ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea in Informatica

Progettazione e sviluppo di un'infrastruttura IaC per competizioni CTF A/D

Relatore:
Prof. Dr. Marco Prandini

Presentata da:
Emanuele Argonni

Correlatori:
Prof. Dr. Andrea Melis

II Sessione
Anno accademico 2024/2025

Abstract

Negli ultimi anni, la digitalizzazione ha determinato un inevitabile aumento degli attacchi informatici, rendendo la cybersecurity una priorità globale per la protezione dei dati e dei sistemi digitali. Le competizioni *Capture The Flag* in modalità *Attack and Defense* rappresentano uno strumento formativo efficace per preparare i futuri professionisti della sicurezza informatica. Tuttavia, l'organizzazione di tali eventi presenta una complessità tecnica intrinseca che ne limita la diffusione.

Il presente lavoro di tesi si propone di rimuovere gli ostacoli tecnici, offrendo una soluzione per automatizzare la creazione dell'infrastruttura necessaria a ospitare competizioni A/D. Attraverso l'utilizzo di strumenti *Infrastructure as Code (IaC)*, il progetto orchestra la creazione e la configurazione di tutti i componenti necessari per una gara: macchine virtuali per i team, router per la rete di gioco, server di gioco e connettività VPN per l'accesso remoto dei partecipanti.

L'architettura sviluppata si distingue dalle soluzioni esistenti per la capacità di distribuire le macchine virtuali su più nodi fisici di un cluster Proxmox, sfruttando al meglio le risorse computazionali disponibili e migliorando la scalabilità. Inoltre, particolare attenzione è stata rivolta alla normalizzazione del traffico di rete, inclusa la mitigazione del *TCP Timestamp fingerprinting*, una vulnerabilità che potrebbe compromettere l'equità della competizione.

Il corretto funzionamento dell'infrastruttura è stato verificato mediante una sessione di prova con 24 partecipanti, durante la quale il sistema ha dimostrato prestazioni adeguate anche con hardware non recente. Questo lavoro fornisce un contributo concreto alla formazione in cybersecurity, semplificando l'organizzazione di competizioni A/D e promuovendo l'apprendimento di competenze pratiche in un ambiente controllato.

Indice

Elenco delle Figure

Elenco dei Codici

Capitolo 1

Introduzione

I sistemi digitali e le reti informatiche, nel tempo, sono diventati parte integrante della società moderna. Oggigiorno siamo costantemente circondati da dispositivi connessi, dagli smartphone agli oggetti domestici intelligenti, fino a sistemi complessi che gestiscono servizi essenziali come la sanità, i trasporti e le banche. La crescente digitalizzazione ha ampliato i vettori di attacco informatico che minacciano l'integrità, la riservatezza dei dati e la disponibilità delle infrastrutture critiche.

La cybersecurity rappresenta oggi una delle principali emergenze nel panorama europeo. I risultati nel nostro paese non sono per niente positivi: l'Italia è al secondo posto tra i paesi più colpiti dell'Unione Europea.

Questo stato di emergenza si traduce in un aumento di cybercrimini, quali ransomware, phishing che mirano al furto di informazioni sensibili e attacchi DDoS che mirano a bloccare l'operatività di aziende e servizi, causando ingenti danni economici.

Purtroppo, molto spesso, l'anello debole di questi eventi drammatici è l'utente "medio" che, inconsapevolmente, si lascia adescare dalle vulnerabilità in rete.

L'unico modo per proteggersi realmente dalle minacce in costante evoluzione è accrescere nelle persone la consapevolezza dei rischi legati alla digitalizzazione.

La sicurezza informatica gioca quindi un ruolo cruciale nella difesa proattiva dei sistemi digitali e richiede figure professionali in grado di identificare e reagire alle minacce emergenti.

In questo contesto, si inseriscono le Capture The Flag (CTF), competizioni informatiche che, con un approccio ludico, permettono alle nuove generazioni di avvicinarsi ai temi della cybersecurity. In particolare, le competizioni in modalità Attack and Defense (A/D) non solo mettono alla prova le competenze tecniche di attacco e difesa dei partecipanti in un ambiente controllato, ma offrono anche un'esperienza di gara che simula le dinamiche del mondo reale.

Tuttavia, questa tipologia di eventi non è particolarmente diffusa, in quanto richiede competenze tecniche specifiche da parte degli organizzatori e risorse economiche importanti da investire nell'hardware.

Ogni gara richiede infatti la configurazione di decine di macchine virtuali, la realizzazione di un'architettura di rete articolata con VPN e firewall, oltre alla creazione di un sistema di gioco adeguato.

Il seguente lavoro di tesi si propone come scopo di ovviare alle problematiche suddette, rimuovendo gli ostacoli tecnici che rallentano la diffusione delle gare A/D e introducendo una soluzione per automatizzare la configurazione dell'apparato infrastrutturale.

Il progetto adotta il paradigma Infrastructure as Code (IaC), che consente di definire le risorse di un'intera infrastruttura tramite codice testuale. Questo approccio garantisce, in primis, una significativa riduzione dei tempi di configurazione dei sistemi, con conseguente diminuzione di eventuali errori umani, ma anche una maggiore scalabilità e riproducibilità dell'infrastruttura stessa.

Una peculiarità del progetto, rispetto ad altre soluzioni esistenti, riguarda la distribuzione delle macchine virtuali su più nodi fisici di un cluster. Grazie a questa funzionalità, gli organizzatori possono sfruttare al meglio le risorse computazionali disponibili per gestire competizioni con un maggior numero di partecipanti.

Inoltre, particolare attenzione è stata dedicata alla normalizzazione del traffico di rete e all'implementazione delle corrette policy di sicurezza necessarie a garantire l'equità della competizione.

Il capitolo 2 introduce il contesto delle competizioni CTF, con particolare attenzione alla modalità Attack and Defense. Successivamente si procede con la descrizione dell'iniziativa CyberChallenge.IT è il principale programma di formazione in cybersecurity in Italia. Vengono inoltre illustrate le nozioni tecniche di base relative a DevOps, Infrastructure as Code e Software Defined Networking.

Il capitolo 3 esamina i requisiti identificati nell'analisi dell'infrastruttura di riferimento e introduce gli strumenti di virtualizzazione e di automazione scelti per l'implementazione. Di seguito viene condotta una valutazione comparativa delle diverse soluzioni esistenti, evidenziando i vantaggi e le limitazioni di ciascuna.

Il capitolo 4 descrive nel dettaglio l'implementazione del codice, illustrando il processo di provisioning e di configurazione delle macchine virtuali, nonché la topologia della rete di gioco e le misure di sicurezza adottate.

Il capitolo 5 presenta i risultati ottenuti durante la fase di test dell'infrastruttura e analizza l'impatto delle varie tecniche di normalizzazione del traffico sulle prestazioni della rete.

Infine, il capitolo 6 riassume gli obiettivi principali del lavoro svolto e propone possibili sviluppi futuri per migliorare ulteriormente l'infrastruttura.

Capitolo 2

Scenari applicativi e stato dell'arte

Il secondo capitolo fornisce il quadro teorico necessario alla comprensione del progetto di tesi. Nella prima sezione 2.1 si analizza il contesto delle competizioni *CTF Attack and Defense* descrivendone le modalità di svolgimento e le dinamiche di gioco. La sezione 2.2 analizza i dati relativi alla diffusione delle competizioni CTF a livello mondiale, evidenziando la disparità tra le gare *Jeopardy* e quelle *Attack and Defense*. Infine, nelle ultime due sezioni 2.3 e 2.4, vengono introdotti i concetti tecnologici su cui si basa il presente lavoro di tesi.

2.1 Competizioni Capture The Flag

Le *Capture The Flag*, note come *CTF* [5], sono competizioni informatiche in cui i concorrenti mettono alla prova le proprie abilità di cybersecurity risolvendo delle sfide proposte dagli organizzatori. Queste competizioni rappresentano uno strumento efficace per la formazione in sicurezza informatica, poiché offrono un ambiente sicuro in cui i partecipanti possono acquisire competenze pratiche e teoriche senza rischi reali.

Le sfide, chiamate *challenge*, sono solitamente classificate in base ai quattro pilastri della sicurezza informatica: Web, Software, Crittografia e Reti. L'obiettivo dei partecipanti è identificare le vulnerabilità presenti nelle *challenge* e trovare un modo per sfruttarle. Come ricompensa, al giocatore viene fornita una *flag* (bandiera), una stringa di testo con un determinato pattern che attesta la risoluzione della sfida. Il nome *Capture The Flag*, infatti, deriva proprio dall'omonimo gioco "rubabandiera" in cui due squadre si sfidano per catturare la bandiera e portarla nella propria base.

L'utilizzo delle CTF in cybersecurity si rivela una scelta particolarmente efficace, poiché sfrutta la *gamification* del processo di apprendimento. La *gamification* [6] [7] consistenell'applicazionee di meccaniche e dinamiche di gioco a contesti non ludici, al fine di rendere più stimolante l'acquisizione di competenze tecniche. Un esempio significativo dell'efficacia di questo approccio proviene da uno studio della Vienna University of Technology, in cui si dimostra che l'approccio competitivo (gamification) non solo è molto apprezzato dagli studenti, ma aumenta anche l'interesse e la motivazione nello studio della cybersecurity [8].

Le competizioni variano per dimensione e prestigio, spaziando da eventi locali organizzati presso ule niversità fino a gare internazionali di grande rilievo ,con premi in denaro anche considerevoli. Tra le modalità più diffuse di *CTF* spiccano le *Jeopardy* e le *Attack and Defense*. Nelle *Jeopardy*, indipendentemente dalla partecipazione singola o in team, ogni "giocatore" trova le flag individualmente e le inoltra al portale di gara. Il punteggio assegnato è direttamente proporzionale al livello di difficoltà della sfida risolta. Nelle *Attack and Defense*, invece, i giocatori organizzati in squadre sono coinvolti simultaneamente in attività di attacco verso gli altri team e di difesa dei propri servizi vulnerabili. A ogni team viene fornita una macchina vulnerabile, comunemente chiamata vulnbox, identica a quelle delle altre squadre. Ogni squadra è responsabile della gestione della propria vulnbox, in cui si eseguono un certo numero di servizi che simulano vulnerabilità informatiche del mondo reale. L'obiettivo di ciascun team è identificare le vulnerabilità, analizzarle,

creare exploit per attaccare gli avversari e, al contempo, difendere i propri servizi implementando contromisure efficaci.

2.1.1 Attack and Defense

Le competizioni *CTF Attack and Defense*, abbreviate in *A/D*, si distinguono per la loro natura dinamica e interattiva che richiede un lavoro di squadra costante tra tutti i componenti del team. A differenza delle *Jeopardy*, dove i partecipanti gareggiano in modo indipendente, le *A/D* richiedono una suddivisione strategica dei ruoli dei giocatori all'interno della squadra: alcuni si occupano degli attacchi, altri della difesa, mentre altri ancora analizzano il traffico di rete e gestiscono l'esecuzione degli exploit.

All'inizio della competizione, ogni team riceve l'accesso remoto alla propria vulnbox che è identica per tutte le squadre partecipanti. Ciascuna vulnbox contiene un numero prestabilito di servizi vulnerabili. I servizi sono server che simulano applicazioni reali e permettono ai clienti di interagire per eseguire determinate operazioni. Ad esempio: un servizio potrebbe essere un sito web che permette la prenotazione di voli aerei. I clienti possono connettersi al servizio tramite un browser web per cercare e prenotare voli. Ogni servizio è costituito da diversi componenti software e contiene al suo interno una o più vulnerabilità intenzionali che costituiscono i vettori di attacco. Tra le vulnerabilità più diffuse figurano SQL Injection, Buffer Overflow e Cross-Site Scripting (XSS), come descritto nel report OWASP delle *Top 10* vulnerabilità più critiche [9]. Sfruttando queste falle, gli attaccanti mirano a esfiltrare dati sensibili custoditi dai servizi, tra cui le flag, che rappresentano l'obiettivo primario dei giocatori.

Ogni competizione A/D ha una durata prestabilita, che può variare da poche ore a più giorni, ed è suddivisa in round (chiamati tick), intervalli temporali che costituiscono l'unità di tempo base della gara. La durata di un tick è tipicamente compresa tra 60 secondi e 5 minuti. Durante ogni round, il sistema di gioco esegue autonomamente una serie di operazioni in ordine casuale:

- Verifica dello stato di funzionamento dei servizi di ogni team
- Inserimento di una nuova flag in ogni servizio di ogni team
- Tentativo di recupero delle flag inserite nelle vulnbox durante i round precedenti
- Calcolo dei punteggi e aggiornamento della classifica

Le flag nelle A/D sono soggette ad un meccanismo di rotazione continua. Ogni flag ha un periodo di validità limitato (tipicamente tra 5 e 10 round), trascorso il quale scade e viene rifiutata dal portale di gioco. Per essere riconoscibili, le flag seguono un formato specifico, definito tramite un'espressione regolare nota a tutti i team.

Il sistema di gioco, chiamato gameserver, si occupa di aggiungere ad ogni round una nuova flag per ciascun servizio e di verificare la reperibilità delle flag inserite nei round precedenti. Un altro compito fondamentale del gameserver è l'aggiornamento del *Service Level Agreement* (SLA) per ciascun team. Lo SLA è un indicatore che misura la percentuale di disponibilità (uptime) e di corretto funzionamento di un servizio nel corso della competizione, rappresentando un fattore determinante per il punteggio finale. Ad ogni tick, il gameserver interagisce con i servizi di ciascun team tramite una serie di controlli automatizzati (checker routine) che simulano il comportamento di un utente legittimo. Il meccanismo di SLA impedisce ai team di adottare strategie difensive semplicistiche come lo spegnimento completo dei servizi o il blocco indiscriminato delle connessioni in ingresso. Al contrario, incentiva i partecipanti a sviluppare e implementare patch mirate in grado di risolvere le vulnerabilità senza compromettere la funzionalità nominale del servizio.

2.2 Motivazione del progetto

L'interesse per le competizioni Capture The Flag, come strumento di formazione nel campo della cybersecurity, ha registrato una crescita significativa negli ultimi anni. Questa tendenza positiva si riflette anche nel panorama nazionale. Il principale programma di formazioni in cybersecurity in Italia, CyberChallenge.IT [10], ha seguito un andamento di espansione progressiva, coinvolgendo ogni anno un numero sempre maggiore di sedi universitarie e studenti iscritti [11].

Analizzando i dati disponibili su CTFtime [12], il principale portale di riferimento per le CTF pubbliche a livello mondiale, emerge un trend di crescita costante negli ultimi 10 ann,i sia nel numero di competizioni organizzate annualmente siainlquelloo di partecipanti attivi.

Tuttavia, analizzando attentamente i grafici 2.1 e 2.2 realizzati tramite le API di CTFtime, si nota che, nonostante la crescita complessiva delle competizioni CTF, le gare in modalità Attack and Defense rappresentano una percentuale estremamente bassa rispetto a quelle in modalità Jeopardy.

L'organizzazione di una competizione Attack and Defense richiede infatti risorse e competenze tecniche significativamente superiori. È necessario progettare un'infrastruttura di rete complessa che garantisca l'isolamento tra i team, gestire la configurazione di decine di macchine virtuali e sviluppare un sistema di gioco *(gameserver)* in grado di gestire in tempo reale la distribuzione delle flag, la verifica della metrica SLA e il calcolo dei punteggi.

Questa complessità organizzativa limita fortemente la possibilità di organizzare sessioni di allenamento per i partecipanti. La mia esperienza personale come concorrente dell'ultima edizione di CyberChallenge.IT 2025 ha evidenziato questa criticità. Durante la preparazione per la finale nazionale di Torino, sono state riscontrate notevoli difficoltà nel testare gli strumenti e le strategie di gara, a causa delle limitate opportunità di esercitazione in ambienti che riproducessero fedelmente le condizioni della competizione reale.

Il presente lavoro di tesi si pone l'obiettivo di affrontare questa sfida,

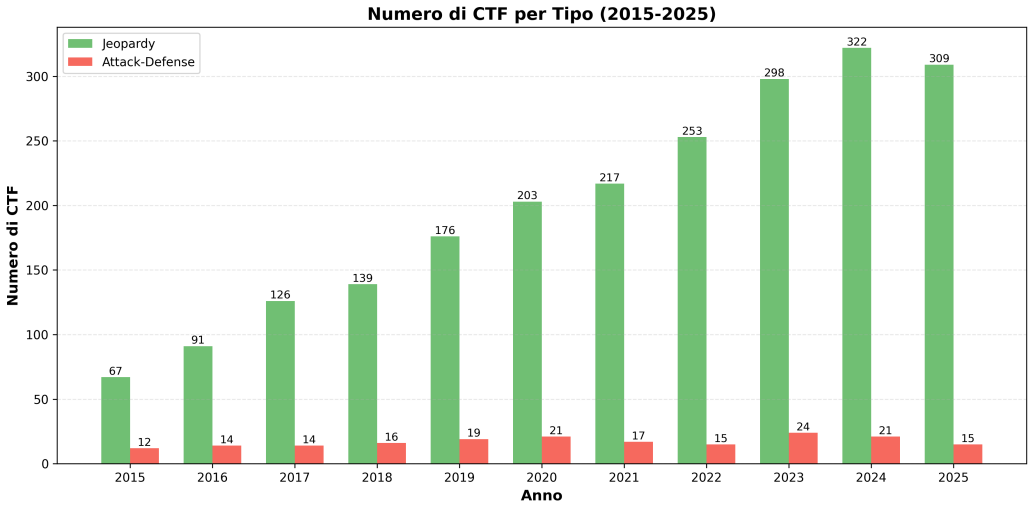

Figura 2.1: Confronto tra il numero di competizioni Jeopardy e Attack and Defense organizzate

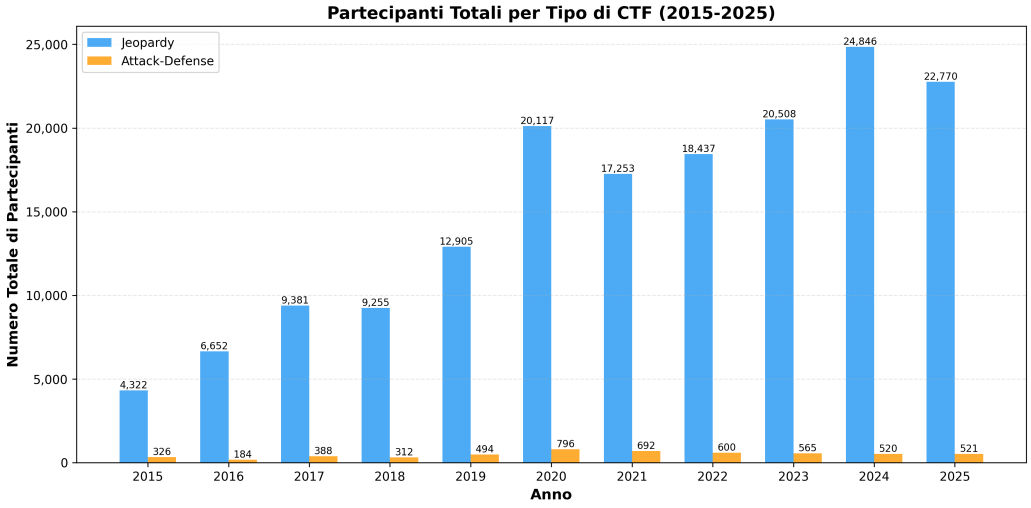

Figura 2.2: Confronto tra il numero di partecipanti a CTF Jeopardy e Attack and Defense

proponendo quindi una soluzione per automatizzare il processo di creazione dell'infrastruttura per competizioni A/D. Lo scopo è progettare e sviluppare un'infrastruttura basata su tecnologie Infrastructure as Code (IaC), in grado di orchestrare automaticamente il *deployment* di tutti i componenti necessari per una gara: la rete di gioco, le vulnbox per ogni team, il gameserver e le configurazioni VPN. Attraverso l'uso di strumenti di automazione come Terraform e Ansible, si intende rendere il processo di configurazione dell'infrastruttura riproducibile, scalabile e accessibile a tutti.

2.3 DevOps e IaC

La crescente complessità nella gestione delle infrastrutture IT ha incentivato l'adozione di nuove metodologie per l'amministrazione di sistemi. Il paradigma DevOps [13] mira a unificare lo sviluppo software e la gestione delle infrastrutture per migliorare la qualità dei prodotti finali.

Uno dei principi fondamentali del DevOps è l'approccio Infrastructure as Code (IaC), che consente di automatizzare la gestione dell'infrastruttura di sistemi tramite codice testuale. L'Infrastructure as Code permette di definire la configurazione di un'infrastruttura, compresi server, reti, macchine virtuali, utilizzando file di configurazione in formato testuale, semplificando notevolmente il lavoro dell'amministratore di sistemi. In particolare, il sistemista descrive lo stato desiderato dell'infrastruttura in un file, mentre gli strumenti IaC si occupano di applicare le modifiche necessarie per raggiungere tale stato. I vantaggi dell'IaC sono molteplici:

- **Riproducibilità:** lo stesso codice genera sempre la stessa infrastruttura indipendentemente dall'ambiente in cui viene eseguito;

- **Scalabilità:** un cambiamento dei requisiti dell'infrastruttura richiede solamente la modifica di alcune variabili nel file di configurazione;

- **Controllo di versione:** il codice dell'infrastruttura può essere gestito con sistemi di controllo di versione per tracciare lo storico delle modifiche e ripristinare versioni precedenti in caso di problemi;

- **Riduzione degli errori umani:** l'automazione riduce al minimo gli interventi manuali, con conseguente diminuzione del rischio di errori umani.

Nel panorama degli strumenti IaC, Terraform e Ansible rappresentano due tecnologie complementari ampiamente utilizzate per la gestione delle infrastrutture.

2.4 Software Defined Networking

Il paradigma del Software Defined Networking introduce un'architettura di rete moderna che separa il piano di controllo (control plane) dal piano di inoltro (data plane). Nelle reti tradizionali, ogni apparato di rete integra sia la logica di controllo sia la funzionalità di inoltro dei pacchetti. Nelle SDN il piano di controllo è centralizzato in un componente software, definito controller, che gestisce l'intera infrastruttura di rete. I dispositivi di rete, come switch e router, si limitano a inoltrare i pacchetti in base alle regole definite dal controller. In un contesto di datacenter, le tecnologie SDN consentono di definire reti virtuali (overlay) per creare segmenti di rete isolati, anche tra macchine virtuali distribuite su nodi fisici diversi. La rete overlay è una rete virtuale che si appoggia a una rete fisica esistente (underlay).

2.4.1 VXLAN

Uno dei protocolli più utilizzati per la creazione di reti overlay è il VXLAN (Virtual Extensible LAN), descritto nel RFC 7348 [14], che consente di estendere le capacità delle tradizionali VLAN per segmentare la rete. VXLAN incapsula i frame Ethernet di livello 2 all'interno di pacchetti UDP,

creando tunnel che operano sulla rete underlay di livello 3 esistente. In pratica, un *frame* Ethernet generato da una macchina virtuale viene incapsulato in un pacchetto UDP e inviato attraverso la rete fisica a un'altra macchina virtuale in esecuzione su un altro host fisico. L'host di destinazione riceve il pacchetto UDP, lo decapsula e consegna il frame Ethernet originale alla macchina virtuale (VM) destinataria. Le VM possono quindi comunicare tra loro come se fossero collegate a un semplice switch L2, indipendentemente dalla loro posizione fisica.

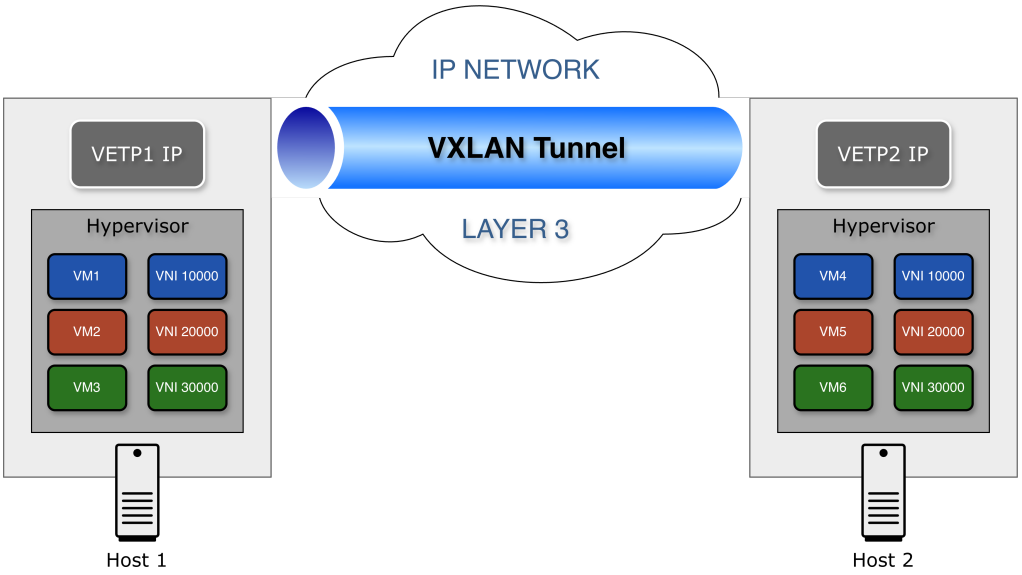

Figura 2.3: Diagramma VXLAN tra due VTEP Hypervisor

I dispositivi che implementano VXLAN, chiamati VXLAN Tunnel End-Points (VTEPs), si occupano dell'incapsulamento e decapsulamento dei pacchetti VXLAN. Ciascun host VTEP deve avere un indirizzo IP che viene utilizzato come indirizzo sorgente dei pacchetti VXLAN. Ogni rete VXLAN è identificata da un VXLAN Network Identifier (VNI) che consente alle macchine virtuali appartenenti allo stesso VNI di comunicare tra loro a livello 2, anche se distribuite su nodi fisici diversi. Il VNI è un identificativo a 24 bit

che consente di creare fino a 16 milioni di reti logiche isolate, superando il limite di 4096 VLAN tradizionali.

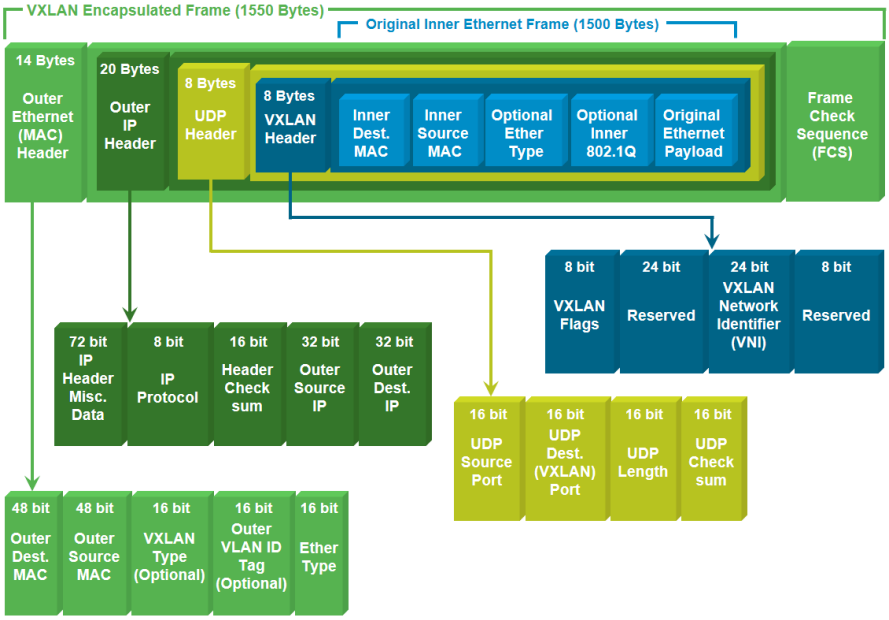

Figura 2.4: Struttura pacchetto VXLAN [1]

Il pacchetto VXLAN utilizza un incapsulamento MAC-in-UDP, in cui il frame Ethernet originale viene inserito all'interno di un pacchetto UDP. La struttura del pacchetto VXLAN, illustrata in figura 2.4, è composta da:

- **Outer Header Ethernet:** L'indirizzo MAC sorgente è quello dell'host VTEP sorgente, mentre l'indirizzo MAC di destinazione è quello del router next-hop lungo il percorso verso il VTEP di destinazione.

- **Outer Header IP:** Contiene gli indirizzi IP di origine e di destinazione degli host VTEP.

- **Outer Header UDP:** La porta di destinazione di default è 4789, mentre la porta di origine è solitamente calcolata dal VTEP tramite un hash dei campi del frame originale.

- **Header VXLAN:** Contiene il VNI, l'identificativo a 24 bit della rete virtuale.
- **Original Inner Frame:** Contiene il frame Ethernet originale completo di header e payload.

Come si può notare dalla figura 2.4, l'incapsulamento VXLAN aggiunge un overhead di 50 byte (54 se viene usato un tag VLAN nella rete underlay) che deve essere gestito correttamente. Le reti Ethernet standard, infatti, hanno un MTU (Maximum Transmission Unit) di 1500 byte, quindi se una VM invia un frame con 1500 byte di dati, il pacchetto incapsulato diventerà di circa 1554 byte. VXLAN non implementa alcun meccanismo nativo di frammentazione, quindi i pacchetti vengono frammentati a livello IP o scartati dai dispositivi di rete. Per gestire questo overhead, è necessario aumentare l'MTU su tutti gli apparati (switch, router) della rete underlay che trasporta il traffico VXLAN. Questo richiede il controllo completo della configurazione della rete fisica tra i nodi VTEP. Alternativamente, è possibile abbassare l'MTU della rete overlay, quindi le interfacce di rete delle VM devono essere configurate con MTU a 1446/1450 byte. Questa soluzione comporta una leggera riduzione delle prestazioni perché i frame presentano un rapporto payload/header meno efficiente, richiedendo l'invio di un numero maggiore di pacchetti per trasferire la stessa quantità di dati.

2.4.2 Flood and Learn

Consideriamo il caso in cui i VTEP siano hypervisor che ospitano macchine virtuali. Quando un VTEP deve inviare un pacchetto VXLAN verso una macchina virtuale remota, deve conoscere l'indirizzo IP del VTEP di destinazione a cui quella VM è connessa. Ogni VTEP mantiene una tabella di mappatura tra gli indirizzi MAC delle VM e gli indirizzi IP dei VTEP corrispondenti. Uno dei meccanismi più semplici per popolare questa tabella è il *flood and learn*. Questo approccio si basa sull'invio di traffico multicast nella rete underlay, così da consentire ai VTEP di apprendere dinamicamente

la posizione degli endpoint. Ad ogni VNI è associato un indirizzo IP multicast specifico. I VTEP che partecipano a quel VNI si iscrivono al gruppo multicast corrispondente per ricevere i pacchetti destinati a quel VNI.

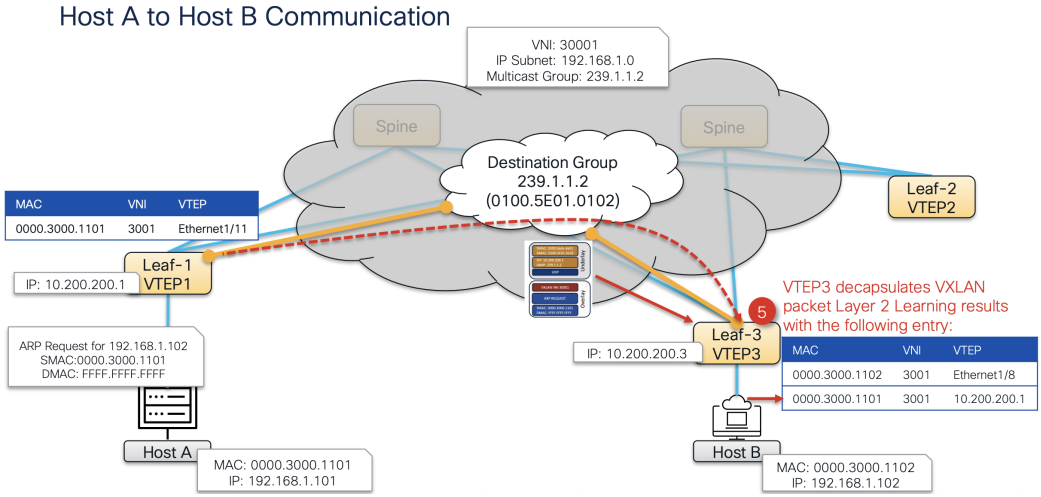

Figura 2.5: Flood and Learn in VXLAN [2]

Quando un VTEP deve inviare un pacchetto a una VM il cui indirizzo MAC non è presente nella tabella di mapping, invia il pacchetto incapsulato al gruppo multicast associato al VNI. Tutti i VTEP che partecipano a quel VNI ricevono il pacchetto multicast, lo decapsulano e memorizzano l'associazione tra l'indirizzo MAC della VM sorgente e l'indirizzo IP del VTEP sorgente 2.5. Il VTEP che possiede la VM di destinazione la consegna alla VM, che risponde normalmente. Il pacchetto di ritorno consente al VTEP sorgente di imparare l'associazione tra l'indirizzo MAC della VM e l'indirizzo IP del VTEP. Tutto il traffico successivo tra le due VM può essere inviato direttamente in unicast.

Capitolo 3

Analisi progettuale

Dopo aver delineato nel capitolo precedente le motivazioni alla base del progetto, in questo capitolo si procede all'analisi dettagliata dei requisiti necessari alla realizzazione di un ambiente di competizione A/D. Nella prima sezione 3.1 viene descritta l'architettura di riferimento ispirata all'infrastruttura della finale nazionale di CyberChallenge.IT e da essa si estraggono i requisiti funzionali e non funzionali che il sistema deve soddisfare (si veda la sezione 3.2). Infine, nella sezione 3.3, viene presentata una panoramica delle tecnologie chiave utilizzate e viene condotta un'analisi comparativa delle soluzioni esistenti per evidenziare il valore aggiunto del presente lavoro di tesi.

3.1 Architettura del sistema

L'obiettivo primario del progetto è la realizzazione di un'infrastruttura IaC che simuli il più fedelmente possibile l'ambiente utilizzato nella finale nazionale di CyberChallenge.IT. Tuttavia, le informazioni pubbliche relative all'implementazione specifica dell'infrastruttura di CyberChallenge.IT sono estremamente limitate. Pertanto, l'architettura di riferimento descritta in questo capitolo si basa sull'esperienza diretta maturata durante la mia par-

tecipazione e sull'analisi del comportamento della rete di gioco durante la competizione.

Il cuore di una competizione Attack and Defense risiede nella sua architettura di rete, che deve garantire al contempo l'isolamento tra le squadre e la connettività controllata per gli attacchi. L'infrastruttura è orchestrata da un router centrale che funge da gateway per tutte le comunicazioni all'interno della rete di gioco. I principali componenti dell'architettura di rete sono:

- **Sottoreti Vulnbox:** Per ogni squadra partecipante viene creata una sottorete dedicata e isolata che ospita la vulnbox. L'isolamento viene gestito tramite VLAN distinte per ogni team. Questo approccio di segmentazione della rete assicura che le vulnbox di ciascuna squadra non possano comunicare direttamente tra loro in LAN, ma solo tramite il router centrale di gioco. L'indirizzo IP delle vulnbox è del tipo '10.60.X.1/24', dove 'X' rappresenta l'ID numerico del team (da 0 a N). L'indirizzo '10.60.0.1' è assegnato al NOP Team (NOn-Playing team): una vulnbox gestita dagli organizzatori in cui non vengono mai applicate patch ai servizi e le cui flag non assegnano punti.

- **Sottoreti VPN:** L'accesso alla rete di gioco da parte dei partecipanti avviene esclusivamente tramite una connessione VPN WireGuard. Anche in questo caso, ad ogni squadra viene assegnata una sottorete VPN distinta, del tipo '10.81.X.0/24'. Questa segmentazione consente di applicare regole di firewall specifiche e di monitorare con precisione il traffico generato da ciascun team.

- **Sottorete Gameserver:** Rete in cui risiede il sistema di gioco con indirizzo IP '10.10.0.1'. Questo indirizzo deve essere raggiungibile da tutti i giocatori perché il gameserver espone servizi WEB quali la classifica, l'endpoint per la sottomissione delle flag e quello per ottenere i flagids.

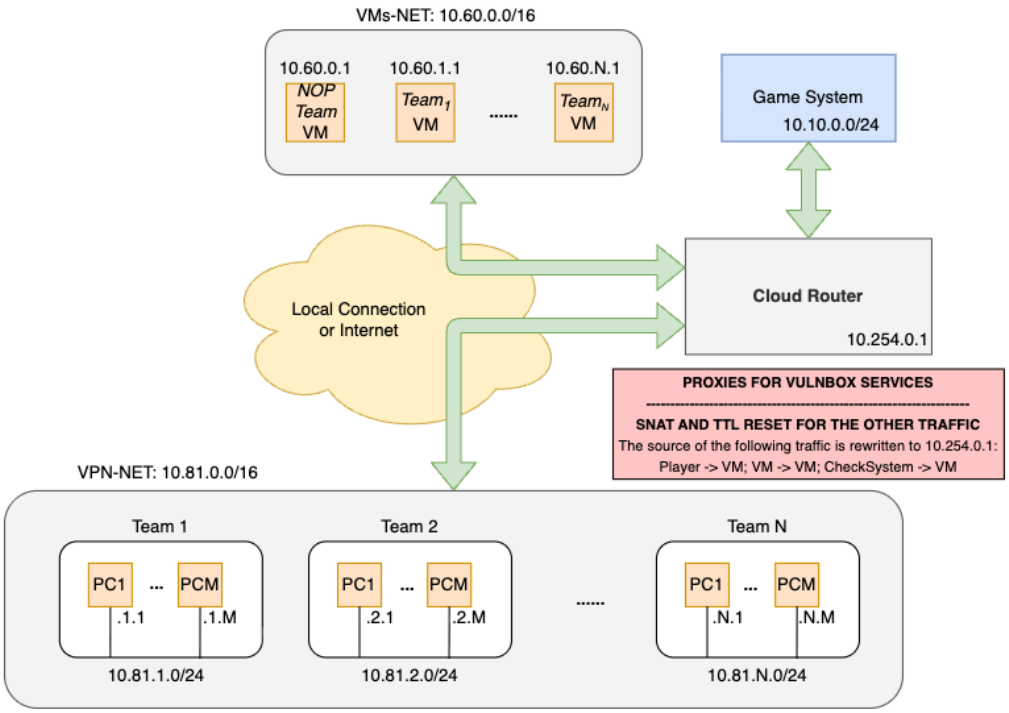

Figura 3.1: Topologia di rete dell'A/D di CyberChallenge.IT [3]

Il flusso di comunicazione è gestito da precise regole di routing e da firewall. Un giocatore del Team 1, connesso tramite VPN a un IP nella rete 10.81.1.0/24, per attaccare la vulnbox del Team 2 (10.60.2.1) dovrà necessariamente attraversare il router centrale. Quest'ultimo si occuperà di instradare correttamente il traffico e di applicare le policy di sicurezza necessarie. Allo stesso modo, il gameserver 10.10.0.1 deve poter raggiungere tutte le vulnbox in tutte le sottoreti 10.60.X.1 per poter eseguire le checker routine e depositare le nuove flag. Infine, l'infrastruttura deve garantire l'anonimizzazione del traffico diretto verso le vulnbox: i pacchetti di rete del checker devono essere indistinguibili da quelli degli exploit dei giocatori.

3.2 Requisiti identificati

Dall'analisi dell'architettura di riferimento emerge una serie di requisiti, sia funzionali sia non funzionali, che il progetto deve soddisfare per offrire un ambiente di gara realistico.

- **Deploy automatico delle VM:** L'intero processo di creazione e configurazione dell'infrastruttura deve essere automatizzato attraverso strumenti IaC. Questo include il provisioning automatico di tutte le macchine virtuali, partendo da un'immagine predefinita (template).

- **Accesso remoto via VPN:** I giocatori devono potersi connettere da remoto alla rete di gioco tramite una connessione VPN. Il sistema deve essere in grado di generare automaticamente le configurazioni VPN per ogni squadra.

- **Anonimizzazione del traffico:** Tutto il traffico di rete diretto alle vulnbox deve essere anonimizzato. I giocatori, catturando il traffico sulla propria VPN, non devono riuscire a distinguere la provenienza delle richieste analizzando gli header dei pacchetti di rete.

- **Scalabilità:** Il progetto deve supportare un numero variabile di team e giocatori, semplicemente modificando le variabili di configurazione, senza richiedere interventi manuali.

- **Riproducibilità:** L'intera infrastruttura deve poter essere creata, configurata e distrutta in modo completamente automatico. Eseguendo lo script di deployment in ambienti diversi, si deve ottenere un sistema identico.

- **Modularità:** L'architettura deve essere modulare: gli organizzatori della competizione CTF devono poter integrare i propri servizi vulnerabili o modificare alcuni componenti senza dover riprogettare l'intero sistema.

- **Distribuzione su più nodi:** Il progetto deve consentire di distribuire le macchine virtuali su più nodi di un cluster Proxmox. Questo consente di superare i limiti di un singolo host fisico e di gestire competizioni con un numero maggiore di partecipanti.

- **Isolamento della rete:** Le vulnbox di ogni team devono operare all'interno di una propria sottorete isolata, in modo da imporre che l'intero traffico passi attraverso il router centrale di gioco.

3.3 Scelta delle tecnologie

La selezione degli strumenti IaC si è orientata a soluzioni prevalentemente open source, ampiamente adottate nella comunità DevOps, per evitare il vendor lock-in e garantire la massima flessibilità.

3.3.1 Proxmox VE

Per la gestione delle macchine virtuali è stato scelto Proxmox Virtual Environment (Proxmox VE) [15], una piattaforma open-source per la virtualizzazione che integra l'hypervisor KVM (Kernel-based Virtual Machine). Innanzitutto, essendo completamente open source e self-hostabile, Proxmox consente di mantenere il controllo totale sull'infrastruttura senza dipendere da servizi cloud esterni. Inoltre, fornisce funzionalità native di clustering che consentono di distribuire le macchine virtuali su più nodi fisici. L'interfaccia web integrata facilita il monitoraggio e la gestione delle VM, mentre l'API REST esposta da Proxmox consente l'integrazione con strumenti di automazione.

3.3.2 Terraform

Per l'orchestrazione e il provisioning dell'infrastruttura è stato scelto Terraform [16], uno strumento IaC sviluppato da HashiCorp che permette di

definire le risorse attraverso file di configurazione scritti in linguaggio dichiarativo HCL. Uno dei principali vantaggi di Terraform è il supporto di centinaia di provider diversi, permettendo di gestire non solo l'infrastruttura Proxmox, ma potenzialmente anche risorse su cloud provider esterni. In questo progetto è stato utilizzato il provider BPG, mantenuto dalla comunità open source, che consente di interfacciarsi con l'API REST di Proxmox.

3.3.3 Ansible

Mentre Terraform si occupa di creare le risorse, Ansible è stato scelto per la configurazione del software sulle macchine virtuali. Ansible [17] è uno strumento open-source che permette di gestire la configurazione di sistemi utilizzando un approccio *agentless*: non richiede l'installazione di software aggiuntivo sulle macchine gestite (managed nodes), ma si connette via SSH per eseguire le operazioni richieste. La configurazione in Ansible viene definita attraverso una lista di task scritti in linguaggio YAML che descrivono lo stato desiderato del sistema.

3.3.4 WireGuard

Per la realizzazione della VPN che consente ai giocatori di accedere da remoto alla rete di gioco è stato scelto WireGuard, un protocollo VPN moderno e prestante, integrato nel Kernel Linux dalla versione 5.6. Nei test di throughput [18], WireGuard ha fornito prestazioni superiori rispetto alle soluzioni VPN tradizionali come OpenVPN e IPSec, pur mantenendo un utilizzo della CPU inferiore. Un ulteriore vantaggio è la sua semplicità: la configurazione, infatti, si basa su semplici file di testo e su coppie di chiavi crittografiche pubbliche/private, che ne facilitano l'automazione tramite Ansible.

3.3.5 Docker

Per il deployment dei servizi vulnerabili (challenge) sulle vulnbox si è scelto di utilizzare la tecnologia di containerizzazione Docker. Anziché installare le dipendenze di ogni servizio direttamente sul sistema operativo della Vulnbox, ogni challenge verrà archiviata come immagine Docker e gestita tramite Docker Compose. Ogni servizio viene eseguito nel proprio container, isolato dagli altri e dal sistema operativo del host.

3.4 Analisi soluzioni esistenti

Prima di procedere allo sviluppo del progetto, è stata condotta un'analisi delle principali soluzioni disponibili per l'organizzazione di competizioni CTF A/D. L'obiettivo è stato identificare i pregi e le limitazioni di ciascun progetto rispetto ai requisiti definiti.

3.4.1 FAUST Gameserver

FAUST CTF Gameserver [19] è un framework per l'organizzazione di competizioni A/D, sviluppato dal team FAUST e utilizzato per ospitare l'omonima competizione. Il progetto si caratterizza per un'architettura modulare in cui i diversi componenti, orchestrati dal controller centrale, comunicano tramite un database PostgreSQL condiviso. Un aspetto positivo di FAUST Gameserver è la qualità della documentazione tecnica, che descrive nel dettaglio l'architettura e il funzionamento del sistema di gioco. Tuttavia, come esplicitamente indicato nella documentazione ufficiale, il framework non include strumenti per la gestione dell'infrastruttura di rete, la configurazione della VPN né il deploy delle VM, ma si concentra esclusivamente sul gameserver.

3.4.2 saarCTF

saarCTF [20] è un'altra soluzione molto conosciuta che fornisce script di setup per tutti i server necessari all'organizzazione di una competizione A/D, ma presenta diverse incompatibilità con i requisiti di questo progetto. In primo luogo, non è presente alcuna integrazione con Proxmox VE: il sistema è progettato per utilizzare immagini in formato OVA tramite Virtualbox. Inoltre, il processo di deployment non è automatizzato e richiede diversi passaggi manuali per la creazione e l'installazione delle immagini delle VM.

3.4.3 CTFBox

CTFBOX [21] è un progetto open-source recente che offre un'interfaccia web e un gameserver con endpoint simili a quelli utilizzati nella competizione A/D di CyberChallenge. La sua architettura di base, però, presenta una limitazione significativa rispetto agli obiettivi di questo progetto: le vulnbox non sono vere macchine virtuali, ma container Docker eseguiti su un singolo host. Questa scelta progettuale, se da un lato semplifica il deployment su un singolo host, dall'altro impedisce la distribuzione del carico su più nodi fisici e limita la scalabilità dell'infrastruttura.

L'analisi delle soluzioni esistenti ha evidenziato che nessuna delle piattaforme esaminate soddisfi completamente i requisiti identificati. Manca infatti un progetto che offra un approccio IaC completo per il deployment automatico di un'infrastruttura basata su macchine virtuali, scalabile e distribuita su più nodi fisici di un cluster Proxmox.

Capitolo 4

Implementazione

Dopo aver definito l'architettura di riferimento e i requisiti del progetto, in questo capitolo viene descritta l'implementazione concreta dell'infrastruttura IaC. In particolare, vengono analizzate nel dettaglio le scelte implementative adottate per ciascun componente del sistema, partendo dalla configurazione dell'ambiente di virtualizzazione Proxmox (sez 4.1), continuando con la creazione delle macchne virtuali e della rete di gioco (sez 4.2), la configurazione del software all'interno delle VM (sez 4.3), fino ad arrivare alle tecniche di anonimizzazione del traffico di rete (sez 4.4). Infine, la sezione 4.5 presenta un'analisi della tecnica di fingerprinting del mittente delle connessioni TCP basata sui timestamp e le contromisure adottate per mitigare questa vulnerabilità.

4.1 Setup dell'ambiente Proxmox

Prima di procedere con il deployment automatizzato dell'infrastruttura, è necessario predisporre l'ambiente di virtualizzazione basato su Proxmox VE. Proxmox VE può essere installato su hardware fisico oppure eseguito come macchina virtuale all'interno di un hypervisor esistente. Per l'installazione, è possibile scaricare l'immagine ISO dal sito di Proxmox VE [15] e seguire la documentazione ufficiale per l'installazione. Una volta completata l'in-

stallazione, è possibile accedere all'interfaccia web di Proxmox tramite un browser, utilizzando l'indirizzo IP assegnato al server. È possibile installare Proxmox VE su più nodi fisici e unirli in un unico cluster per distribuire le macchine virtuali su più host fisici. All'interno del progetto viene fornito un file README con le istruzioni per la creazione di un *API Key* con i privilegi minimi necessari per il provisioning delle risorse con Terraform.

4.1.1 Template delle macchine virtuali

Per consentire a Terraform di creare rapidamente le macchine virtuali necessarie per la competizione, è stato realizzato uno script bash che automatizza la creazione di una VM template utilizzata come base per tutte le VM del progetto. Lo script si occupa di scaricare l'immagine cloud di Debian 13 in formato QCOW2, installarci il pacchetto qemu-guest-agent per permettere a Proxmox di avere informazioni sulle interfacce di rete, e infine creare una nuova VM template utilizzando l'immagine scaricata come disco di avvio.

4.2 Creazione delle macchine virtuali con Terraform

Una volta preparato l'ambiente Proxmox, l'intero processo di creazione delle macchine virtuali è orchestrato da Terraform. Terraform non supporta nativamente Proxmox, quindi è stato deciso di utilizzare un provider di terze parti, BPG [22], che permette di interfacciarsi con l'API REST di Proxmox. L'utilizzatore deve semplicemente modificare il file `variables.tf` per definire il numero di team partecipanti, il numero di giocatori per team, l'orario di inizio e fine della competizione e altri parametri di configurazione.

Tutte le VM sono clonate in modalità linked clone a partire dal template creato in precedenza, riducendo significativamente il tempo di provisioning e lo spazio richiesto su disco. La linked clone crea una nuova VM che condivide

i dischi virtuali con il template di origine, memorizzando solo le differenze rispetto al template. Le interfacce di rete delle VM sono collegate alla rete SDN VXLAN 4.2.1, con VLAN ID univoci per ogni team per garantire l'isolamento del traffico. L'MTU è impostato a 1 che indica di utilizzare l'MTU della rete sottostante (1446 byte come descritto in 2.4.1). Ogni VM viene configurata con Cloud-Init per automatizzare la creazione dell'utente, l'installazione delle chiavi SSH e la configurazione dei parametri di rete per l'interfaccia di Management.

Lo snippet di codice riporta un esempio del file di configurazione Terraform che definisce la risorsa per la creazione delle vulnbox per ogni team partecipante. La variabile `proxmox_node_names` contiene la lista dei nomi dei nodi del cluster Proxmox. Le vulnbox vengono automaticamente distribuite su più nodi utilizzando l'operatore modulo per calcolare l'indice del nodo in base all'ID del team.

```
resource "proxmox_virtual_environment_vm" "vulnboxes" {
  for_each    = local.teams
  name        = "ctf-vulnbox-team${each.value.id}"
  description = "Vulnbox for team ${each.value.id}"
  node_name   = var.proxmox_node_names[each.value.id %
  ↪  length(var.proxmox_node_names)]
  vm_id       = each.value.vm_id
  started       = true

  clone {
    vm_id     = var.template_id
    node_name = var.template_node
    full      = false
  }
  agent {
    enabled = true
  }
  cpu {
    cores = 4
    type  = "host"
  }
  memory {
    dedicated = var.vulnbox_memory
  }
  bios = "ovmf"
  scsi_hardware = "virtio-scsi-pci"
```

```
26
27    # Management interface
28    network_device {
29      bridge  = "adnet"
30      model   = "virtio"
31      vlan_id = each.value.vlan_id
32      mtu     = 1
33    }
34    # Game interface
35    network_device {
36      bridge  = "adnet"
37      model   = "virtio"
38      vlan_id = each.value.vlan_id
39      mtu     = 1
40    }
41
42    disk {
43      datastore_id = var.proxmox_storage_name
44      interface    = "scsi0"
45      size         = 15
46    }
47
48    initialization {
49      datastore_id = var.proxmox_storage_name
50      user_account {
51        username = var.cloud_init_user
52        password = var.cloud_init_password
53        keys     = var.ssh_keys
54      }
55      ip_config {
56        ipv4 {
57          address = "172.16.${each.value.id}.1/24"
58          gateway = "172.16.${each.value.id}.254"
59        }
60      }
61    }
62  }
```

Codice 4.1: Definizione delle Vulnbox in Terraform

4.2.1 Configurazione della Rete e SDN

Per soddisfare il requisito di distribuzione delle VM su più nodi fisici, è stato necessario superare la limitazione dei tradizionali Linux Bridge che operano a livello di singolo host. La soluzione adottata è stata l'utilizzo del-

la funzionalità di Software Defined Networking (SDN) integrata in Proxmox VE. Grazie all'utilizzo del provider terraform BPG è stato possibile configurare una rete VXLAN (Virtual Extensible LAN), che permette di creare una rete virtuale di livello 2 (overlay) sopra la rete fisica di livello 3 esistente tra i nodi del cluster. In questo modo, le macchine virtuali possono comunicare tra loro come se si trovassero sullo stesso segmento di rete, indipendentemente dal nodo fisico del cluster su cui sono in esecuzione. La configurazione di VXLAN in Proxmox è stata effettuata attraverso l'interfaccia SDN, creando una zona di tipo VXLAN e specificando gli indirizzi IP dei nodi (peers) che partecipano al tunnel. L'MTU della zona è stata impostata a 1446 byte per gestire correttamente l'overhead di incapsulamento VXLAN ed evitare problemi di frammentazione dei pacchetti senza dover modificare l'MTU di tutti gli apparati fisici della rete underlay (come descritto nella sezione 2.4.1). Inoltre è stata creata una rete virtuale (VNet) associata alla zona VXLAN con VNI 20000, che funge da bridge virtuale per le interfacce di rete delle VM. La VNet è configurata in modalità VLAN-aware per permettere l'isolamento del traffico tra le diverse sottoreti delle vulnbox usando le VLAN tradizionali. Questa segmentazione garantisce che tutte le comunicazioni della gara passino attraverso il router centrale dove vengono applicate le regole di firewall.

```
resource "proxmox_virtual_environment_sdn_zone_vxlan" "adzone" {
  id    = "adzone"
  peers = var.proxmox_node_ips
  mtu   = 1446
}

resource "proxmox_virtual_environment_sdn_vnet" "adnet" {
  id         = "adnet"
  zone       = proxmox_virtual_environment_sdn_zone_vxlan.adzone.id
  alias      = "CTF AD VNet"
  tag        = 20000
  vlan_aware = true
}
```

Codice 4.2: Configurazione della rete SDN VXLAN in Terraform

È importante notare che se il firewall integrato in Proxmox è abilitato, bisogna assicurarsi che la porta UDP 4789 sia aperta per permettere la comunicazione VXLAN tra i nodi.

4.3 Configurazione software con Ansible

L'intera infrastruttura è organizzata in 4 categorie di macchine virtuali, ognuna con un ruolo specifico all'interno della competizione A/D: il router di gioco, il gateway router, il gameserver e le vulnbox. Una volta che le VM sono avviate e raggiungibili in rete, Ansible si occupa di configurare il software all'interno di ciascuna VM, installando i pacchetti necessari, configurando i servizi di rete e applicando le policy di sicurezza richieste. Terraform, una volta terminato il provisioning delle risorse, si occupa di generare dinamicamente il file di inventario di Ansible, elencando tutte le VM create con i relativi indirizzi IP di management ed eventuali jump host ssh necessari per raggiungerle. L'architettura implementata prevede una separazione netta tra la rete di gioco e la rete di management. Ogni macchina virtuale dell'infrastruttura dispone di due interfacce di rete: l'interfaccia di gioco con indirizzo nella sottorete 10.60.X.0/24, attraverso cui passa tutto il traffico della competizione gestito dal router centrale e l'interfaccia di management con indirizzo nella rete 172.16.0.0/16, utilizzata dagli organizzatori per la gestione del sistema. Questa doppia interfaccia garantisce che eventuali malfunzionamenti sulla rete di gioco non impediscano agli amministratori di accedere alle VM per interventi di manutenzione. Il playbook Ansible è organizzato in ruoli: ogni categoria di VM (router, vulnbox, gameserver) ha il proprio set di task che descrivono la configurazione desiderata.

4.3.1 Gateway Router

Il gateway router è la prima macchina virtuale ad essere configurata da Ansible, perché svolge due funzioni necessarie per il deployment del resto dell'infrastruttura. In primo luogo, fornisce accesso a Internet a tutte le

VM attraverso SNAT masquerading implementato con nftables. Il playbook Ansible abilita l'IP forwarding sul gateway router e crea una regola di postrouting che modifica l'indirizzo IP sorgente dei pacchetti provenienti dalle VM sostituendolo con l'IP dell'interfaccia esterna del gateway router. In secondo luogo, il gateway router funge da jump host SSH per permettere ad Ansible di connettersi alle altre VM dell'infrastruttura che non dispongono di un indirizzo IP pubblico. Ansible infatti, necessita di collegarsi ad ogni VM per eseguire i task di configurazione, ma le VM sono raggiungibili solo attraverso la rete di management privata che non è accessibile dalla macchina su cui viene eseguito Ansible.

```
table ip filter {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow established and related connections
        ct state { established, related } accept

        # Allow ssh
        tcp dport 22 accept

        # Allow beszel agent (monitoring software) port
        tcp dport 45876 accept

        # Allow traffic to internet
        oifname "{{ outbound_interface }}" accept
    }
}

table ip nat {
    chain postrouting {
        type nat hook postrouting priority 100;

        # Masquerade traffic for the external interface
        oifname "{{ outbound_interface }}" masquerade
    }
}
```

Codice 4.3: Regole nftables per il gateway router

4.3.2 Gameserver

Il gameserver è il cuore di una competizione CTF A/D e ospita i componenti del sistema di gioco: il database per lo stato della partita, il checker che verifica la disponibilità dei servizi e deposita le flag, l'interfaccia web per la visualizzazione della classifica e l'endpoint per la sottomissione delle flag. Dopo aver analizzato le soluzioni esistenti, è stato scelto di utilizzare CTFBox come base per il sistema di gioco, apportando alcune modifiche per adattarlo alle esigenze specifiche del progetto. In particolare, sono stati rimossi tutti i componenti relativi all'infrastruttura e sono stati mantenuti solo i servizi relativi al gameserver necessari per la gestione della competizione:

- **Database PostgreSQL:** Utilizzato per memorizzare lo stato della partita, le informazioni sui team, le flag e i punteggi.

- **Checker:** Un programma che si occupa di eseguire ad ogni round gli script Python per la routine di controllo dei servizi vulnerabili delle vulnbox.

- **Interfaccia Web:** Un'applicazione web che fornisce una dashboard per visualizzare la classifica in tempo reale.

- **Endpoint di sottomissione delle flag:** Un endpoint API sulla porta 8080 che permette ai giocatori di inviare le flag raccolte durante la competizione. Ogni giocatore può sottomettere le flag tramite richieste HTTP PUT inserendo come header il Team Token assegnato.

- **Endpoint di richiesta flagID:** Un endpoint API GET porta 8081 che permette ai giocatori di richiedere la lista delle flagID. Le flagID sono dei suggerimenti che il gameserver fornisce ai giocatori per facilitare la ricerca delle flag all'interno dei servizi vulnerabili.

4.3.3 Vulnbox

Il playbook Ansible si occupa di installare Docker e Docker Compose, copiare le cartelle delle challenge e avviare ogni servizio con lo script di deploy. L'utilizzatore deve copiare all'interno della cartella files del ruolo vulnbox le cartelle delle challenge della competizione. La scelta di utilizzare Docker permette agli organizzatori di distribuire facilmente i propri servizi vulnerabili fornendo semplicemente un file docker-compose.yml e uno script di deploy. Per il test del progetto sono state utilizzate le 4 challenge sviluppate per la finale nazionale di CyberChallenge.IT 2024.

L'interfaccia della rete di gioco viene configurata con systemd-networkd usando un template Ansible Jinja2 che imposta l'indirizzo IP statico 10.60.X.1 in base all'ID del team e assegna una rotta statica per l'indirizzo IP del router centrale con cui vengono instradate tutte le comunicazioni verso le vulnbox. Inoltre, viene installato e configurato il software di monitoraggio Beszel Agent [23], che invia metriche di utilizzo delle risorse del sistema (CPU, memoria, spazio su disco, rete) al server di monitoraggio.

```
[Match]
Name=game

[Network]
Address={{ team_network_prefix }}.{{ team_id }}.1/24

[Route]
Destination=10.254.0.1
Gateway={{ team_network_prefix }}.{{ team_id }}.254
```

Codice 4.4: Configurazione dell'interfaccia di rete della vulnbox

Terminata la configurazione, Ansible, usando le API di Proxmox, crea uno snapshot di tutte le vulnbox della gara. Lo snapshot cattura lo stato attuale della VM, permettendo agli organizzatori di ripristinare rapidamente la vulnbox ad uno stato pulito. I team, infatti, durante la gara possono richiedere agli organizzatori il reset della propria vulnbox in caso di malfunzionamenti durante la competizione.

4.3.4 Router di gioco

Il router di gioco si occupa di instradare tutto il traffico tra le diverse sottoreti delle vulnbox, la rete VPN dei giocatori e il gameserver applicando le policy di firewall necessarie. Il router centrale è l'unica VM con 3 interfacce di rete: una per la rete di management, una per la rete di gioco e una con indirizzo IP pubblico, attraverso cui i giocatori si connettono tramite VPN. L'interfaccia di gioco è collegata alla rete SDN VXLAN in modalità trunk, permettendo al router di gestire il traffico di tutte le sottoreti delle vulnbox attraverso la creazione di subinterfacce VLAN. Il playbook Ansible si occupa di creare le subinterfacce VLAN per ogni team, assegnando a ciascuna di esse un indirizzo IP in formato 10.60.X.254, oltre all'interfaccia principale con indirizzo 10.254.0.1 che funge da gateway predefinito per tutte le comunicazioni nella rete di gioco.

Per la gestione del traffico VPN, viene installato Wireguard e vengono generate le configurazioni per tutti i giocatori, creando una chiave privata e pubblica per ogni utente e assegnando un indirizzo IP nel formato 10.81.X.Y, dove X è il numero del team e Y è l'ID del giocatore. Le configurazioni Wireguard vengono generate con Ansible a partire dal seguente template Jinja2. La PrivateKey è la chiave privata che identifica il giocatore, mentre la PublicKey è la chiave pubblica del router. Ad ogni giocatore viene assegnato un indirizzo IP nel formato 10.81.X.Y, dove X è l'ID del team e Y il numero del giocatore. L'Endpoint specifica l'indirizzo IP pubblico e la porta del server VPN a cui il client deve connettersi. La direttiva AllowedIPs definisce gli indirizzi IP che devono essere instradati attraverso la VPN, ovvero tutte le sottoreti delle vulnbox e l'indirizzo del gameserver. Inoltre, viene installata WGDashboard, un'applicazione web che consente agli organizzatori di visualizzare lo stato delle connessioni VPN in modo semplice e intuitivo.

```
[Interface]
PrivateKey = {{ client_private_key }}
Address = {{ client_ip }}/32

[Peer]
PublicKey = {{ server_public_key }}
Endpoint = {{ VPN_endpoint }}:51820
AllowedIPs = 10.81.0.0/16, 10.60.0.0/16, 10.10.0.1/32
PersistentKeepalive = 25
```

Codice 4.5: Template Wireguard per le configurazioni dei giocatori

Infine, viene installato e configurato nftables per implementare le regole di firewall necessarie a garantire la sicurezza della rete di gioco. In particolare, vengono generati 3 file nftables: uno per ogni fase di gioco (grace period, open network, closed game). Durante il grace period, ogni giocatore può accedere solamente al gameserver e alla propria vulnbox, mentre tutto il resto del traffico viene bloccato. Nella fase di open network, i giocatori possono eseguire gli attacchi verso tutte le vulnbox avversarie. Infine, nella fase di closed game, tutto il traffico viene bloccato eccetto quello verso il gameserver per poter vedere la classifica finale. Vengono anche creati dei timer di sistema (systemd timers) che impostano automaticamente le regole di firewall appropriate in base alla fase di gioco corrente, in modo da non richiedere interventi manuali da parte degli organizzatori durante la competizione.

Lo snippet di codice riporta un esempio delle regole di firewall utilizzate durante la fase di open network. In questa fase, viene permesso tutto il traffico proveniente dal gameserver verso le vulnbox, il traffico proveniente dalla rete VPN dei giocatori verso le vulnbox e verso i servizi del gameserver. Inoltre viene consentito il traffico VPN tra i giocatori della stessa squadra, mentre tutto il resto del traffico viene bloccato (policy default drop).

```
table inet ctf_firewall {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow established and related connections
        ct state { established, related } accept

        # Allow all traffic from the game server to the vulnboxes
        iifname "ens19.1000" ip saddr 10.10.0.1 ip daddr
        ↪  10.60.0.0/16 accept

        # Allow VPN players to communicate with all vulnboxes
        iifname "wg0" ip saddr 10.81.0.0/16 ip daddr 10.60.0.0/16
        ↪  accept

        # Allow VPN players to communicate with the game server
        ↪  services
        iifname "wg0" ip saddr 10.81.0.0/16 ip daddr 10.10.0.1 tcp
        ↪  dport { 80, 8080, 8081 } accept

        # Allow player VPN communication within the same team
        {% for team_id in range(1, teams|length + 1) %}
        iifname "wg0" oifname "wg0" ip saddr 10.81.{{ team_id
        ↪  }}.0/24 ip daddr 10.81.{{ team_id }}.0/24 accept
        {% endfor %}

        # Block all other communication between team VPN networks
        iifname "wg0" oifname "wg0" drop
    }
}

```

Codice 4.6: Regole di firewall rete di gioco - fase Open Network

4.4 Tecniche di anonimizzazione del traffico di rete

Uno dei requisiti fondamentali dell'infrastruttura è che i giocatori non devono poter distinguere il traffico proveniente dal checker da quello degli exploit degli altri team analizzando i pacchetti di rete catturati sulla propria

vulnbox. Le prime tecniche di anonimizzazione adottate sono state l'utilizzo del SNAT (Source Network Address Translation) e la normalizzazione del TTL (Time To Live) per tutti i pacchetti in uscita dal router centrale. Il SNAT viene implementato con una regola di postrouting in nftables che modifica l'indirizzo IP sorgente di tutti i pacchetti destinati alle vulnbox, sostituendolo con l'indirizzo IP dell'interfaccia di gioco del router centrale 10.254.0.1. Il TTL è un campo dell'header IP che viene decrementato ad ogni hop e potrebbe essere utilizzato per inferire informazioni sulla distanza tra il mittente e il destinatario del pacchetto. In questo modo, tutti i pacchetti diretti alle vulnbox appaiono come provenire dallo stesso indirizzo IP (quello del router) e con lo stesso valore di TTL, rendendo difficile ai giocatori distinguere tra traffico legittimo del checker e traffico malevolo degli exploit.

```
table inet ctf_nat {
    chain snat_to_vulnboxes {
        type nat hook postrouting priority srcnat;

        # SNAT traffic to vulnboxes
        ip daddr 10.60.0.0/16 snat 10.254.0.1
    }
}
table inet ctf_mangle {
    chain postrouting {
        type filter hook postrouting priority mangle;

        # TTL normalization for traffic to vulnboxes
        ip daddr 10.60.0.0/16 ip ttl set 64;
    }
}
```

Codice 4.7: Regole di NAT e normalizzazione TTL sul game router

4.5 Analisi e mitigazione del TCP Timestamp Fingerprinting

Nel contesto delle competizioni CTF Attack/Defense, l'identificazione del mittente di un pacchetto di rete costituisce una vulnerabilità critica che può compromettere l'equità della competizione. In particolare, la possibilità di distinguere i pacchetti provenienti dal gameserver (checker) da quelli inviati dai team avversari rappresenta un vantaggio significativo. L'obiettivo della normalizzazione del traffico, o "scrubbing", è modificare i pacchetti in entrata per conformarli a un'unica "impronta". Questo processo rimuove le caratteristiche uniche dello stack TCP/IP del mittente originale, facendo apparire tutti i pacchetti come se provenissero dalla stessa fonte. Durante le prime prove sull'infrastruttura, è emerso che le tecniche di anonimizzazione di base non fossero sufficienti a prevenire un'analisi più sofisticata. Nonostante tutti i pacchetti catturati sulle vulnbox abbiano lo stesso indirizzo IP di origine, è stato identificato un possibile vettore di fingerprinting passivo basato sul campo TCP Timestamp dell'header TCP.

Il Transmission Control Protocol (TCP) è un protocollo di livello trasportato che garantisce l'affidabilità del trasferimento dati mediante la ritrasmissione dei pacchetti persi, l'ordinamento dei segmenti e il controllo del flusso. Quasi tutte le challenge delle competizioni CTF A/D utilizzano servizi basati su TCP o su protocolli applicativi che si appoggiano a TCP, come HTTP. Per comprendere il meccanismo di fingerprinting, ovvero di identificazione di un'impronta basata sui timestamp, è necessario analizzare la struttura dell'header TCP. L'intestazione (header) di un segmento TCP ha una dimensione minima di 20 byte, ma può estendersi fino a 60 byte grazie all'uso del campo Options.

Il campo Options ha una lunghezza variabile fino a 40 byte e serve a permettere l'aggiunta di estensioni future al protocollo TCP. A ogni opzione è associato un identificatore univoco (Kind) e una lunghezza (Length) che specifica la dimensione dell'opzione in byte. Il formato standard dell'header

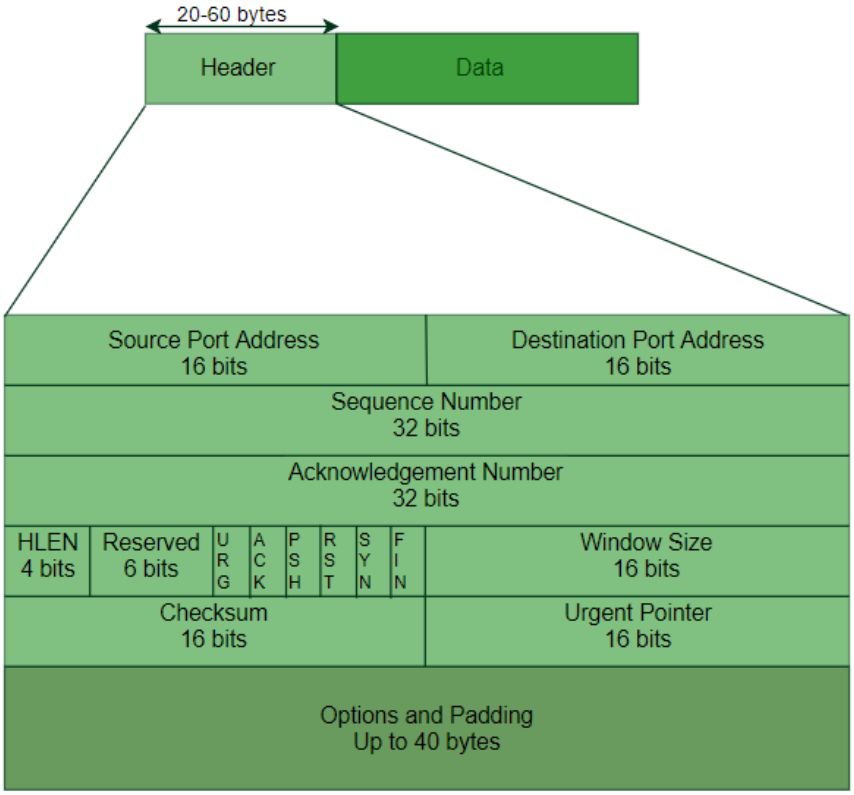

Figura 4.1: Struttura di un segmento TCP [4]

include campi essenziali come Porta Sorgente, Porta Destinazione, Numero di Sequenza, Numero di Acknowledgment, Flag di Controllo e Checksum.

Per adattare il protocollo TCP all'evoluzione delle reti ad alta velocità, con l'RFC 1323 e successivamente con il 7323 (TCP Extensions for High Performance [24]) sono state introdotte diverse estensioni, tra cui l'opzione TCP Timestamp. Questa opzione è identificata dal valore Kind 8 e ha una lunghezza fissa di 10 byte.

Il Timestamp viene utilizzato per due scopi principali:

- **Misurazione del Round-Trip Time (RTTM):** Il protocollo TCP deve conoscere il Round Trip Time della connessione per calcolare correttamente il Retransmission Timeout (RTO), ovvero il tempo dopo il quale un pacchetto che non è ancora stato riconosciuto (con ACK) deve

essere ritrasmesso.

- **Protection Against Wrapped Sequences (PAWS):** Il campo del Numero di Sequenza in TCP è un contatore a 32 bit che in reti multi-gigabit può esaurirsi in pochi secondi e ricominciare da zero (wrap around). Senza il timestamp, il ricevente potrebbe non essere in grado di distinguere se un segmento con un numero di sequenza già visto è un vecchio duplicato o un nuovo segmento con numero di sequenza avvolto.

L'opzione TCP Timestamp include due campi principali: il Timestamp Value (TSval) e il Timestamp Echo Reply (TSecr), entrambi di 4 byte. Il valore TSval rappresenta il timestamp corrente del mittente, mentre TSecr è utilizzato nei segmenti di acknowledgment per riportare il valore TSval ricevuto in precedenza. Il mittente include il TSval nel segmento TCP inviato. Quando il destinatario manda il pacchetto di Acknowledgment, copia il valore TSval nel campo TSecr del segmento ACK. In questo modo, il mittente, quando riceve l'ACK, può calcolare l'RTT sottraendo il valore TSecr dal suo valore corrente del timestamp.

Il valore del Timestamp Value (TSval) non è un orario assoluto, ma un contatore monotono crescente (timestamp clock) che come definito dal RFC 7323 [24] alla sezione 4.1, *Values of this clock MUST be at least approximately proportional to real time.* Quindi ogni sistema ha un proprio timestamp clock differente dagli altri. Solitamente nei sistemi operativi moderni, il timestamp viene incrementato con una frequenza di 1000Hz (una volta al millisecondo) e inizializzato a partire da un valore casuale al boot del sistema.

Per verificare la possibilità di fingerprinting basato sui TCP Timestamp, è stato utilizzato *Euriclea* [25], un tool open-source sviluppato specificamente per l'identificazione delle impronte dei mittenti nelle competizioni CTF A/D. Euriclea analizza passivamente i segmenti TCP usando una coda nfqueue di Netfilter e ne estrae il valore TSval e il tempo di ricezione del pacchetto. Utilizzando questi dati, Euriclea calcola la differenza tra il tempo di ricezione (unix timestamp in millisecondi) e il valore TSval del pacchetto. Questa

differenza rimane costante per tutte le connessioni TCP provenienti dallo stesso mittente, permettendo a Euriclea di associare un'impronta univoca (haiku) a ciascun mittente.

Durante la fase di testing, Euriclea e Wireshark hanno permesso di evidenziare differenze sostanziali nel modo in cui i diversi sistemi operativi gestiscono l'opzione Timestamp, influenzando la possibilità di fingerprinting:

- **Windows 11:** Non invia l'opzione TCP Timestamp (Kind 8) di default, rendendo impossibile il fingerprinting basato su questo campo.

- **macOS:** Randomizza il valore iniziale (offset) del timestamp per ogni nuova connessione TCP (sessione TCP definita dal 3-way handshake, scambio di dati e terminazione). Quindi ogni volta che viene stabilita una nuova connessione TCP (anche verso lo stesso indirizzo IP e porta), il valore iniziale del timestamp è casuale. Questo comportamento impedisce di correlare due connessioni diverse alla stessa macchina sorgente, poiché non esiste un offset comune che le leghi.

- **Linux:** Il comportamento del timestamp in Linux è regolato dal parametro di sistema `net.ipv4.tcp_timestamps`. Il valore predefinito (1), come definito nella documentazione del kernel Linux [26], abilita i timestamp TCP randomizzando l'offset iniziale del timestamp per ogni connessione. Tuttavia, durante i test, catturando i pacchetti con Wireshark su una vulnbox, è stato osservato che un mittente specifico aveva sempre lo stesso offset iniziale del timestamp. Quando invece i pacchetti venivano catturati su un'altra vulnbox, lo stesso mittente presentava un offset iniziale diverso. Approfondendo l'analisi e analizzando il codice sorgente del kernel Linux 4.8, è stato possibile comprendere il motivo di questo comportamento. L'algoritmo di generazione dell'offset iniziale del timestamp utilizza una funzione di hash che prende in input solamente la coppia di indirizzi IP sorgente e destinazione, ma ignora le porte TCP. Questo comportamento consente ai giocatori di fingerprintare il mittente in modo affidabile, poiché tutte le connessioni

di uno specifico mittente verso la vulnbox avranno la stessa differenza tra il tempo di ricezione e il valore TSval.

```
u32 secure_tcp_ts_off(const struct net *net, __be32 saddr, __be32
↪  daddr)
{
    if (READ_ONCE(net->ipv4.sysctl_tcp_timestamps) != 1)
        return 0;

    ts_secret_init();
    return siphash_2u32((__force u32)saddr, (__force u32)daddr,
                &ts_secret);
}

/* This will initiate an outgoing connection. */
int tcp_v4_connect(struct sock *sk, struct sockaddr *uaddr, int
↪  addr_len)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct inet_sock *inet = inet_sk(sk);
    ...
    WRITE_ONCE(tp->tsoffset,
                           secure_tcp_ts_off(net, inet->inet_saddr,
                                             inet->inet_daddr));
    ...
}

/* Compute TCP options for SYN packets. This is not the final
network wire format yet. */
static unsigned int tcp_syn_options(struct sock *sk, struct sk_buff
↪  *skb,
                                    struct tcp_out_options *opts,
                                    struct tcp_key *key)
{
    struct tcp_sock *tp = tcp_sk(sk);
    ...
    if (likely(timestamps)) {
                opts->options |= OPTION_TS;
                opts->tsval = tcp_skb_timestamp_ts(tp->tcp_usec_ts,
                ↪  skb) + tp->tsoffset;
                opts->tsecr = tp->rx_opt.ts_recent;
                remaining -= TCPOLEN_TSTAMP_ALIGNED;
          }
    ...
}
```

Codice 4.8: Codice Kernel Linux 6.17.9 per il calcolo del TCP Timestamp

Nello snippet di codice 4.8 vengono mostrate le tre funzioni del kernel Linux coinvolte nel calcolo del TCP Timestamp TSval per il pacchetto SYN iniziale di una connessione TCP. La funzione `tcp_v4_connect` viene chiamata quando viene stabilita una nuova connessione TCP e imposta l'offset del timestamp chiamando la funzione `secure_tcp_ts_off`. La funzione `secure_tcp_ts_off` calcola l'offset del timestamp utilizzando una funzione di hash (siphash) che prende in input solamente gli indirizzi IP sorgente e destinazione, ignorando le porte TCP. Infine, la funzione `tcp_syn_options` costruisce l'header TCP del pacchetto SYN, includendo il campo TSval calcolato sommando il timestamp clock corrente con l'offset della connessione.

Nella figura 4.2 è mostrato un esempio di output di Euriclea in esecuzione su una vulnbox durante la fase di testing dell'infrastruttura. La vulnbox riceve pacchetti TCP sulla porta 3000 da due diversi mittenti: un sistema macOS e un sistema Linux. Tutti i pacchetti hanno come indirizzo IP sorgente quello del router centrale (10.254.0.1) a causa del SNAT. I primi due pacchetti provengono da un sistema Linux, ma da due connessioni TCP distinte (una generata con `curl` e l'altra con `nc`). Nonostante ciò, Euriclea riesce a identificare che entrambi i pacchetti provengono dallo stesso mittente e associare l'haiku *hot-ugly* a quel sistema Linux. I successivi due pacchetti provengono da un sistema macOS, anch'essi da due connessioni TCP distinte. In questo caso, Euriclea non riesce a correlare i due pacchetti allo stesso mittente, generando due haiku differenti: *late-foolish* e *soft-blue*.

```
root@ctf-vulnbox-team1:~/euriclea/cmd/nfqueue# ./nfqueue -queue 0
[3]     10.254.0.1 -> 172.18.0.2 (hot-ugly:83838): GET / HTTP/1.1..Host: 10.60.1.1:3000..User-Agent: curl/8.14.1..Accept: */*....
[9]     10.254.0.1 -> 172.18.0.2 (hot-ugly:83838): test da linux.
[15]    10.254.0.1 -> 172.18.0.2 (late-foolish:16157): GET / HTTP/1.1..Host: 10.60.1.1:3000..User-Agent: curl/8.7.1..Accept: */*....
[21]    10.254.0.1 -> 172.18.0.2 (soft-blue:66582): test da macos.
```

Figura 4.2: Euriclea mostra le impronte dei mittenti basate sui TCP Timestamp

Per mitigare la vulnerabilità del *TCP Timestamp Fingerprinting*, sono state analizzate due soluzioni distinte, valutandone l'efficacia e l'impatto sulle prestazioni della rete.

4.5.1 nftables

La soluzione più immediata consiste nel rimuovere completamente l'opzione *TCP Timestamp* dai pacchetti in transito verso le vulnbox utilizzano nftables sul router di gioco. La regola mostrata nello snippet di codice intercetta tutti i pacchetti TCP con il flag SYN e li modifica rimuovendo l'opzione *TCP Timestamp*. Impedendo l'invio dell'opzione Timestamp nel 3-way handshake, il mittente non invierà mai il campo TSval nei pacchetti successivi della connessione.

```
table inet ctf_mangle {
    chain strip_tcp_timestamps {
        type filter hook forward priority mangle; policy accept;

        # Remove TCP timestamps
        ip daddr 10.60.0.0/16 tcp flags syn reset tcp option
        ↪  timestamp
    }
}
```

Codice 4.9: Regola nftables per rimuovere l'opzione *TCP Timestamp*

Questa tecnica è efficace nel prevenire il fingerprinting basato sui timestamp, poiché elimina completamente il campo che viene utilizzato per l'identificazione del mittente.

4.5.2 Proxy TCP

La scelta definitiva è ricaduta sull'installazione di un proxy TCP layer 4 (HAProxy) sul router centrale, che funge da intermediario tra i giocatori e le vulnbox. HAProxy riceve le connessioni TCP dai giocatori e dal gameserver, stabilisce una nuova connessione verso la vulnbox e inoltra i dati tra i due endpoint. Poiché la connessione verso la vulnbox è generata interamente dallo stack TCP/IP del router centrale, tutti i parametri del protocollo TCP dipendono esclusivamente dal router centrale su cui è in esecuzione HAProxy. Questa soluzione più sofisticata garantisce l'anonimato completo del mittente, eliminando qualsiasi possibilità di fingerprinting basata anche su altre ca-

ratteristiche dello stack TCP/IP originale. I giocatori non sono consapevoli della presenza del proxy, dato che inviano i pacchetti direttamente all'indirizzo IP della vulnbox (10.60.X.1). Il router centrale utilizzando la funzionalità di Destination NAT (DNAT) di nftables reindirizza il traffico destinato alle vulnbox verso il proxy HAProxy in esecuzione sulla stessa macchina. Il DNAT permette di modificare l'indirizzo IP di destinazione dei pacchetti in transito. Lo snippet di codice 4.10 mostra l'esempio di una regola DNAT che reindirizza tutto il traffico TCP dei servizi di gioco di due vulnbox verso HAProxy. Quando un pacchetto TCP destinato ai servizi di gioco (porte 1337, 3000, 3001, 8000, 8443) arriva al router centrale con destinazione 10.60.X.1, la regola di DNAT modifica l'indirizzo IP di destinazione con 10.60.X.254 (indirizzo del router nella sottorete della vulnbox).

```
1 table inet ctf_nat {
2   chain dnat_to_haproxy {
3     type nat hook prerouting priority dstnat; policy accept ;
4         ip daddr 10.60.0.1 tcp dport { 1337,3000,3001,8000,8443 }
          ↪  dnat to 10.60.0.254
5         ip daddr 10.60.1.1 tcp dport { 1337,3000,3001,8000,8443 }
          ↪  dnat to 10.60.1.254
6     }
7 }
```

Codice 4.10: Regole di DNAT per reindirizzare il traffico verso HAProxy

HAProxy è configurato per ascoltare sulle interfacce di rete del router centrale e inoltrare le connessioni verso le rispettive vulnbox (10.60.X.1) in base alla porta di destinazione. Lo snippet di codice 4.11 mostra un esempio di configurazione di HAProxy che definisce i backend e frontend per due servizi di gioco esposti dalle vulnbox. Con la direttiva frontend si specifica l'indirizzo IP e la porta su cui HAProxy deve ascoltare le connessioni in ingresso, mentre con la direttiva backend si definisce l'indirizzo IP e la porta della vulnbox verso cui inoltrare il traffico.

```
frontend f_team0_p3000
    bind 10.60.0.254:3000
    default_backend b_team0_p3000

backend b_team0_p3000
    server s_team0_3000 10.60.0.1:3000

frontend f_team1_p3000
    bind 10.60.1.254:3000
    default_backend b_team1_p3000

backend b_team1_p3000
    server s_team1_3000 10.60.1.1:3000
```

Codice 4.11: Esempio di configurazione di HAProxy

No.	Time	Source	Destination	Protocol	Timestamp value	Info
1	2025-11…	10.254.0.1	10.60.1.1	TCP	396156303	51603 → 3000 [SYN,…
7	2025-11…	10.254.0.1	10.60.1.1	TCP	4177356607	48190 → 3000 [SYN]…
13	2025-11…	10.254.0.1	10.60.1.1	TCP		34136 → 3000 [SYN]…
19	2025-11…	10.254.0.1	10.60.1.1	TCP		51765 → 3000 [SYN,…
25	2025-11…	10.254.0.1	10.60.1.1	TCP	3050863814	45144 → 3000 [SYN]…
31	2025-11…	10.254.0.1	10.60.1.1	TCP	3050865986	45154 → 3000 [SYN]…

Figura 4.3: Confronto dei valori TCP Timestamp in tre scenari di mitigazione

La figura 4.3 mostra il confronto dei valori TCP Timestamp nei pacchetti SYN catturati con Wireshark in tre scenari distinti: senza alcuna mitigazione, con la rimozione dell'opzione TCP Timestamp tramite nftables e utilizzando HAProxy come proxy TCP. Per ogni scenario, sono stati inviati pacchetti TCP da due host distinti verso la vulnbox in esame. Nello scenario senza mitigazione (colore blu), i pacchetti SYN provenienti dai due host mostrano valori TSval differenti, permettendo di distinguere le due fonti. Nello scenario con la rimozione dell'opzione TCP Timestamp tramite nftables (colore rosso), i pacchetti SYN non contengono più l'opzione Timestamp, rendendo impossibile il fingerprinting basato su questo campo. Infine, nello scenario con HAProxy (colore verde), i pacchetti SYN ricevuti dalla vulnbox presentano lo stesso offset TSval, poiché entrambi i pacchetti sono stati generati dallo

stack TCP/IP del router centrale su cui è in esecuzione HAProxy. Questo dimostra l'efficacia della soluzione basata su proxy nel garantire l'anonimato completo del mittente.

Capitolo 5

Risultati

5.1 Valutazione delle prestazioni delle soluzioni di mitigazione

Per valutare l'impatto delle soluzioni di anonimizzazione del traffico sulle prestazioni della rete, sono stati condotti dei test utilizzando *iperf3* per misurare la velocità di trasferimento dati tra due VM ubicate sullo stesso nodo Proxmox (con CPU Intel i5-8400). Per assicurarsi che il test misurasse le prestazioni del router centrale, le VM client e server di iperf3 sono state collocate in due sottoreti VLAN differenti, costringendo il traffico a passare attraverso il router di gioco per l'instradamento.

I test sono stati eseguiti in tre scenari distinti: normalizzazione di base con SNAT e TTL reset, con la rimozione dell'opzione TCP Timestamp tramite nftables e utilizzando HAProxy come proxy TCP. Ogni test è stato eseguito tre volte per garantire l'affidabilità dei risultati e sono stati misurati sia il throughput in Gbps che l'utilizzo di un singolo core della CPU del router centrale. I risultati ottenuti sono riassunti nella tabella seguente:

I risultati indicano che la rimozione dell'opzione TCP Timestamp tramite nftables non ha avuto alcun impatto significativo sul throughput della rete, mantenendo una velocità di trasferimento di 16.3 Gbps, identica a quella ottenuta senza alcuna mitigazione. Al contrario, l'utilizzo di HAProxy come

Scenario	Throughput	Uso CPU 1 core
Solo SNAT e TTL Reset	16.3 Gbps	25%
Rimozione Timestamp (nftables)	16.3 Gbps	25%
Proxy TCP (HAProxy)	10.4 Gbps	99%

Tabella 5.1: Risultati dei test di throughput nei diversi scenari di mitigazione

proxy TCP ha comportato una riduzione del throughput a 10.4 Gbps e un elevato utilizzo di un singolo core della CPU, ma offre una protezione completa contro ogni forma di fingerprinting basata sullo stack TCP/IP originale. Questo overhead è dovuto al fatto che HAProxy agisce come intermediario, gestendo due connessioni TCP distinte per ogni flusso di dati (client-proxy e proxy-server) anziché limitarsi a modificare i pacchetti in transito come avviene con nftables. Considerando che le vulnbox sono distribuite su più nodi Proxmox, dove la velocità della rete fisica tra i nodi potrebbe rappresentare un collo di bottiglia maggiore, la soluzione basata su HAProxy risulta comunque adeguata per garantire l'anonimato senza compromettere le prestazioni della competizione.

5.2 Sessione di prova dell'infrastruttura

Per verificare il corretto funzionamento del progetto è stata organizzata una sessione di prova durante un incontro di Ulisse Lab, presso i laboratori di Ingegneria. L'obiettivo principale era verificare la stabilità del sistema in uno scenario reale, replicando le condizioni di una vera competizione CTF Attack and Defense. Hanno partecipato alla simulazione 24 studenti, suddivisi in 4 team da 6 persone ciascuno, più il NOP Team. L'infrastruttura è stata distribuita interamente sul cluster Proxmox di Ulisse Lab, utilizzando 3 nodi fisici connessi tramite una rete Gigabit Ethernet. Ogni nodo era equipaggiato con hardware piuttosto datato, con CPU Intel i5-2400, 16GB di RAM e dischi HDD in rete. Si tratta di specifiche hardware di livello medio-basso che permettono di verificare l'efficienza dell'infrastruttura an-

che in presenza di risorse limitate. La creazione delle VM con Terraform è stata istantanea, grazie all'utilizzo del linked clone, mentre il provisioning con Ansible ha impiegato circa 45 minuti a causa delle prestazioni limitate dei dischi HDD. A ogni macchina virtuale sono stati allocati 2 core di CPU e 3GB di RAM, risorse minime sufficienti per eseguire i servizi senza compromettere le prestazioni.

La prova ha avuto una durata di 2 ore, durante le quali non si sono verificati rallentamenti o interruzioni del servizio. Tutti i partecipanti sono riusciti a connettersi alla rete di gioco usando i profili VPN generati automaticamente e a interagire con le proprie vulnbox senza problemi. Tutti i servizi vulnerabili sono rimasti disponibili per l'intera durata della competizione e il gameserver ha gestito correttamente la checker routine e l'aggiornamento della classifica in tempo reale. Durante la prova è stato anche verificato il corretto funzionamento delle tecniche di anonimizzazione del traffico di rete. L'analisi del traffico catturato sulle vulnbox ha confermato che tutte le comunicazioni provenienti dal router centrale apparivano con lo stesso indirizzo IP sorgente e che l'offset del TCP Timestamp era identico per tutte le connessioni. Questo ha reso impossibile il fingerprinting dei mittenti, garantendo l'equità della competizione. Il risultato della prova dimostra come l'utilizzo di un'infrastruttura IaC distribuita su più nodi permetta di ottenere un sistema scalabile e affidabile, anche utilizzando hardware non di ultima generazione.

Capitolo 6

Conclusioni e sviluppi futuri

Il presente lavoro di tesi si è concluso con la progettazione e lo sviluppo di un'infrastruttura scalabile e automatizzata per competizioni CTF Attack and Defense, utilizzando strumenti di Infrastructure as Code quali Terraform e Ansible. La soluzione proposta si distingue da quelle esistenti per la sua completezza: questo sistema gestisce a 360 gradi l'intero ciclo di vita della competizione, dalla creazione delle macchine virtuali alla configurazione della rete, includendo la gestione del gameserver e le tecniche di anonimizzazione del traffico. Inoltre, l'utilizzo della connettività VPN consente ai partecipanti di accedere alla rete di gioco ovunque si trovino e di organizzare eventi sia in presenza sia online.

I risultati presentati nel capitolo precedente dimostrano che tutti gli obiettivi prefissati sono stati raggiunti. La sessione di prova condotta presso i laboratori ha confermato l'efficacia delle politiche di sicurezza adottate e la stabilità dell'infrastruttura anche in presenza di hardware non di ultima generazione. Inoltre, la funzionalità di distribuzione del carico su più nodi fisici assicura la scalabilità necessaria per organizzare eventi con un numero elevato di partecipanti. Dal punto di vista educativo, questo progetto contribuisce in modo concreto alla formazione di nuovi professionisti della cybersecurity, offrendo un ambiente realistico in cui sviluppare competenze pratiche in materia di sicurezza informatica. L'infrastruttura, infatti, oltre a poter essere

utilizzata per reali competizioni, è stata ideata proprio con l'obiettivo di migliorare la fase di preparazione dei nuovi partecipanti di CyberChallenge.IT presso l'Università di Bologna.

Nonostante i risultati positivi, sono stati individuati alcuni possibili sviluppi futuri. In primo luogo, si potrebbe migliorare la resilienza dell'infrastruttura nei confronti dei malfunzionamenti. Attualmente, il gamerouter rappresenta un *Single Point of Failure*: in caso di blocco della VM del router, l'intera rete di gioco diventerebbe inaccessibile. Una soluzione potrebbe consistere nell'introduzione di un meccanismo di *High Availability* (HA) che configuri un router di backup, pronto a intervenire tramite un sistema di failover automatico.

Un secondo sviluppo potrebbe riguardare l'inclusione di un Intrusion Detection System (IDS), uno strumento che consente di analizzare il traffico di rete in tempo reale e di generare alert in caso di comportamenti anomali, come scansioni di porte o attacchi DDoS, spesso vietati nelle competizioni.

Per migliorare ulteriormente la scalabilità dell'infrastruttura, si potrebbe valutare l'integrazione di provider di servizi di cloud computing tramite Terraform. Questo permetterebbe agli organizzatori di ridurre i costi legati all'acquisto di hardware fisico, noleggiando solo le risorse necessarie in base al numero di partecipanti.

Infine, si potrebbe realizzare una piattaforma web che permetta di gestire l'iscrizione dei team, la distribuzione delle credenziali delle vulnbox e delle configurazioni VPN ai partecipanti. Con quest'ultima aggiunta, l'intero processo di organizzazione di una competizione CTF Attack and Defense verrebbe automatizzato, semplificando ulteriormente il lavoro degli organizzatori.

Appendice A

Glossario

VM Virtual Machine (Macchina Virtuale) è un ambiente virtuale che simula il funzionamento di un computer fisico.

VXLAN Virtual Extensible LAN è una tecnologia di virtualizzazione di rete che consente di creare reti virtuali (overlay) sopra una rete fisica esistente (underlay).

VTEP VXLAN Tunnel Endpoint è un dispositivo che incapsula e decapsula i pacchetti VXLAN.

VPN Virtual Private Network permette di creare una connessione privata e sicura tra dispositivi attraverso Internet.

MTU Maximum Transmission Unit è la dimensione massima in byte di un pacchetto dati che può essere inviato su una rete.

SNAT Source Network Address Translation è una tecnica di rete che permette di modificare l'indirizzo IP sorgente dei pacchetti.

DNAT Destination Network Address Translation è una tecnica di rete che permette di modificare l'indirizzo IP di destinazione dei pacchetti.

Indirizzo MAC Media Access Control è un identificativo a 48 bit associato a un'interfaccia di rete.

TCP Transmission Control Protocol è un protocollo di rete di livello trasporto che rende affidabile la comunicazione tra mittente e destinatario.

TTL Time To Live è un campo dell'header IP che indica il numero massimo di hop che un pacchetto può attraversare prima di essere scartato.

IDS Intrusion Detection System è un sistema di sicurezza che monitora il traffico di rete per rilevare attività sospette.

SDN Software Defined Networking è un approccio alla gestione delle reti che separa il piano di controllo dal piano dati, permettendo una gestione centralizzata e programmabile della rete.

Jinja2 motore di template per Python che consente di generare file di configurazione dinamici con Ansible.

Header parte iniziale di un pacchetto di rete che contiene informazioni di controllo.

Payload parte di un pacchetto di rete che contiene i dati.

Provisioning processo di creazione e configurazione delle risorse di un'infrastruttura IT.

IaC Infrastructure as Code è una pratica di gestione di infrastrutture IT che permette di definire le risorse tramite codice.

QCOW2 QEMU Copy On Write è un formato di file immagine per macchine virtuali.

High Availability caratteristica di un sistema che mira a garantire un elevato livello di operatività.

Bibliografia

[1] "Vxlan packet format figure," accesso: 07-11-2025. [Online]. Available: https://www.analysisman.com/2018/05/vxlan-how-it-works.html

[2] I. Cisco Systems, "Introduction to vxlan," accesso: 09-11-2025. [Online]. Available: https://www.ciscolive.com/c/dam/r/ciscolive/global-event/docs/2024/pdf/BRKDCN-1621.pdf

[3] CyberChallenge.IT, "Regolamento finale nazionale cyberchallenge.it," 2025, accesso: 01-11-2025. [Online]. Available: https://ad.cyberchallenge.it/

[4] "Tcp segment structure," accesso: 16-11-2025. [Online]. Available: https://www.geeksforgeeks.org/computer-networks/services-and-segment-structure-in-tcp/

[5] E. U. A. for Cybersecurity (ENISA), "Enisa report - ctf events," ENISA, Tech. Rep., 2021.

[6] K. M. Kapp, *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education*, 1st ed. Pfeiffer & Company, 2012.

[7] A. Marczewski, *Gamification: A Simple Introduction*. Andrzej Marczewski, 2013. [Online]. Available: https://books.google.it/books?id=IOu9kPjlndYC

[8] A. Dabrowski, M. Kammerstetter, E. Thamm, E. Weippl, and W. Kastner, "Leveraging competitive gamification for sustainable fun and profit in security education," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/3gse15/summit-program/presentation/dabrowski

[9] OWASP, "Owasp top 10," accesso: 05-11-2025. [Online]. Available: https://owasp.org/www-project-top-ten/

[10] CyberChallenge.IT, "Regolamento cyberchallenge.it," 2025, accesso: 26-10-2025. [Online]. Available: https://cyberchallenge.it/rules/

[11] ——, "Cyberchallenge.it - statistiche ed edizioni," 2025, accesso: 26-10-2025. [Online]. Available: https://cyberchallenge.it/stats/

[12] CTFtime.org, "Ctftime - portale competizioni capture the flag," 2025, accesso: 10-11-2025. [Online]. Available: https://ctftime.org/

[13] A. Saxena, S. Singh, S. Prakash, T. Yang, and R. S. Rathore, "Devops automation pipeline deployment with iac (infrastructure as code)," in *2024 IEEE Silchar Subsection Conference (SILCON 2024)*. IEEE, Nov. 2024, p. 1–6. [Online]. Available: http://dx.doi.org/10.1109/SILCON63976.2024.10910699

[14] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348, Aug. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7348

[15] P. S. S. GmbH, "Proxmox ve documentation," accesso: 12-11-2025. [Online]. Available: https://pve.proxmox.com/wiki/Main_Page

[16] HashiCorp, "Terraform," accesso: 12-11-2025. [Online]. Available: https://developer.hashicorp.com/terraform

[17] I. Red Hat, "Ansible documentation," accesso: 12-11-2025. [Online]. Available: https://docs.ansible.com/

[18] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *Network and Distributed System Security Symposium*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:2590070

[19] F. Team, "Faust ctf platform," accesso: 10-11-2025. [Online]. Available: https://ctf-gameserver.org/

[20] M. Bauer, "saarctf - ctf game server," accesso: 10-11-2025. [Online]. Available: https://github.com/MarkusBauer/saarctf-servers

[21] Domysh, "Ctfbox," accesso: 10-11-2025. [Online]. Available: https://github.com/domysh/ctfbox

[22] bpg, "Terraform proxmox provider." [Online]. Available: https://github.com/bpg/terraform-provider-proxmox

[23] "Beszel monitoring software." [Online]. Available: https://beszel.dev/

[24] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323, Sep. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7323

[25] drank40, "Euriclea - tcp timestamp fingerprinting." [Online]. Available: https://github.com/drank40/euriclea

[26] T. L. K. Organization, "Linux kernel documentation," accesso: 18-11-2025. [Online]. Available: https://docs.kernel.org/networking/ip-sysctl.html

Ringraziamenti

Al termine di questo percorso di studi, desidero esprimere la mia gratitudine a tutte le persone che hanno contribuito alla mia crescita accademica e umana rendendo possibile la realizzazione di questa tesi.

In primo luogo, un sentito ringraziamento va al mio relatore, Prof. Marco Prandini e al correlatore, Prof. Andrea Melis, per la loro disponibilità e il supporto durante lo sviluppo di questo progetto. Un ringraziamento particolare va a Eyad Issa per avermi fornito l'accesso alle risorse del cluster e a tutto il gruppo di Ulisse Lab per l'opportunità che mi ha permesso di crescere professionalmente.

Un pensiero speciale va all'esperienza di CyberChallenge.IT 2025, che ha segnato indubbiamente una tappa fondamentale nel mio percorso accademico. Grazie ai tutor che mi hanno formato e soprattutto ai miei compagni di squadra per l'impegno e i risultati ottenuti alla finale nazionale. È grazie a voi se mi sono avvicinato al mondo della cybersecurity e delle CTF, rendendo di fatto possibile la realizzazione di questo progetto di tesi. Mi auguro che i futuri partecipanti che rappresenteranno la nostra sede possano continuare a portare in alto il nome dell'Università di Bologna.

Non posso dimenticare i miei compagni di università: Samu, Lollo, Omar, Alice, Greg e Diego. Grazie per le risate che hanno alleggerito i momenti di stress e per tutte le giornate trascorse insieme in laboratorio tra studio, esperimenti e progetti di ogni genere. La vostra amicizia ha reso questo viaggio molto più piacevole e indimenticabile.

Ringrazio di cuore la mia famiglia per il sostegno costante, per la pazienza

dimostrata nelle inevitabili giornate di tensione e per avermi dato la possibilità e i mezzi di intraprendere questo percorso di studi credendo sempre in me anche quando io stesso facevo fatica a farlo.

Infine, voglio ringraziare tutte le persone che mi sono state vicine in questi anni, ma in particolare Laura che mi ha sempre supportato e incoraggiato a dare il massimo. Grazie per essere stata al mio fianco, per aver condiviso con me sia le soddisfazioni sia le difficoltà e per avermi sostenuto in ogni momento di incertezza.