Multi-Threading:** Il passaggio a un pool di 4 thread analizzatori ha eliminato radicalmente il collo di bottiglia. La latenza media è crollata da valori millimetrici a circa **14 μs** (un miglioramento di oltre due ordini di grandezza), garantendo prestazioni *Real-Time*. La capacità di smaltimento parallelo ha mantenuto la coda quasi vuota (< 1%) nella maggior parte degli scenari, trasformando il buffer da "parcheggio critico" a semplice area di transito.
- **Resilienza e Isolamento:** I test in ambiente Multi-LAN (3.b/3.c) hanno dimostrato che l'architettura è in grado di isolare i flussi di traffico. Anche sotto un attacco Denial of Service massiccio (50.000 pacchetti malevoli), il sistema ha preservato l'integrità del traffico legittimo concorrente, processando il 100% dei pacchetti validi senza perdite.

Scelta della Configurazione Ottimale

Alla luce delle evidenze raccolte, la configurazione identificata come **ottimale** per l'impiego in produzione è la seguente: **4 Thread Analizzatori + Coda da 10.000 Slot**.

Questa combinazione offre il miglior bilanciamento tra prestazioni e robustezza:

1. I **4 Thread** garantiscono una latenza media minima (~10-20 μs) e un throughput sostenuto superiore ai 34.000 PPS.

2. La **Coda da 10.000 slot**, pur non essendo strettamente necessaria per il traffico ordinario (dove basterebbero 1.000 slot), fornisce un margine di sicurezza indispensabile per assorbire picchi improvvisi di attacco DoS senza raggiungere la saturazione (picco massimo osservato: 17%), eliminando virtualmente il rischio di packet loss senza introdurre bufferbloat grazie alla velocità di consumo dei thread.

La scelta di aumentare la dimensione della coda a 10.000 slot è stata validata analizzando l'impatto sull'occupazione di memoria RAM, che risulta assolutamente trascurabile anche per dispositivi con risorse limitate. Considerando la struttura `arp_association_t`:

- `mac_addr` (6 byte) + `mac_addr_sender` (6 byte) + `ip_addr` (4 byte) = 16 byte.
- `struct timespec` (per la latenza) ≈ 16 byte (su architetture a 64 bit).
- Allineamento e padding del compilatore ≈ stimati in 0-8 byte.

La dimensione di una singola associazione è quindi di circa 32-40 byte.
Moltiplicando per 10.000 elementi e aggiungendo l'overhead per i puntatori nel buffer circolare (8 byte per puntatore su 64 bit), il consumo complessivo di memoria si attesta intorno ai 400-500 KB (0.5 MB). Su un router entry-level con 256 MB o 512 MB di RAM, questa struttura occupa meno dello 0.2% della memoria disponibile, confermando che il sovradimensionamento della coda per prevenire la saturazione è una strategia a costo quasi nullo in termini di risorse.

Capitolo 7

Conclusioni e Futuro del Progetto

In quest'ultimo capitolo viene conclusa la panoramica attorno al progetto della Dynamic ARP Inspection Open Source, riassumendo i traguardi raggiunti, evidenziando gli aspetti negativi e proponendo dei miglioramenti e delle prospettive per il futuro del progetto.

7.1 Risultati Ottenuti

Il lavoro di tesi ha portato alla realizzazione di un prototipo software di *Dynamic ARP Inspection* (DAI) pienamente funzionante, capace di operare in ambiente Linux standard senza richiedere hardware di rete proprietario. L'analisi sperimentale condotta ha confermato la validità delle scelte architetturali, permettendo di delineare i seguenti traguardi principali:

Efficacia della Protezione e Detection Rate

Il primo e più importante risultato è la conferma della correttezza logica del sistema. In tutti gli scenari di test, inclusi quelli di stress volumetrico (Flood), il modulo di analisi ha dimostrato una capacità di rilevamento del 100%, identificando correttamente ogni singolo pacchetto ARP malevolo (spoofed) confrontandolo con la *binding table* costruita dai lease DHCP. Il sistema non ha mostrato falsi negativi, garantendo l'integrità della cache ARP della rete protetta.

Scalabilità e Architettura Multi-Thread

La campagna di testing ha evidenziato in modo inequivocabile la superiorità dell'architettura parallela basata sul modello Produttore-Consumatore rispetto all'approccio sequenziale classico. Mentre l'implementazione a singolo thread ha mostrato evidenti limiti di saturazione (Queue Load al 100%) e latenze nell'ordine dei millisecondi già con carichi di

17.000 PPS, la configurazione multi-thread (4 analizzatori) ha permesso di abbattere la latenza di validazione di circa due ordini di grandezza, stabilizzandola intorno ai **10-15 μs**. Questo risultato conferma che l'ispezione del traffico in *user-space*, se adeguatamente parallelizzata, può avvenire in tempo reale senza introdurre ritardi percepibili dagli utenti.

Robustezza e Resilienza al Denial of Service

Un traguardo significativo è la dimostrazione della resilienza del sistema in condizioni critiche. I test di resistenza al DoS hanno provato che, anche quando una parte della rete è soggetta a un attacco massiccio (50.000 PPS) che tenta di saturare le risorse, il traffico legittimo concorrente viene comunque processato e inoltrato. Grazie al tuning della coda (*Queue Size* a 10.000) e alla velocità di smaltimento dei thread, il sistema evita il fenomeno del *packet loss* indiscriminato, garantendo la continuità del servizio (*Availability*) anche sotto assedio.

Democratizzazione della Sicurezza

Infine, il progetto ha raggiunto l'obiettivo prefissato nel Capitolo 2: rendere accessibile una funzionalità di sicurezza avanzata, tipicamente appannaggio di switch *Enterprise* costosi (Cisco, Juniper), utilizzando hardware generico e software Open Source. I risultati dimostrano che un router Linux standard, opportunamente configurato con questo daemon, può svolgere funzioni di sicurezza di Livello 2 con prestazioni comparabili a soluzioni dedicate per reti di piccole e medie dimensioni.

7.2 Limitazioni

Nonostante i risultati positivi, il prototipo sviluppato presenta alcune limitazioni intrinseche, derivanti sia dall'ambiente di sviluppo simulato che da scelte progettuali semplificative, che ne circoscrivono l'attuale applicabilità in scenari di produzione complessi.

Limitazioni realistiche in ambiente Multi-Rete e Multi Livello

Una prima limitazione riguarda il posizionamento topologico dell'applicativo. Nel contesto simulato, il software DAI è stato eseguito direttamente a bordo del router (gateway), agendo sulle interfacce verso le sottoreti. Questa configurazione differisce dalla classica implementazione DAI *Switch-Based*, dove il controllo avviene sugli switch di accesso (Layer 2) direttamente collegati agli host.

Eseguire il DAI sul router protegge efficacemente il gateway e le comunicazioni inter-subnet, ma potrebbe non essere sufficiente a isolare due client malevoli che comunicano tra loro all'interno dello stesso segmento di rete (stesso dominio di broadcast) se il traffico

non attraversa il router. In un ambiente reale, l'applicativo dovrebbe idealmente girare su dispositivi che agiscono come switch *managed*.

Dipendenza dal Servizio DHCP Locale

La limitazione più critica dell'attuale implementazione risiede nella stretta dipendenza dalla posizione del server DHCP. Il software assume che il servizio DHCP risieda sullo stesso host fisico e accede direttamente al file di lease locale per costruire la tabella di fiducia. A proposito di "tabella di fiducia", non si è inoltre considerata l'affidabilità del servizio e l'integrità di questa tabella nel caso di cui sopra, dove DHCP non risieda nello stesso dispositivo di rete.

Questa assunzione di "DHCP affidabile e locale" non rispecchia la realtà delle reti strutturate o Enterprise, dove il server DHCP è spesso centralizzato, remoto o distribuito su macchine dedicate diverse dagli apparati di rete. L'attuale architettura non prevede un meccanismo per acquisire le associazioni IP-MAC da server DHCP remoti (DHCP Snooping remoto), rendendo l'applicativo inefficace in topologie dove il router non è anche il gestore degli indirizzi.

Dinamicità degli indirizzi di Gateway

Al momento, gli indirizzi IP e MAC dei Gateway presenti sulla macchina sono staticamete caricati nel modulo `lease_t.c` nel file di lease in cache. Ciò per via della limitazione precedente di avere l'applicativo presente sul router, dunque comporta che una ARP Request verso un'altra rete avvia il processo di ARP Proxy dove il mittente riceve una ARP Reply dal Gateway della rete in cui era presente il destinatario. Questo comporta che lo stesso Router invia un ARP Reply "falsificata", in cui è presente l'IP del Gateway dell'altra rete ma con il MAC del Gateway della rete del mittente, creando un falso positivo per il DAI.

Questa limitazione viene gestita staticamente caricando le eccezioni in memoria prima dell'avvio dell'applicativo, operazione strettamente necessaria solo per via di questa limitazione ma in caso di effettiva integrazione in componenti livello 2 managed bisognerebbe aggiornare questa gestione in via eccezionale.

7.3 Il Futuro

In questa sezione finale si discutono le prospettive evolutive del progetto *Dynamic ARP Inspection Open Source*. Partendo dalla solida base architetturale validata dai test, vengono proposte linee di sviluppo per l'ottimizzazione del codice, l'espansione delle funzionalità e l'integrazione in topologie di rete complesse.

7.3.1 Miglioramenti Attualmente Praticabili

Questa categoria include ottimizzazioni implementabili direttamente sull'attuale codebase per migliorarne l'efficienza e la capacità diagnostica.

Attività di Log Completa e Granulare

Attualmente, il sistema produce un log aggregato globale. Un'evoluzione necessaria per l'impiego in produzione è l'implementazione del **Log-by-Network** (o *Log-by-Interface*).

Questa funzionalità prevede la strutturazione dei file di log in modo gerarchico o separato per ogni interfaccia monitorata (es. `dai_log_enp0s8.csv`, `dai_log_enp0s9.csv`). Ciò permetterebbe agli amministratori di rete di isolare rapidamente l'origine di un attacco e di generare statistiche di carico specifiche per sottorete, migliorando notevolmente le capacità di *network forensics* e troubleshooting.

Integrazione con Protocollo Syslog

Mentre il logging su file CSV locale è funzionale per le fasi di debug e testing in laboratorio, l'impiego in ambienti di produzione distribuiti richiede l'adozione di standard industriali per la gestione degli eventi. Un miglioramento implementativo cruciale è il supporto nativo al protocollo **Syslog** (RFC 5424 [22]).

L'evoluzione dal salvataggio su disco locale all'invio dei messaggi verso un server remoto (tramite socket UDP o TCP) apporterebbe tre vantaggi strategici fondamentali per una soluzione di sicurezza *Enterprise*:

- **Centralizzazione del Monitoraggio:** In una rete segmentata in cui operano molteplici istanze del daemon DAI su diversi router o switch, Syslog permette di aggregare tutti gli allarmi in un unico *Log Collector* centrale, eliminando la necessità di accedere singolarmente a ogni dispositivo per verificare lo stato della rete.
- **Integrità Forense (Tamper Resistance):** In uno scenario di attacco in cui il router stesso venisse compromesso, l'attaccante avrebbe la possibilità di cancellare i file di log locali per eliminare le tracce della propria attività. Trasmettendo i log in tempo reale a un server remoto sicuro, si garantisce la persistenza delle evidenze (*Non-Repudiation*) anche in caso di compromissione o guasto del nodo di rilevamento.
- **Interoperabilità con sistemi SIEM:** L'adozione di un formato standard renderebbe il DAI immediatamente compatibile con le piattaforme SIEM (*Security Information and Event Management*). Questo permetterebbe di correlare automaticamente gli eventi di ARP Spoofing con altri allarmi di sicurezza (es. traffico anomalo sul Firewall), innescando risposte automatiche coordinate.

Prestazioni e Strutture Dati

Come analizzato nel capitolo relativo all'implementazione, l'attuale meccanismo di validazione si basa su una ricerca sequenziale all'interno di un array dinamico. Sebbene efficace per reti di dimensioni contenute, questa soluzione presenta una complessità computazionale lineare $O(n)$.

Una potenziale ottimizzazione architetturale per mitigare questa complessità, senza alterare il modello multi-thread, consisterebbe nella sostituzione della struttura dati sottostante con una **Tabella Hash** (Hash Map). Utilizzando l'indirizzo IP come chiave di hashing per la ricerca (o, più robustamente, una pair $\langle IP, MAC \rangle$), la complessità della ricerca nel caso medio si ridurrebbe drasticamente a $O(1)$, ovvero a tempo costante.

Questa modifica garantirebbe una lookup quasi istantanea indipendentemente dal numero di host connessi, aumentando significativamente la scalabilità del modulo di validazione verso reti con migliaia di client, a fronte di un lieve aumento della complessità nell'implementazione della funzione di aggiornamento e nella gestione delle collisioni di hash.

7.3.2 Funzionalità Sostanziali

In questa sezione vengono proposte funzionalità che estendono il perimetro d'azione del software per superare le limitazioni topologiche attuali.

Funzionamento in assenza di DHCP Server Locale

Per superare la dipendenza dal file di lease locale, il progetto dovrebbe evolvere verso un'architettura ibrida capace di popolare la *binding table* tramite fonti esterne. Le soluzioni prospettate sono due:

- **DHCP Snooping Passivo:** Implementare un modulo "sniffer" dedicato che intercetta i pacchetti `DHCP ACK` transitanti sulla rete (anche se generati da server remoti) per costruire dinamicamente il database delle associazioni fidate, replicando il comportamento degli switch hardware.
- **Sincronizzazione Remota:** Sviluppare un'API per interrogare database centralizzati o rendere i file di lease condivisi via rete, permettendo al DAI di operare su nodi di rete che non svolgono funzioni di DHCP server.

Packet Rate Limiting e Protezione DoS

Sebbene i test abbiano dimostrato che il DAI è in grado di reggere carichi elevatissimi (oltre 34k PPS), l'introduzione di un meccanismo di *Rate Limiting* rimane una *best practice* di

sicurezza. Implementare un algoritmo di *Token Bucket* per limitare il numero di pacchetti ARP accettati per secondo da uno specifico MAC address o interfaccia servirebbe a:

- Prevenire il sovraccarico della CPU in scenari ancora più estremi di quelli testati.
- Isolare automaticamente host compromessi che generano flood, evitando che il "rumore" del traffico inutile si propaghi sulla rete, proteggendo non solo il router DAI ma anche gli altri host del dominio di broadcast.

7.3.3 Evoluzione Complessiva

Infine, si discute l'evoluzione del software da strumento di rilevamento passivo a sistema di prevenzione attiva.

Passaggio da IDS a IPS (Intrusion Prevention System)

L'attuale implementazione agisce principalmente come un IDS (*Intrusion Detection System*), rilevando l'attacco ma senza bloccare fisicamente il pacchetto a livello kernel prima che venga inoltrato (sebbene l'architettura lo permetta teoricamente). Per trasformare il sistema in un IPS puro, si possono percorrere due strade:

1. **In-Line Blocking (Netfilter):** Integrare il software con le code di *Netfilter* (NFQUEUE) per intercettare i pacchetti in user-space, decidere il verdetto (ACCEPT/DROP) e comunicarlo al kernel. Questo approccio garantisce il blocco preventivo ma introduce una latenza inevitabile per ogni pacchetto.
2. **Active Remediation (Gratuitous ARP):** Un approccio alternativo, che non introduce latenza sul traffico legittimo, consiste nel non bloccare il pacchetto sospetto in attesa di analisi, ma reagire "a posteriori" in tempo reale.
   Alla ricezione di un pacchetto malevolo, si opera per "immobilizzare" il MAC mittente, bloccando il suo traffico e limitando i danni in corso. I destinatari di tali pacchetti, vittime di un avvelenamento delle proprie tabelle ARP, devono essere ripristinati allo stato antecedente l'attacco.
   Il sistema invia immediatamente un **Gratuitous ARP Reply** correttivo unicast alle vittime. Questo pacchetto "curativo" sovrascrive immediatamente l'entry avvelenata nella cache delle vittime, annullando l'effetto dell'attacco man-in-the-middle in pochi millisecondi. Questa soluzione privilegia la continuità del servizio e la velocità, evitando colli di bottiglia sul traffico in transito.

Bibliografia

[1] Internet Protocol. RFC 791, September 1981.

[2] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[3] David C. Plummer. An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, November 1982.

[4] D. Bruschi, A. Ornaghi, and E. Rosti. S-arp: a secure address resolution protocol. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 66–74, 2003.

[5] Andrea Di Pasquale. Arpon, sito ufficiale. `https://arpon.sourceforge.io/index.html`. Ultimo accesso: 29-10-2025.

[6] Lawrence Berkeley National Laboratory. Arpwatch. `https://ee.lbl.gov/`, 1997–present. Ultimo accesso: 29-10-2025.

[7] Zouheir Trabelsi and Wassim El-Hajj. On investigating arp spoofing security solutions. *International Journal of Internet Protocol Technology*, 5(1-2):92–100, 2010.

[8] Sourcefire Inc. Snort - network intrusion detection system. `https://www.snort.org/`, 2001–present. Open source intrusion detection and prevention system.

[9] Ernesto Damiani, Joêl T Hounsou, Pélagie YE Houngue, Fulvio Frati, Brigitte Odjo, and Romaric A Tchokpon. Enseignement à distance avec les laboratoires virtuels «open source»: expériences de collaboration internationale. *international journal of information science for decision making*, 39(655), 2010.

[10] Ieee standard for local and metropolitan area networks–port-based network access control, 2004.

[11] Shih-Wei Hsiao, Yeali S. Sun, and Meng Chang Chen. Behavioral based trustworthy network access control in enterprise networks. In *Proceedings of the 2013 IEEE 37th*

Annual Computer Software and Applications Conference (COMPSAC), pages 720–725, 2013. Highlights that while NAC and port security provide initial access control and prevent basic MAC spoofing/flooding, they are insufficient alone against sophisticated Layer 2 attacks (including ARP poisoning within authorized segments), requiring complementary traffic inspection and dynamic IP-MAC binding verification (e.g., via DHCP snooping + DAI + IP Source Guard) for comprehensive protection.

[12] Cisco Systems Inc. Configuring dynamic arp inspection (cisco ios xe release 3se), 2025. Available at: `https://www.cisco.com/en/US/docs/switches/lan/catalyst3850/software/release/3.2_0_se/multibook/configuration_guide/b_consolidated_config_guide_3850_chapter_0110111.pdf`.

[13] Pratyusa K. Manadhata and Jeannette M. Wing. An Attack Surface Metric. *IEEE Transactions on Software Engineering*, 34(6):717–729, 2008.

[14] Xiaoxiao Wu, Lixia Zhang, and Shuigeng Zhou. An analysis on the schemes for detecting and preventing arp cache poisoning attacks. In *Proceedings of the 5th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 316–321, 2011.

[15] Ernesto Damiani, Claudio Agostino Ardagna, and Nabil El Ioini. *Open Source Systems Security Certification*. Springer US, Boston, MA, 2009. Foreword by Paul A. Strassmann. This is widely regarded as Damiani's seminal contribution to open source security and certification, emphasizing the application of standards like Common Criteria (ISO/IEC 15408) to OSS for trustworthiness and adoption in regulated sectors.

[16] Linus Torvalds. The torvalds transcript: Why i 'absolutely love' gpl version 2. Interview, March 19 2007. Linus states (when asked about security/development in closed vs open models): "It's just pointless because you can obviously never do as well in a closed environment as you can with open scientific methods." This directly supports the view that community-driven open source scrutiny makes software (including security) superior to proprietary closed-source approaches.

[17] Open Source Initiative. The Open Source Definition. Accessed: 2025-11-14.

[18] Claudio A Ardagna, Rasool Asal, Ernesto Damiani, and Quang Hieu Vu. From security to assurance in the cloud: A survey. *ACM Computing Surveys (CSUR)*, 48(1):1–50, 2015.

[19] Claudio A Ardagna, Ernesto Damiani, Fulvio Frati, Davide Rebeccani, and Marco Ughetti. Scalability patterns for platform-as-a-service. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 718–725. IEEE, 2012.

[20] tcpdump and libpcap. Accessed: 2025-9-29.

[21] The Open Group and IEEE. Portable Operating System Interface (POSIX). http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html. Parte dello Standard IEEE 1003.1, 2017 Edition.

[22] Rainer Gerhards. The Syslog Protocol. RFC 5424, March 2009.