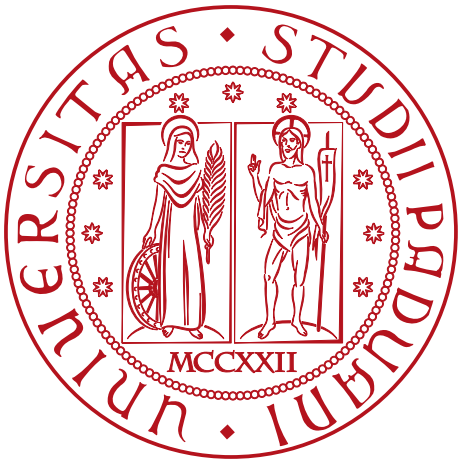
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di un'infrastruttura per competizioni Capture The Flag Attacco/Difesa

Tesi di laurea

Relatore

Prof. Eleonora Losiouk

Laureando

Augusto Cesare Zanellato

2000555

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante l'attività di stage interno, della durata di 300 ore, dal laureando Augusto Cesare Zanellato presso l'Università degli Studi di Padova.
L'obiettivo era la progettazione di un'infrastruttura atta ad organizzare competizioni di sicurezza informatica di tipo Capture The Flag, nello specifico secondo la modalità Attacco/Difesa. Nello specifico sono state sviluppate le componenti software necessarie per la gestione della rete di gioco, dei team e dei partecipanti, dei servizi e dei server a cui i partecipanti hanno accesso.
Il fine ultimo del progetto è quello di migliorare l'addestramento dei partecipanti al progetto CyberChallenge.IT che ogni anno rappresentano l'Università degli Studi di Padova nella competizione nazionale.
All'interno del documento sono descritte le tecnologie utilizzate, le scelte progettuali effettuate e le modalità di utilizzo dell'infrastruttura sviluppata, evidenziando e motivando le scelte progettuali ed architetturali effettuate.

"Non ti agitare."

— Anonimo

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Professoressa Eleonora Losiouk, relatrice della mia tesi, per la fiducia che ha riposto in me affidandomi lo svolgimento del progetto.

Desidero ringraziare con affetto i miei genitori per il sostegno, i sacrifici, il grande aiuto e per la loro vicinanza durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici e tutte le persone che mi sono state vicine in questi anni per tutti i bellissimi momenti passati insieme e le avventure vissute.

Padova, Settembre 2023 Augusto Cesare Zanellato

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

All'interno del seguente capitolo verranno descritti il contesto applicativo e le motivazioni che hanno portato alla realizzazione del progetto, oltre ad un'introduzione ai concetti chiave necessari per la comprensione del lavoro svolto.

1.1 Concetti chiave

1.1.1 Competizioni Capture The Flag

Una competizione Capture The Flag (CTF) è una competizione di sicurezza informatica che consiste nello scovare vulnerabilità nascoste dagli organizzatori all'interno di sistemi informatici ben delimitati, e infine sfruttarle per ottenere informazioni segrete, chiamate in gergo *flag*. Il nome deriva dal gioco omonimo "Cattura la Bandiera", in cui due squadre si sfidano per conquistare la bandiera dell'avversario.
Esistono varie modalità di gioco, le più comuni sono:

- *Jeopardy*: i partecipanti devono risolvere una serie di prove di vario tipo, ognuna delle quali assegna un punteggio. Le prove sono organizzate in categorie, e i partecipanti possono scegliere liberamente quale prova risolvere;
- *Attacco/Difesa*: i partecipanti sono divisi in squadre, ogni squadra dispone di accesso ad un server in cui sono presenti delle applicazioni vulnerabili uguali per tutte le squadre. L'obiettivo di ogni squadra è difendere i propri servizi dagli attacchi, e allo stesso tempo attaccare i servizi delle altre squadre. La modalità *Attacco/Difesa* viene inoltre descritta in dettaglio nel paragrafo dedicato: 1.1.1.1.

Nello specifico il progetto si concentra sulla modalità *Attacco/Difesa*, in quanto è quella per cui è più difficile trovare occasioni di allenamento, dato che, a differenza delle competizioni *Jeopardy* non è possibile allenarsi in autonomia usando le varie piattaforme online in quanto sono richiesti degli avversari.

1.1.1.1 Modalità Attacco/Difesa

A differenza della modalità *Jeopardy*, dove gli organizzatori della competizione rendono disponibili le prove da risolvere, una competizione *Attacco/Difesa* richiede che ad ogni squadra partecipante sia assegnato un server su cui è presente una copia delle stesse

applicazioni vulnerabili. Inoltre la competizione è suddivisa in *round* i quali hanno una durata fissa (di solito variabile tra i 20 e i 120 secondi), ad ogni round gli organizzatori controllano il corretto funzionamento di ogni servizio appartenente ad ogni squadra e, se tutto funziona correttamente, ci aggiungono una *flag* diversa per ognuno; inoltre ogni squadra dovrà attaccare ogni servizio di ogni altra squadra per ottenere la rispettiva flag ed inviarla agli organizzatori in modo da ricevere i punti corrispondenti.

Di seguito sono indicate le componenti tipiche di un'infrastruttura che ospita una competizione *Attacco/Difesa*:

- *Router di gioco*: è un router a cui sono collegate le squadre partecipanti, i server a loro assegnati, e il server degli organizzatori. Il router è configurato in modo da mascherare l'indirizzo IP sorgente del traffico diretto verso i server dei giocatori;
- *Server dei team* (o Vulnbox$_{[G]}$): è il server assegnato ad ogni squadra su cui sono in esecuzione i servizi vulnerabili. I team hanno accesso come *root* al proprio server e possono installare cose o modificare configurazioni a loro piacimento;
- *Servizi vulnerabili* (o più brevemente *servizi*): sono le applicazioni vulnerabili che vengono installate su ogni *vulnbox* dagli organizzatori. I team possono modificare a piacimento il comportamento dei servizi in modo da risolvere le vulnerabilità riscontrate. Generalmente un servizio ha uno scopo definito (ad esempio può essere una piattaforma per prendere appunti) e solitamente contiene qualche tipo di dato segreto che non deve essere pubblicamente accessibile, lì è dove verrà inserita la *flag*;
- *Checker*$_{[G]}$: è un programma eseguito dagli organizzatori ad ogni *round*, si occupa di aggiungere una *flag* ad ogni servizio appartenente ad ogni team e soprattutto, si occupa anche di verificare che i servizi siano raggiungibili e correttamente funzionanti, in modo che i team non possano renderli irraggiungibili o non funzionanti così da non perdere più *flag*;
- *Gamesystem*$_{[G]}$: è il server gestito dagli organizzatori che si occupa di gestire la competizione, in particolare:
  - esegue i *checker* ad ogni round;
  - riceve le *flag* inviate dai team;
  - aggiorna la classifica dei team in base alle *flag* ricevute.

Durante la competizione ogni squadra ha i seguenti compiti:

- Ad ogni round lanciare attacchi verso tutte le altre squadre sfruttando le vulnerabilità trovate all'interno dei servizi;
- Difendersi dagli attacchi delle altre squadre, applicando modifiche ai servizi per risolvere le vulnerabilità trovate e/o installando strumenti di difesa come *firewall* o *IDS*. Per rilevare gli attacchi delle altre squadre è inoltre molto diffuso monitorare il traffico in ingresso sul proprio server;

Il punteggio di ogni squadra è calcolato tenendo conto delle *flag* rubate e perse per ogni servizio, infine viene moltiplicato per il coefficiente di Service Level Agreement (SLA) il quale è calcolato come il rapporto tra il tempo in cui il servizio è stato raggiungibile e il tempo totale della competizione.

1.1.2 Progetto CyberChallenge.IT

Il progetto CyberChallenge.IT[6] è un programma di formazione sulla sicurezza informatica rivolto a studenti della scuola secondaria di secondo grado e universitari, è inoltre riconosciuto come progetto di valorizzazione delle eccellenze dal Ministero dell'Istruzione. Nell'anno 2023 più di 40 sedi universitarie hanno aderito, formando in totale più di 800 studenti.
Il progetto è articolato nelle seguenti fasi:

1. **Test di ammissione**: consiste in una prove di selezione online divisa in un test sulle abilità logico-matematiche e algoritmiche di base e in un test di programmazione che consiste nella risoluzione di diversi problemi in un tempo limitato. Al termine di questa fase i 20 migliori studenti di ogni sede vengono ammessi a partecipare alla fase di addestramento;

2. **Addestramento**: consiste in una serie di lezioni teoriche e laboratori pratici tenuti da esperti, durante le quali gli studenti affrontano diversi temi tra cui: *Web Security*, *Network Security*, *Software Security*, *Hardware Security* e *Crittografia*. È inoltre a disposizione degli studenti una piattaforma online di allenamento in cui è presente una grande quantità di challenge in formato *Jeopardy* di vario tipo;

3. **Gara locale**: consiste in una CTF in formato *Jeopardy* durante la quale si affrontano sfide simili a quelle viste durante la fase di addestramento. Al termine della gara i migliori 6 studenti di ogni sede vengono selezionati per avanzare alla fase successiva;

4. **Gara nazionale**: consiste in una CTF in formato *Attacco/Difesa* durante la quale le squadre che rappresentano le università competono tra loro per ottenere il podio.

1.2 Organizzazione della relazione

Il secondo capitolo descrive gli scopi e gli obiettivi da raggiungere con il progetto;

Il terzo capitolo analizza le soluzioni esistenti nei rispettivi punti di forza e criticità;

Il quarto capitolo approfondisce e motiva le scelte progettuali effettuate;

Il quinto capitolo descrive le scelte implementative effettuate;

Il sesto capitolo approfondisce le modalità di verifica e validazione del progetto;

Nel settimo capitolo vengono riassunti i risultati ottenuti e vengono proposte possibili estensioni future.

Nell'appendice A viene riportata una guida al gioco di competizioni CTF *Attacco/Difesa* per principianti, in modo da fornire un'introduzione al contesto applicativo del progetto.

Durante la stesura della relazione sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;

- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: parola$_{[G]}$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione del progetto

2.1 Il background

Ho partecipato al progetto CyberChallenge due volte: prima come studente nell'edizione 2022 e poi come insegnante nell'edizione 2023 per gli argomenti *Network Security* e *Software Security*. Avendo visto il progetto da entrambi i punti di vista è sorta l'idea di proporre, come attività di stage interno, lo sviluppo di una piattaforma per potenziare l'allenamento nella specialità *Attacco/Difesa* durante la fase di addestramento del progetto, in quanto le modalità di gioco sono radicalmente diverse da quelle a cui i partecipanti sono abituati svolgendo *challenge* in modalità *Jeopardy*.

2.2 L'idea

Il progetto descritto all'interno della relazione nasce dall'idea di creare una piattaforma di allenamento per la modalità *Attacco/Difesa* che possa essere utilizzata dagli studenti durante la fase di addestramento in preparazione alla gara nazionale.
L'obiettivo del progetto è lo sviluppo di un'infrastruttura di gioco il più simile possibile a quella usata nella gara nazionale del progetto CyberChallenge.IT, in modo che gli studenti possano allenarsi in modo efficace e prepararsi al meglio per la competizione, testando nel modo più agevole possibile eventuali strumenti che hanno sviluppato per la gara.
Inoltre, l'infrastruttura di gioco deve essere il più possibile automatizzata nella fase di configurazione, in modo da ridurre al minimo il lavoro manuale da parte degli istruttori.
È inoltre desiderabile implementare un sistema di monitoraggio dell'infrastruttura di gioco, in modo da poter rilevare eventuali problemi e risolverli il prima possibile.
Infine, è desiderabile che l'infrastruttura di gioco sia il più possibile scalabile, in modo da poter essere utilizzata anche in competizioni con un numero di partecipanti maggiore.

2.3 Gli obiettivi

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- *O* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- *D* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- *F* per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.
Si prevede lo svolgimento dei seguenti obiettivi:

- Obbligatori
  - *O01*: infrastruttura per CTF Attacco/Difesa;
  - *O02*: Documentazione del progetto;
  - *O03*: Dimostrazione con una demo del lavoro svolto;
- Desiderabili
  - *D01*: Seconda demo;
  - *D02*: Automatizzazione del setup lato rete ed infrastruttura;
- Facoltativi
  - *F01*: Monitoraggio in tempo reale dell'infrastruttura;

2.4 Pianificazione del lavoro

Il periodo di svolgimento del progetto è stato dal 15 Maggio 2023 al 7 Luglio 2023, per la durata totale di 300 ore.
La pianificazione delle attività settimanali era la seguente:

- **Prima Settimana (24 ore)**
  - Analisi delle soluzioni già esistenti;
  - Approfondimento delle tecnologie da utilizzare;
- **Seconda Settimana (36 ore)**
  - Approfondimento delle tecnologie da utilizzare;
  - Sviluppo software;
- **Terza Settimana (40 ore)**
  - Sviluppo software;
- **Quarta Settimana (40 ore)**

  - Progettazione e sviluppo infrastruttura;

- **Quinta Settimana (40 ore)**
  - Sviluppo servizi per prima demo;
- **Sesta Settimana (40 ore)**
  - Sviluppo servizi per prima demo;
  - Prima demo;
- **Settima Settimana (40 ore)**
  - Analisi dell'andamento della prima demo e conseguenti miglioramenti;
  - Sviluppo dei servizi per seconda demo;
- **Ottava Settimana (40 ore)**
  - Sviluppo dei servizi per seconda demo;
  - Seconda demo;

2.5 Variazioni rispetto alla pianificazione iniziale

Durante lo svolgimento del progetto la pianificazione iniziale è stata leggermente rivista. È stato infatti deciso di effettuare una sola demo alla fine del progetto, invece di due come inizialmente previsto. Questo ha permesso di risparmiare tempo che è stato utilizzato per automatizzare il setup dell'infrastruttura.
Quindi in termini di modifiche agli obiettivi elencati nella sezione 2.3 si ha che:

- *D01* è stato eliminato;

Capitolo 3

Analisi delle soluzioni esistenti

Di seguito verranno analizzate le soluzioni open source esistenti per l'organizzazione di CTF *Attacco/Difesa*, nel valutarle verrà tenuto conto dei seguenti aspetti:

- **Modularità**: la soluzione deve essere divisa in moduli che svolgono funzioni specifiche, in modo da poter essere estesa e modificata facilmente;
- **Estensibilità**: la soluzione deve essere facilmente estendibile, in modo da poter aggiungere funzionalità non presenti;
- **Documentazione**: la soluzione deve essere ben documentata, in modo da poter essere utilizzata, compresa e modificata facilmente;
- **Portabilità tra cloud**: la soluzione deve essere facilmente portabile tra cloud provider diversi;
- **Similarità con la piattaforma di CyberChallenge.IT**: dato che l'obiettivo del progetto è lo sviluppo di una piattaforma simile a quella di CyberChallenge.IT è preferibile scegliere una soluzione con caratteristiche e modalità d'interfacciamento simili.

3.1 iCTF Framework

iCTF Framework[20] è un framework sviluppato dal team Shellphish[26], il team accademico dell'Università della California, Santa Barbara. È stato sviluppato con l'obiettivo di essere usato per la competizione iCTF, è scritto in Python e si basa su Flask[14]. La gestione dell'infrastruttura è delegata a Vagrant[28], Terraform[27] e Ansible[2], l'uso è possibile solo su cloud AWS[1].

Vantaggi

- Molto scalabile, è stata usata per edizioni di iCTF con più di 300 squadre partecipanti;
- Corredato di strumenti per la creazione di *vulnbox*.

Svantaggi

- La documentazione non è completa e lascia diverse parti del framework senza spiegazione;
- È fortemente dipendente da AWS[1];
- Sono presenti molte sezioni di codice legacy, non più utilizzate, che rendono difficile la comprensione del codice.

3.2 FAUST ctf-gameserver

ctf-gameserver[13] è un gamesystem sviluppato dal team FAUST[12], il team accademico dell'Università Friedrich-Alexander di Erlangen-Nuremberg. È scritto in Python usando il framework Django[7]. È usato per la competizione annuale FAUST CTF.

Vantaggi

- Documentazione completa e ben scritta, eccezion fatta per la parte relativa al *deploy*;
- Codice ben strutturato e facilmente comprensibile;

Svantaggi

- Mancanza di strumenti per la creazione di *vulnbox* e il loro collegamento in rete;
- Non sono presenti indicazioni su come eseguire il *deploy*.

3.3 EnoEngine

EnoEngine[10] è un gamesystem modulare sviluppato da ENOFLAG[11], il team accademico dell'Università Tecnica di Berlino. È scritto in C# ed ha come particolarità il fatto di non fornire un'interfaccia web, difatti l'interfaccia è un progetto separato che comunica con EnoEngine tramite il database PostgreSQL[22]. Viene usato per l'organizzazione della competizione annuale ENOWARS.

Vantaggi

- Molto modulare;
- Buone performance.

Svantaggi

- Manca un frontend di riferimento;
- Il deploy non è ben documentato.

3.4 Hackerdom Checksystem

checksystem[18] è un gamesystem sviluppato da Hackerdom[17], il team accademico dell'Università Statale degli Urali. È scritto in Perl usando il framework Mojolicious. Viene usato per la competizione annuale RuCTF.

Vantaggi

- Scalabile
- Fornisce strumenti per la creazione di *vulnbox* e il loro collegamento in rete.

Svantaggi

- Scritto in Perl, un linguaggio definito ironicamente come *write-only*;
- Documentazione non completa.

3.5 ForcAD

ForcAD[15] è un gamesystem sviluppato dal team *C4T BuT S4D*[4]. È scritto in Python usando il framework Flask[14]. È inoltre completo di tutte le componenti

Vantaggi

- Molto scalabile: è pensato per essere scalato orizzontalmente su più macchine con il minimo sforzo;
- Supporta nativamente il deploy in ambienti cloud usando Docker[8] e Docker Compose[9];
- Resiliente in caso di crash: è in grado di ripristinare lo stato della competizione in caso di crash totale o riavvio forzato della macchina su cui è in esecuzione;
- Buona documentazione e codice ben strutturato;
- È la piattaforma più simile a quella usata dal progetto CyberChallenge.IT[1].

Svantaggi

- Non fornisce strumenti per la creazione di *vulnbox*;
- Fornisce solo parzialmente strumenti per la gestione della rete di gioco.

3.6 Conclusione

Dopo aver analizzato le soluzioni esistenti è stato scelto ForcAD come base per lo sviluppo del progetto, in quanto è la soluzione che meglio soddisfa i requisiti individuati.

[1] Diverso tempo dopo la scelta di ForcAD come base del progetto ho scoperto, parlando con uno degli organizzatori, che le prime versioni della piattaforma di CyberChallenge erano basate su una versione di ForcAD ampiamente modificata.

Capitolo 4

Progettazione

In questo capitolo verranno descritte le scelte progettuali effettuate nello sviluppo del progetto, in particolare verranno trattate le scelte relative alla configurazione di rete, all'infrastruttura, al monitoraggio e alle modifiche applicate a ForcAD[15].

4.1 Progettazione della rete di gioco

Per la rete di gioco è stato scelto di usare WireGuard[31] come Virtual Private Network (VPN) per due motivi:

- È molto leggero e veloce, quindi richiede poche risorse sul router di gioco;
- La configurazione e la gestione sono assai più semplici rispetto ad altre soluzioni come OpenVPN o IPSec.
- È la stessa VPN usata dal progetto CyberChallenge.IT, quindi l'uso è propedeutico alla preparazione della gara e inoltre risulta permessa dal firewall di dipartimento.

È stato inoltre scelta una configurazione con un singolo router VPN a cui tutto è collegato (Figura 4.1) piuttosto che una configurazione con un router VPN per ogni squadra (Figura 4.2) per i seguenti motivi:

- Riduce in modo significativo il numero di server da gestire (si passa da $n + 1$ a 1, dove n è il numero di squadre partecipanti);
- Riduce la complessità delle regole del firewall;
- WireGuard[31] permette molto più *throughput* rispetto ad OpenVPN a parità di hardware, quindi diventa possibile gestire tutto il traffico con un singolo router;

4.1.1 Assegnazione degli indirizzi IP

L'assegnazione degli indirizzi IP (in Tabella 4.1) è stata fatta con l'unico criterio di essere uguale a quella usata dal progetto CyberChallenge.IT, con l'eccezione della macchina di monitoraggio e della rete della giuria in quanto non raggiungibili dai partecipanti.

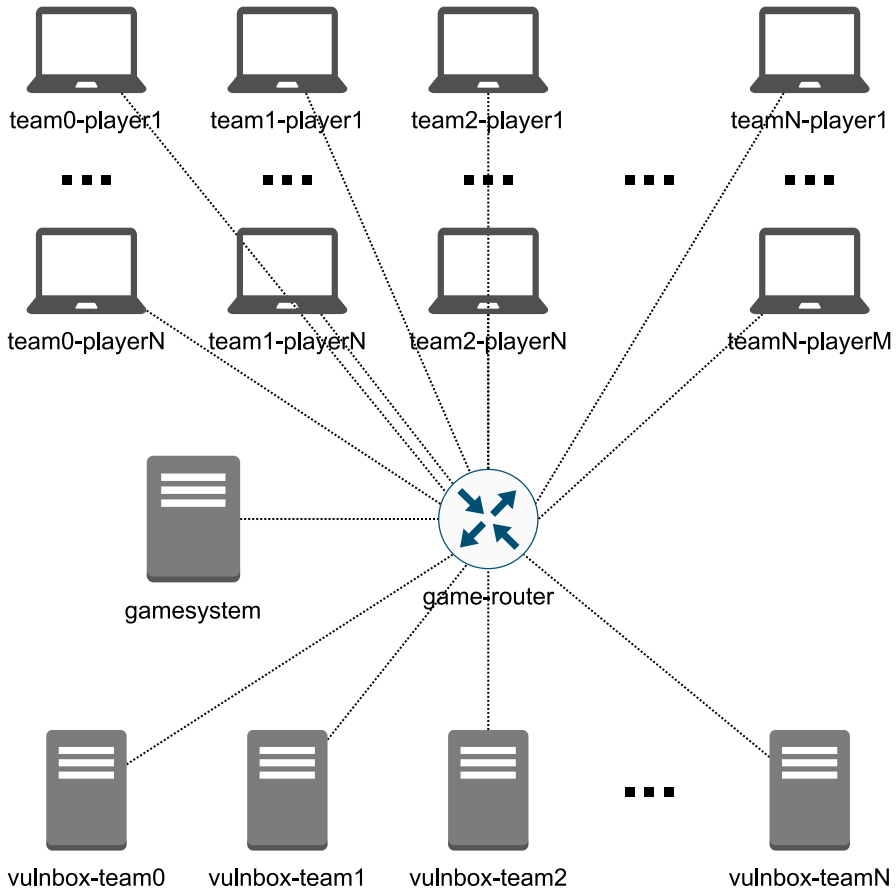


Figura 4.1: Esempio di rete di gioco con singolo router VPN

Di seguito è presente un diagramma di rete completo (Figura 4.3) che include tutti i collegamenti[1] e gli indirizzi IP utilizzati.

[1]Tutti i collegamenti sono da intendersi effettuati mediante VPN

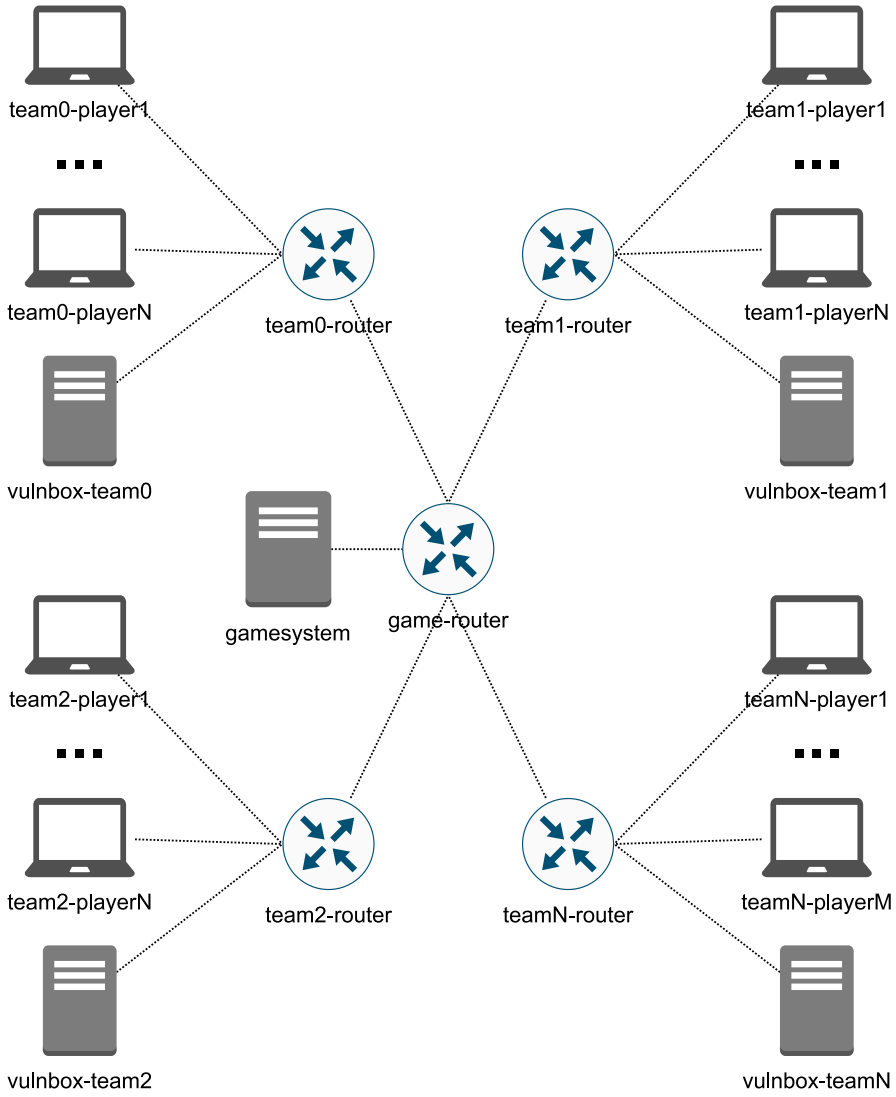


Figura 4.2: Esempio di rete di gioco con un router VPN per squadra

Tabella 4.1: Assegnazione degli indirizzi IP

Indirizzo (CIDR)	Descrizione
10.10.0.1/32	Gamesystem
10.10.10.0/24	Rete della giuria
10.11.0.1/32	Macchina di monitoraggio
10.60.n.1/32	Vulnbox del team n
10.80.n.0/24	Rete del team n
10.254.0.1/32	Router di gioco

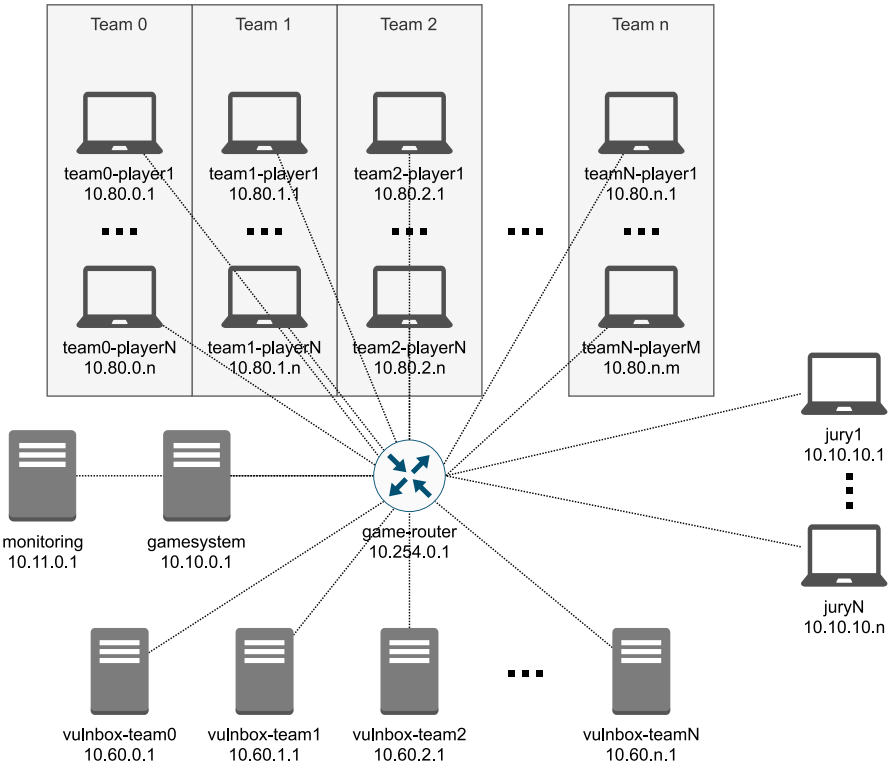


Figura 4.3: Architettura finale della rete di gioco

4.2 Progettazione dell'infrastruttura

Per l'automatizzazione della creazione dei server è stato scelto di usare Ansible[2] al posto di Terraform[27] per i seguenti motivi:

- Semplicità di utilizzo;
- Non necessita di sincronizzazione dello stato;
- Può venire usato anche per la configurazione dei server.

Per l'infrastruttura è stato scelto di usare server virtuali forniti dal provider cloud Hetzner Cloud[19] per i seguenti motivi:

- Ottimo rapporto qualità/prezzo;
- Disponibilità di *API REST* per automatizzare la gestione delle risorse;
- Supporto *first-party* per Ansible[2];
- I server virtuali sono molto veloci da creare e distruggere, quindi è possibile creare un'infrastruttura completa in pochi minuti;

Inoltre Hetzner Cloud[19] fornisce anche server virtuali basati su architettura ARM64, la quale permette di ottenere circa il doppio delle risorse hardware a parità di prezzo rispetto ai classici server basati su architettura AMD64.

4.2.1 Specifiche hardware dei server

Nella Tabella 4.2 sono riportate le specifiche hardware dei server virtuali usati per le varie componenti dell'infrastruttura di gioco.

Tabella 4.2: Specifiche hardware dei server virtuali

Ruolo	Codice	Specifiche
Gamesystem	`cax31`	8 vCPU ARM64, 16GB RAM, 160GB SSD
Macchina di monitoraggio	`cax21`	4 vCPU ARM64, 8GB RAM, 80GB SSD
Vulnbox	`cpx21`	3 vCPU AMD64, 4GB RAM, 80GB SSD
Router di gioco	`cax11`	2 vCPU ARM64, 4GB RAM, 40GB SSD

Di seguito le motivazioni delle scelte effettuate:

- **Gamesystem**: è stato scelto un hardware molto potente in quanto avrà in esecuzione ForcAD[15] e tutte le sue dipendenze, tra cui ben due database diversi (PostgreSQL[22] e Redis[25]), inoltre è necessario che sia in grado di gestire un carico di lavoro molto elevato in quanto dovrà gestire tutte le richieste dei team e le frequenti esecuzioni dei checker. È stata scelta una macchina ARM64 per motivi prettamente di costo, ma qualora fosse necessario, è possibile l'uso di un server con architettura AMD64;
- **Macchina di monitoraggio**: è stato scelto un hardware sufficientemente potente per poter eseguire lo stack di monitoraggio (dettagliato nella sezione 4.3). È stata scelta una macchina ARM64 in quanto tutti i software compatibili supportano quell'architettura e non sono presenti motivi particolari per preferire AMD64;

- **Vulnbox**: è stato scelto un hardware con specifiche sufficienti ad eseguire i servizi vulnerabili. È stato scelto di usare una macchina AMD64 in quanto è la stessa architettura usata dal progetto CyberChallenge.IT;
- **Router di gioco**: è stata scelto un hardware di potenza limitata in quanto WireGuard[31] ha una richiesta di risorse molto bassa. È stata scelta una macchina ARM64 in quanto non sussistono motivi per preferire AMD64.

4.3 Progettazione del monitoraggio

Per monitorare l'infrastruttura e l'andamento della competizione è stato scelto di usare lo stack Prometheus[23] e Grafana[16] per i seguenti motivi:

- Sono software molto diffusi e ben supportati dalla comunità;
- Richiedono relativamente poche risorse;
- ForcAD[15] espone nativamente delle metriche compatibili con Prometheus[23];
- Sono disponibili esportatori di metriche Prometheus[23] per tutti i componenti dell'infrastruttura;

Nello specifico è stato scelto di monitorare le seguenti metriche:

- Utilizzo delle risorse hardware dei server che ospitano l'infrastruttura di gara (vulnbox escluse);
- Statistiche del traffico di rete;
- Tutte le metriche nativamente supportate da ForcAD[15];
- Metriche relative alle performance di PostgreSQL[22] e Redis[25].

4.4 Modifiche da apportare a ForcAD

Dopo la scelta di ForcAD[15] come base per il progetto sono state individuate le seguenti modifiche da apportare per renderlo simile alla piattaforma usata dal progetto CyberChallenge.IT:

- Modifica del protocollo di ricezione delle *flag* da parte del gamesystem;
- Aggiunta del supporto al NOP Team[G];
- Aggiunta del supporto per un orario di fine della competizione;
- Aggiunta delle *API* utilizzate dal *frontend* della classifica di CyberChallenge.IT in modo che i partecipanti possano sviluppare degli strumenti che prendono informazioni dalla classifica;

Capitolo 5

Implementazione e codifica

È stato scelto di automatizzare per prima cosa il *setup* dell'infrastruttura, in seguito l'implementazione della rete seguita dallo sviluppo delle modifiche a ForcAD[15] e infine l'implementazione del monitoraggio in quanto è l'ordine delle dipendenze tra i vari componenti.

5.1 Implementazione dell'infrastruttura

Come prima cosa è stato necessario implementare l'infrastruttura, in particolare è stato necessario implementare la creazione, la configurazione e la distruzione delle macchine virtuali su cui verranno eseguiti i vari componenti.

Sono stati implementati i seguenti *playbook* Ansible[2]:

- `generate_creds.yml`: genera, a partire dalla configurazione della competizione, le credenziali necessarie per accedere alle vulnbox. Genera inoltre una coppia di chiavi SSH che viene usata da Ansible[2] per accedere ai server virtuali per configurarli e dagli organizzatori qualora sia necessario accedere manualmente alle vulnbox o alle componenti dell'infrastruttura di gioco;
- `setup.yml`: invoca `generate_creds.yml` qualora necessario, poi crea e configura tutte le macchine virtuali richieste installando eventuali dipendenze e configurando i servizi necessari;
- `teardown.yml`: distrugge tutte le macchine virtuali create dal playbook `setup.yml`;

5.2 Implementazione della rete

5.2.1 Configurazioni WireGuard

È stato scritto un modulo Python che si occupa di generare le configurazioni WireGuard[31] per i partecipanti, le vulnbox e tutta l'infrastruttura di gioco. Inizialmente è stato testato l'uso di una soluzione già esistente, wggen[30], ma sono state riscontrate alcune limitazioni che hanno portato alla scrittura di un modulo ad-hoc, come ad esempio il fatto che non supporti la generazione di configurazioni per più subnet contemporaneamente.

Il nuovo modulo ha mantenuto il nome `wggen` perché inizialmente erano state apportate solo modifiche minori, in seguito è stato però totalmente riscritto.
La nuova versione è basata sul concetto di gruppi e sottogruppi ognuno dei quali ha dei *peer* assegnati, dove un gruppo è un insieme di sottogruppi e ogni sottogruppo rappresenta una subnet. I gruppi e i sottogruppi possono anche avere un solo componente qualora necessario. Di seguito alcuni esempi pratici:

- `team`: gruppo con n sottogruppi dove n è il numero di squadre partecipanti, ogni sottogruppo rappresenta la *subnet* di ogni squadra, ogni sottogruppo ha un numero di *peer* pari al numero di componenti della squadra. Al gruppo è assegnata la subnet `10.80.0.0/16`, ogni sottogruppo prende una subnet grande `24` bit. Ogni squadra ha quindi assegnata una *subnet* del tipo `10.80.`n`.0/24` dove n è il numero della squadra e ogni partecipante ha l'indirizzo IP `10.80.`n`.`m dove m è il numero del giocatore all'interno della squadra;
- `vulnbox`: gruppo con n sottogruppi dove n è il numero di squadre partecipanti, ogni sottogruppo ha un solo *peer* che rappresenta la *vulnbox* della squadra. Al gruppo è assegnata la subnet `10.60.0.0/16`, ad ogni sottogruppo viene assegnata una subnet grande `24` bit di cui viene preso solo l'indirizzo IP terminante con `.1`, quindi ogni vulnbox avrà l'indirizzo IP `10.60.`n`.1` dove n è il numero della squadra;
- `gamesystem`: gruppo con 1 sottogruppo composto da 1 *peer* al quale viene assegnato l'indirizzo IP `10.10.0.1`, è stato scelto di mantenere l'organizzazione in sottogruppi per mantenere la coerenza con le altre configurazioni;
- `jury`: gruppo con 1 sottogruppo composto da n peer (dove n è il numero degli organizzatori) al quale viene assegnato un indirizzo IP dalla subnet `10.10.10.0/24`.

Le configurazioni WireGuard[31] vengono quindi copiate e applicate sui server tramite il playbook Ansible[2] `setup.yml` citato nella sezione 5.1.

5.2.2 Regole firewall

Sono stati individuati i seguenti requisiti per le regole firewall:

- Tutto il traffico verso le vulnbox deve essere soggetto a Source Network Address Translation (SNAT) e TTL Reset[G] in modo da impedire ai partecipanti di bloccare l'accesso alla loro vulnbox da parte delle altre squadre;
- Gli organizzatori possono sempre accedere a qualsiasi macchina collegata alla rete di gioco;
- La macchina che si occupa del monitoraggio deve sempre poter accedere a tutte le macchine della rete di gioco;
- Il gamesystem deve sempre poter accedere a tutte le vulnbox;
- Tutti devono poter accedere al gamesystem;
- Le subnet assegnate ai partecipanti devono essere isolate tra di loro in modo da evitare attacchi tra squadre diretti ai pc personali;
- La rete di gioco deve supportare due stati:

  - **Rete chiusa**: ogni squadra può collegarsi solo alla propria vulnbox;
  - **Rete aperta**: ogni squadra può collegarsi a tutte le vulnbox;

Per l'ultimo requisito state quindi definite due *chain* di iptables: `closed-network` e `open-network`, ognuna con le seguenti regole:

- `closed-network`: `-m set -match-set team-vulnbox src,dst -j ACCEPT`, usando l'*ipset* `team-vulnbox` il quale associa l'indirizzo della *subnet* di ogni squadra a quello della rispettiva vulnbox;
- `open-network`: `-d 10.60.0.0/16 -j ACCEPT`, quindi tutto il traffico verso le vulnbox è permesso.

Dai precedenti requisiti sono state ricavate le seguenti regole:

```
# Vieta tutto il traffico di default,
# poi verranno aggiunte le esclusioni
-P INPUT DROP
-P FORWARD DROP
# Permette le connessioni gia' stabilite
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
# Permette l'accesso degli organizzatori a tutto
-A FORWARD -s 10.10.10.0/24 -j ACCEPT
# Permette l'accesso della macchina di monitoraggio a tutto
-A FORWARD -s 10.11.0.1 -j ACCEPT
# Permette l'accesso del gamesystem alle vulnbox
-A FORWARD -s 10.10.10.0/24 -d 10.60.0.0/16 -j ACCEPT
# Permette a tutti l'accesso al gamesystem
-A FORWARD -d 10.10.0.0/24 -j ACCEPT
# Effettua TTL reset sul traffico verso le vulnbox
-A POSTROUTING -t mangle -o game -j TTL --ttl-set 137 -d
    10.60.0.0/16
# Effettua SNAT sul traffico verso le vulnbox
-A POSTROUTING -t nat -o game -j SNAT --to 10.254.0.1 -d
    10.60.0.0/16
```

Nelle regole appena descritte il gamesystem viene trattato come una *subnet* pur essendo una singola macchina in modo da permettere la scalabilità dell'esecuzione dei checker su più macchine qualora diventasse necessario.

Al momento della configurazione della rete viene impostata di default sullo stato di *rete chiusa* aggiungendo la regola *iptables* `-A FORWARD -j closed-network`. Per passare allo stato di *rete aperta* è necessario rimuovere la regola precedente e aggiungere la regola `-A FORWARD -j open-network`.

5.3 Implementazione delle modifiche a ForcAD

Prima di esporre le modifiche apportate a ForcAD[15] verrà brevemente esposta l'architettura originale del sistema.

5.3.1 Architettura di ForcAD

Nella Figura 5.1 è rappresentata in modo semplificato l'architettura originale di ForcAD[15], in particolare sono stati omessi i seguenti collegamenti tra i vari componenti:

- `http-receiver` → `Redis`: le flag ricevute vengono salvate in Redis[25] in modo da poter verificare eventuali futuri duplicati velocemente;
- `Celery` → `Redis`: il risultato dell'ultima esecuzione dei checker è sempre disponibile su Redis[25] in modo da poterlo recuperare velocemente;

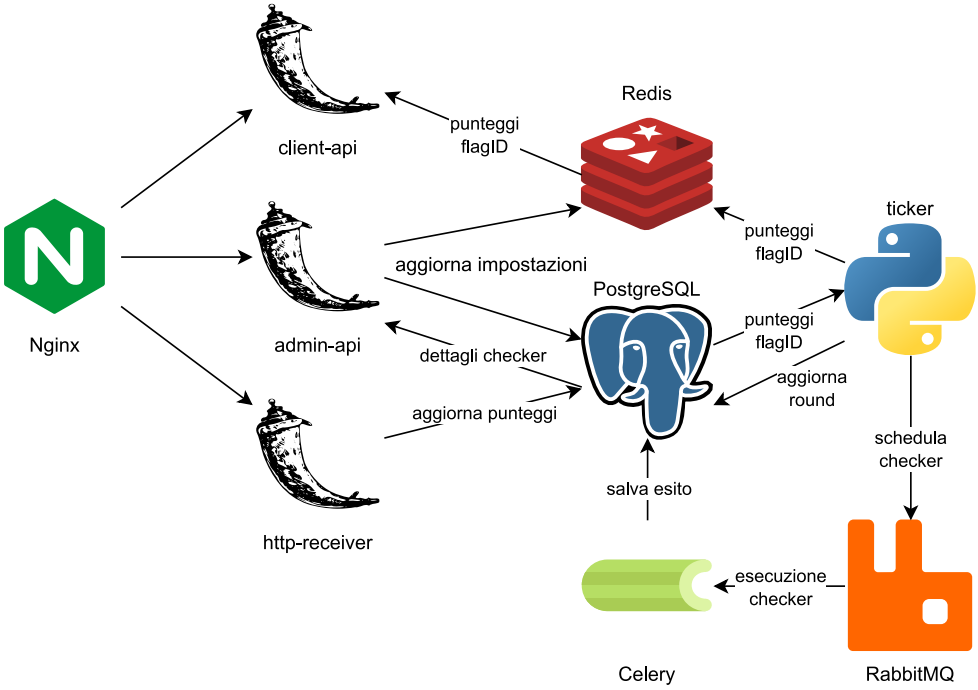


Figura 5.1: Architettura originale di ForcAD

Di seguito vengono descritti i componenti principali:

- Nginx[21]: si occupa di fare da *reverse proxy* verso gli altri componenti, in particolare verso `client-api`, `admin-api` e `http-receiver`, inoltre si occupa di servire i file statici necessari per il corretto funzionamento della classifica. Inoltre, qualora uno dei servizi fosse scalato orizzontalmente su più istanze svolge anche la funzione di *load balancer*. Svolge inoltra la funzione di *rate limiter* per evitare sovraccarico dei servizi;
- PostgreSQL[22]: database relazionale che contiene tutte le informazioni relative alla competizione, in particolare contiene le informazioni relative alle squadre, ai giocatori, alle flag e ai risultati dei checker;
- Redis[25]: database chiave-valore che viene usato come *cache* per migliorare le prestazioni di ForcAD[15] e per salvare temporaneamente i risultati dei checker prima che vengano salvati su PostgreSQL[22];
- RabbitMQ[24]: *message broker* che viene usato per gestire l'esecuzione dei checker in modo asincrono;

- Celery[5]: *task queue* che si occupa di gestire l'esecuzione dei checker in modo asincrono consumando i messaggi da RabbitMQ[24]. Supporta agevolmente l'uso di più processi *worker* che possono essere in esecuzione anche su server separati;
- `client-api`: è un microservizio scritto in Python usando il framework Flask[14] che si occupa di fornire ai partecipanti i dati necessari alla visualizzazione della classifica e i dati necessari allo svolgimento degli attacchi contro le altre squadre;
- `admin-api`: altro microservizio basato su Flask[14], viene utilizzato dagli organizzatori per vedere informazioni dettagliate sull'esecuzione dei checker per la diagnostica di eventuali errori e per la gestione delle squadre;
- `http-receiver`: altro microservizio basato su Flask[14], si occupa di ricevere le *flag* inviate dai partecipanti, di verificarne la validità e, nel caso siano valide, di accreditare i punti alla squadra attaccante;
- `ticker`: è un componente del gamesystem scritto in Python che si occupa di programmare l'esecuzione dei checker inviando, quando appropriato, messaggi su RabbitMQ[24], gestisce lo stato della competizione e fa avanzare i vari *round*, al termine di ognuno dei quali aggiorna la classifica e le informazioni necessarie alle squadre per effettuare gli attacchi;

Ognuno dei componenti precedentemente citati è eseguito all'interno di un container Docker[8] e qualora necessario può essere scalato orizzontalmente in modo indipendente dagli altri componenti. L'orchestrazione dei vari componenti necessari al funzionamento di ForcAD[15] è gestita da Docker Compose[9], anche se in modo *opaco* in quanto è necessario l'uso di una Command Line Interface (CLI) dedicata per gestire i vari componenti: ForcAD[15] fornisce infatti uno *script* Python chiamato `control.py` il quale permette di configurare e controllare l'esecuzione dei vari componenti che lo compongono.

5.3.2 Modifiche apportate a ForcAD

Come descritto nella sezione 4.4 sono state apportate le seguenti modifiche a ForcAD[15]:

5.3.2.1 Modifica protocollo ricezione *flag*

È stata la prima modifica apportata in quanto è quella di minore entità, in particolare alle risposte del servizio `http-receiver` è stato aggiunto un campo `success` che indica se l'elaborazione della *flag* inviata ha avuto esito positivo o meno. Non verrà descritta ulteriormente proprio in virtù della piccola entità.

5.3.2.2 Aggiunta supporto NOP Team

È stato introdotto il supporto al NOP Team, per raggiungere lo scopo è stata aggiunta un'opzione di configurazione applicabile ad ogni squadra registrata la quale indica se la squadra è un NOP Team oppure no, successivamente è stato modificato il servizio `http-receiver` per considerare non valide le flag rubate da un NOP Team restituendo un messaggio di errore adeguato.

5.3.2.3 Aggiunta supporto orario di fine competizione

È stato aggiunto il supporto per l'orario di fine competizione, in particolare è stato aggiunta un'opzione di configurazione del gioco la quale indica l'orario di fine e successivamente è stato modificato il servizio `ticker` per considerare la competizione terminata qualora l'orario corrente sia successivo a quello di fine competizione e interrompere quindi la periodica esecuzione dei checker. Inoltre è stato modificato `http-receiver` per smettere di accettare *flag* dopo la fine della competizione.

5.3.2.4 Implementazione *API* usate da CyberChallenge.IT

È stata la modifica di entità maggiore a ForcAD[15] in quanto ha richiesto prima di tutto il *reverse engineering* delle *API* utilizzate dal progetto CyberChallenge.IT per il funzionamento della classifica in quanto non sono ufficialmente documentate, successivamente è stato necessario implementarle seguendo la specifica ricavata.
Per perseguire lo scopo è stato aggiunto un nuovo microservizio scritto in Python usando il framework Flask[14] chiamato `ccit-api` il quale si occupa di fornire i dati necessari alla visualizzazione della classifica secondo le specifiche di CyberChallenge.IT, nello specifico espone i seguenti *endpoint*:

- `/api/status`: Espone informazioni sullo stato della competizione, nello specifico momento di inizio e fine, durata dei *round* e *round* corrente;
- `/api/scoreboard/table/<round>`: Espone la classifica completa del *round* specificato, nello specifico per ogni servizio appartenente ad ogni squadra espone il punteggio totale, il numero di *flag* rubate, quelle perse e la percentuale di SLA, inoltre mostra anche il punteggio totale di ogni squadra;
- `/api/scoreboard/chart/<round>`: Espone l'andamento nel tempo del punteggio totale delle prime 10 squadre al *round* specificato;
- `/api/scoreboard/team/table/<squadra>`: Espone per la squadra specificata una cronologia del punteggio totale e di ogni servizio compreso di *flag* rubate, quelle perse e percentuale di SLA;
- `/api/scoreboard/team/chart/<squadra>`: Espone l'andamento del punteggio di ogni servizio della squadra specificata a partire dall'inizio della competizione;

5.3.3 Automatizzazione del setup

La configurazione e l'esecuzione di ForcAD[15] è integrata nel *playbook* Ansible[2] `setup.yml` citato nella sezione 5.1.

5.4 Implementazione del monitoraggio

La fase di implementazione del monitoraggio è composta da due parti: prima l'installazione e configurazione di Prometheus[23] e relativi *exporter* seguita dalla creazione delle *dashboard* su Grafana[16].

5.4.1 Installazione e configurazione di Prometheus

L'installazione dello stack di monitoraggio è stata automatizzata tramite il *playbook* Ansible[2] `setup.yml` citato nella sezione 5.1, nello specifico:

- Viene creato un server virtuale denominato `monitoring`;
- Viene installato Docker[8] e Docker Compose[9];
- Vengono caricati sul server virtuale i file di configurazione di Prometheus[23] e Grafana[16] e il relativo file `docker-compose.yml`;
- Viene eseguito il comando `docker-compose up -d` per avviare i vari servizi.

Sui server da monitorare (nello specifico: sul server che esegue il gamesystem, sul router WireGuard[31] e sulla stessa macchina che si occupa del monitoraggio) è stato installato il *node exporter* di Prometheus[23] tramite Ansible[2].

5.4.2 Creazione delle *dashboard* su Grafana

Sono state create le seguenti *dashboard* su Grafana[16]:

- Monitoraggio esecuzione dei checker;
- Statistiche su flag ricevute;
- Utilizzo risorse hardware dei server (Figura 5.2);
- Metriche sulle performance di PostgreSQL[22] (Figura 5.3);
- Metriche interne di Prometheus[23];
- Metriche sulle performance di Redis[25] (Figura 5.4);
- Traffico passante per il router WireGuard[31].

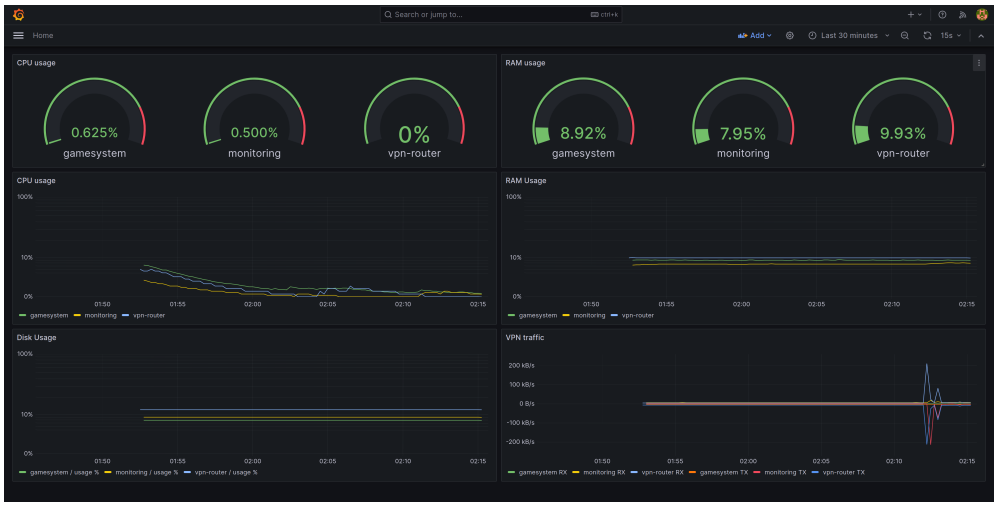


Figura 5.2: Dashboard di monitoraggio delle risorse hardware

Figura 5.3: Dashboard con metriche sulle performance di PostgreSQL

Figura 5.4: Dashboard con metriche sulle performance di Redis

Capitolo 6

Verifica e validazione

Il corretto funzionamento del progetto è stato verificato sia tramite l'uso di test automatici che tramite vere e proprie simulazioni di competizioni a cui hanno partecipato gli studenti che hanno rappresentato l'Università degli Studi di Padova all'edizione 2023 della competizione nazionale del progetto CyberChallenge.IT.

6.1 Test automatici

Per verificare il corretto funzionamento delle modifiche apportate a ForcAD[15] sono stati estesi i test automatici di sistema già presenti per testare le nuove funzionalità aggiunte.

6.2 Simulazioni di competizioni

Per verificare inoltre che l'implementazione delle *API* fosse davvero compatibile con quella del progetto CyberChallenge.IT è stata chiesta la collaborazione dei 6 studenti che hanno rappresentato l'Università degli Studi di Padova all'ultima edizione della competizione nazionale del progetto CyberChallenge.IT.
Per la simulazione gli studenti sono stati divisi in due squadre da 3 persone ciascuna, ognuna delle quali ha ricevuto un server su cui erano installati alcuni servizi vulnerabili presi da competizioni passate.
Le squadre hanno quindi avuto a disposizione 2 ore per testare il corretto funzionamento degli strumenti che avevano preparato per la competizione nazionale allo scopo di scovare eventuali differenze con il gamesystem usato da CyberChallenge.IT.
Non sono state riscontrate differenze significative, questo ha permesso di confermare che l'implementazione delle *API* è compatibile con quelle del progetto CyberChallenge.IT.

Capitolo 7

Conclusioni

7.1 Raggiungimento degli obiettivi

Tutti gli obiettivi definiti nella sezione 2.3, dopo le modifiche concordate nella sezione 2.5, sono stati raggiunti.

7.2 Consuntivo finale

Il preventivo iniziale di 300 ore è stato rispettato in tutta la sua articolazione, eccezion fatta per la quinta e la sesta settimana le quali sono state dedicate all'automatizzazione del setup dell'infrastruttura e al relativo monitoraggio in tempo reale al posto che alla preparazione della prima demo.

7.3 Conoscenze acquisite

Lo sviluppo di questo progetto mi ha permesso di apprendere nuove conoscenze riguardanti le tecnologie utilizzate, in particolare Redis[25], le *message queue* e la gestione della configurazione di un sistema composto da molteplici server mediante l'uso di Ansible[2].

7.4 Sviluppi futuri

In futuro potrebbe essere interessante sviluppare più servizi vulnerabili in modo da poter effettuare più simulazioni durante ogni edizione del progetto CyberChallenge.IT.

7.5 Valutazione personale

In conclusione valuto molto positivamente l'esperienza di stage interno, sia per le conoscenze acquisite che per la possibilità di migliorare concretamente la formazione dei futuri studenti che parteciperanno al progetto CyberChallenge.IT presso la sede di Padova.

Appendice A

Manuale Utente

o *CTF Attacco/Difesa per principianti*

Per giocare è necessario collegarsi alla rete VPN della competizione, per farlo è necessario installare il software WireGuard[31] e collegarsi usando il file di configurazione fornito dagli organizzatori. Ad esempio in un sistema Linux è sufficiente il seguente comando, assumendo che il file di configurazione si chiami `player2.conf` e sia nella cartella corrente:

```
sudo wg-quick up ./player2.conf
```

Attenzione: `./` prima del nome del file è necessario, altrimenti `wg-quick` prova a caricare il file di configurazione `/etc/wireguard/player2.conf` che non esiste.

Alla fine della competizione per disconnettersi dalla rete VPN è sufficiente il seguente comando:

```
sudo wg-quick down ./player2.conf
```

A.1 Rete e Setup

La competizione ha luogo nella *subnet* `10.0.0.0/8`, ogni squadra ha a disposizione una vulnbox all'indirizzo `10.60.`n`.1` dove n è il numero della squadra. Inoltre, ogni membro della squadra avrà sulla rete di gioco l'indirizzo `10.80.`$n.m$ dove m è il numero del giocatore all'interno della squadra e n è il numero della squadra.

La squadra numero 0 è il NOP Team, ovvero una squadra che non ha giocatori, ma solo una vulnbox la quale non riceve alcuna patch alle vulnerabilità; le flag rubate al NOP Team non vengono conteggiate nel punteggio finale e non danno punti.

Le vulnbox vengono fornite già collegate alla rete di gioco e con i servizi vulnerabili già installati, i partecipanti possono collegarsi alla propria vulnbox tramite il protocollo *SSH* usando l'utente `root` e la password fornitagli dagli organizzatori.

Il gamesystem è responsabile per la gestione della competizione, in particolare si occupa di:

- aggiungere le *flag* alle vulnbox;
- controllare l'integrità dei servizi;
- ricevere le flag rubate dai team in cambio di punti;
- aggiornare e mostrare la classifica.

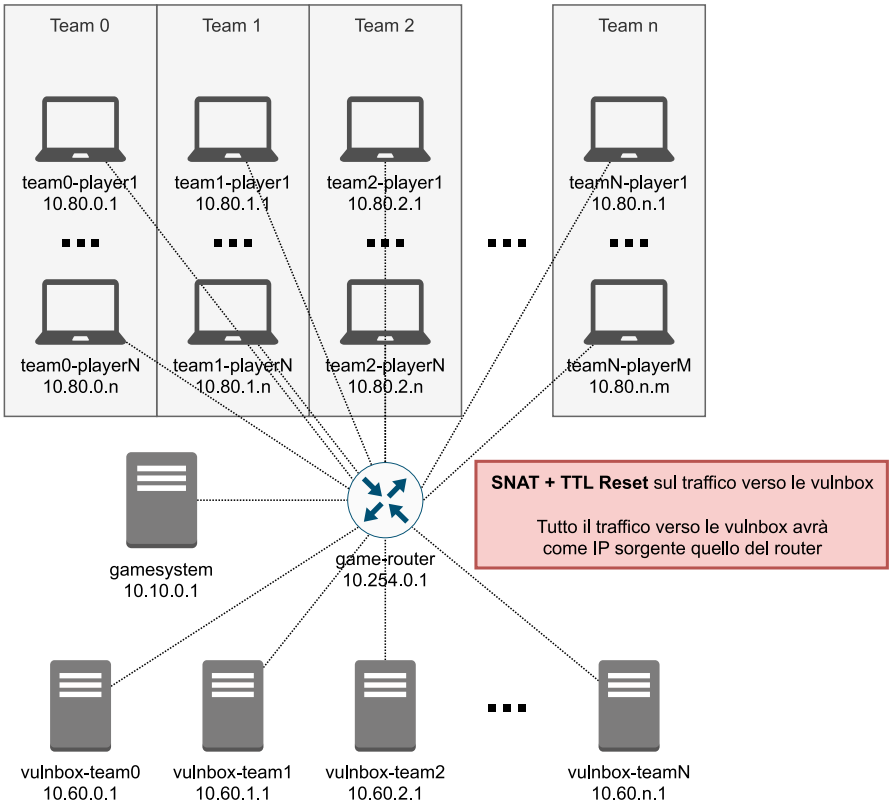


Figura A.1: Schema della rete di gioco come visibile ai partecipanti

I partecipanti devono quindi attaccare le vulnbox delle altre squadre ed ottenere le *flag*, dopodiché devono inviarle al gamesystem per ottenere i punti corrispondenti. Al tempo stesso i partecipanti devono difendere i servizi installati sulla propria vulnbox dagli attacchi delle altre squadre, per farlo possono installare strumenti di difesa come *firewall* o *IDS* o modificare il comportamento dei servizi per risolvere le vulnerabilità trovate.

I membri di ogni team sono liberi di usare la VPN di gioco per comunicare tra di loro, ma non è permesso comunicare con i membri di altre squadre.

Attenzione: in caso di problemi con la vulnbox causati da modifiche effettuate dai partecipanti, è possibile chiedere un reset agli organizzatori. È sconsigliato ricorrere al reset in quanto comporta la perdita di tutti i dati presenti sulla vulnbox e inoltre non è immediato, quindi è possibile che la vulnbox rimanga inutilizzabile per diversi minuti durante i quali verranno persi punti SLA.

A.2 Punteggio

La competizione è divisa in *round* della durata di 120 secondi, ad ogni round il gamesystem aggiunge una *flag* ad ogni servizio appartenente ad ogni squadra, inoltre controlla che i servizi siano raggiungibili e funzionanti provando a recuperare legittimamente[1] le flag inviate

[1] Il gamesystem non sfrutta le vulnerabilità presenti nei servizi per recuperare le flag

precedentemente.

Le squadre guadagnano punti in base alle flag rubate e perdono punti in base alle flag perse, inoltre il punteggio di ogni servizio viene moltiplicato per il coefficiente di SLA il quale è calcolato come il rapporto tra il tempo in cui il servizio è stato raggiungibile e il tempo totale della competizione.

Di seguito alcune considerazioni riguardo al sistema di punteggi:

- La percentuale di SLA non viene aggiunta al punteggio finale, ma viene usata per moltiplicare il punteggio di ogni servizio;
- Il punteggio attribuito ad ogni flag dipende dalla differenza di punteggio nel servizio tra le due squadre coinvolte;
- Si ottengono più punti rubando flag da squadre con un punteggio del servizio più alto;
- Si ottengono meno punti rubando flag da squadre con un punteggio del servizio più basso;
- Per ogni squadra la classifica mostra il punteggio totale e il punteggio per ogni servizio, oltre al numero di flag rubate e perse e la percentuale di SLA per ogni servizio;

Ad ogni round il gamesystem esegue al massimo 3 tipi di controlli per ogni servizio di ogni squadra:

- **Check SLA**: controlla che il servizio sia raggiungibile e funzionante, in caso contrario viene considerato *down* e non viene eseguito nessun altro controllo;
- **Put flag**: aggiunge una *flag* al servizio, in caso di errore viene considerato *down*;
- **Get flag**: recupera una delle *flag* precedentemente inserite, in caso di errore viene considerato *down*. Questo controllo viene eseguito solo se almeno uno dei precedenti n **put flag** ha avuto successo, dove n è il numero di round per cui una flag è considerata valida;

Ai fini della percentuale di SLA un servizio viene considerato correttamente funzionante solo se, durante un certo round, tutti e tre i controlli hanno avuto successo.

A.3 Flag

Le *flag* sono stringhe composte da 31 caratteri alfanumerici seguiti da un carattere `=`. Le flag rispettano sempre l'espressione regolare `/^[A-Z0-9]{31}=$/`. Le flag rubate possono essere inviate al gamesystem mediante una richiesta *HTTP PUT* all'indirizzo `http://10.10.0.1:8080/flags` come array di stringhe; le richieste devono inoltre essere autenticate usando l'*header HTTP* `X-Team-Token` con il token fornito agli organizzatori.

Ad esempio si può usare il seguente script *Python* per inviare una flag:

```python
import requests

TEAM_TOKEN = "4242424242424242"
flags = [
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=",
    "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB="
]

print(requests.put("http://10.10.0.1:8080/flags", headers={
    "X-Team-Token": TEAM_TOKEN
}, json=flags).text)
```

Per ogni flag inviata la richiesta precedente ritorna un oggetto *JSON* con il seguente formato:

```
{
    "msg": f"[{flag}] {message}",
    "flag": flag,
    "status": true/false
}
```

Dove `message` può avere i seguenti valori:

- `Accepted: X flag points`
- `Denied: invalid flag`
- `Denied: flag from nop team`
- `Denied: flag is your own`
- `Denied: flag too old`
- `Denied: flag already claimed`

Il codice di risposta 500 indica che la richiesta è malformata, che l'header con il token di autenticazione non è valido oppure che la competizione è finita, in ogni caso sarà presente nel corpo della risposta un messaggio di errore dettagliato.

A.4 Flag IDs

Alcuni servizi hanno dei *Flag IDs*, cioè delle informazioni addizionali che possono servire durante la scrittura di exploit, un esempio abbastanza frequente di *Flag ID* è lo username dell'account con cui il gamesystem ha inserito la flag nel servizio. I *Flag IDs* vengono forniti solo per le flag che sono valide al momento della richiesta.

È possibile ottenere una lista dei servizi e dei team facendo una richiesta *HTTP GET* all'indirizzo `http://10.10.0.1:8081/flagIds`. Per ottenere i *Flag IDs* è necessario filtrare almeno per servizio o per team (è possibile applicare entrambe le tipologie di filtri).

Ad esempio per richiedere i *Flag IDs* per il servizio `service1` si può fare una richiesta *HTTP GET* all'indirizzo `http://10.10.0.1:8081/flagIds?service=service1`, mentre per richiedere i *Flag IDs* per il team `team1` si può fare una richiesta *HTTP GET* all'indirizzo `http://10.10.0.1:8081/flagIds?team=team1`, infine per richiedere i *Flag IDs* per il servizio `service1` e per il team `team1` si può fare una richiesta *HTTP GET* all'indirizzo `http://10.10.0.1:8081/flagIds?team=team1&service=service1`. Ad ognuna di queste richieste il gamesystem risponderà con un oggetto *JSON* con il seguente formato:

```
{
    "service1": {
        "team1": [
        "flag_id_service_1_team_1",
        "flag_id_service_1_team_1",
        "flag_id_service_1_team_1"
        ],
        ...
    },
    ...
}
```

Acronimi e abbreviazioni

CLI Interfaccia testuale usata per interagire con software di vario genere. 21

CTF In informatica con il termine *Capture The Flag* (ing. cattura la bandiera) si indica una competizione di sicurezza informatica che consiste nello scovare vulnerabilità nascoste dagli organizzatori all'interno di sistemi informatici ben delimitati, e infine sfruttarle per ottenere informazioni segrete, chiamate in gergo *flag*. Il nome deriva dal gioco omonimo, in cui due squadre si sfidano per conquistare la bandiera dell'avversario. 1, 3, 8, 27, 31, 32

SLA Nell'ambito delle CTF *Attacco/Difesa* è il coefficiente che indica il rapporto tra il tempo in cui il servizio è stato raggiungibile e il tempo totale della competizione. È usato nel calcolo del punteggio di ogni servizio appartenente ad ogni squadra. 2, 22, 28, 29

SNAT Viene utilizzato per impedire ai partecipanti di distinguere il traffico generato dagli altri partecipanti da quello generato dal gamesystem sulla base dell'indirizzo IP sorgente. 18

VPN Una VPN è una tecnologia che permette di creare una rete privata virtuale su una rete pubblica come Internet. In questo modo è possibile collegare due o più computer in una rete virtuale come se fossero collegati direttamente tra loro. Nelle CTF *Attacco/Difesa* è usata per simulare una rete locale con tutti le squadre e le rispettive vulnbox. 11, 12, 27, 28, 32

Glossario

checker È il software che si occupa di interagire con i servizi durante una CTF *Attacco/Difesa*, ad ogni round verifica il corretto funzionamento dei servizi appartenenti ai team e aggiunge ad ognuno di essi una flag. 2, 19–23

gamesystem È una componente fondamentale delle CTF *Attacco/Difesa*, si occupa di gestire la competizione, eseguendo i checker, verificando le flag inviate dai team e aggiornando la classifica. 2, 9, 10, 13, 15, 16, 18, 19, 21, 23, 25, 27–32

NOP Team È una squadra parte di una competizione *Attacco/Difesa* che non ha giocatori, ma solo una vulnbox gestita dagli organizzatori la quale resta attiva per tutta la durata della competizione senza ricevere alcuna patch alle vulnerabilità. Viene solitamente usata per testare i propri attacchi prima di lanciarli contro le altre squadre. Le *flag* rubate al NOP Team non fanno ottenere punti. 16, 21, 27

TTL Reset Impostazione ad un valore fisso del campo *Time To Live* di un pacchetto IP. Viene usato per impedire di fare *fingerprinting* per distinguere il traffico generato dagli attacchi da quello generato dal gamesystem. 18

vulnbox È il server assegnato ad ogni squadra su cui sono in esecuzione i servizi vulnerabili. Le squadre hanno accesso come *root* al proprio server e possono installare cose o modificare configurazioni a loro piacimento. La vulnbox viene fornita dagli organizzatori già configurata per essere collegata alla rete di gioco tramite VPN, i servizi vulnerabili sono inoltre già installati. 2, 13, 15–19, 27, 28, 31, 32

Bibliografia

Riferimenti bibliografici

[29] Giovanni Vigna et al. *Ten Years of iCTF: The Good, The Bad, and The Ugly.* San Diego, CA: USENIX Association, ago. 2014. URL: https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna.

Siti web consultati

[1] *Amazon Web Services*. URL: https://aws.amazon.com/ (cit. alle pp. 8, 9).

[2] *Ansible*. URL: https://www.ansible.com/ (cit. alle pp. 8, 15, 17, 18, 22, 23, 26).

[3] *Attack/Defense for Beginners*. URL: https://2022.faustctf.net/information/attackdefense-for-beginners/.

[4] *C4T BuT S4D*. URL: https://cbsctf.ru/ (cit. a p. 10).

[5] *Celery*. URL: https://docs.celeryq.dev/ (cit. a p. 21).

[6] *CyberChallenge.IT*. URL: https://cyberchallenge.it/ (cit. a p. 3).

[7] *Django*. URL: https://www.djangoproject.com/ (cit. a p. 9).

[8] *Docker*. URL: https://www.docker.com/ (cit. alle pp. 10, 21, 23).

[9] *Docker Compose*. URL: https://docs.docker.com/compose/ (cit. alle pp. 10, 21, 23).

[10] *EnoEngine*. URL: https://github.com/enowars/EnoEngine (cit. a p. 9).

[11] *EnoFlag*. URL: https://enoflag.de/ (cit. a p. 9).

[12] *FAUST*. URL: https://faust.cs.fau.de/ (cit. a p. 9).

[13] *FAUST ctf-gameserver*. URL: https://github.com/fausecteam/ctf-gameserver (cit. a p. 9).

[14] *Flask*. URL: https://flask.palletsprojects.com/ (cit. alle pp. 8, 10, 21, 22).

[15] *ForcAD*. URL: https://github.com/pomo-mondreganto/ForcAD (cit. alle pp. 10, 11, 15–17, 19–22, 25).

[16] *Grafana*. URL: https://grafana.com/ (cit. alle pp. 16, 22, 23).

[17] *HackerDom*. URL: https://hackerdom.ru/ (cit. a p. 9).

[18] *Hackerdom Checksystem*. URL: https://github.com/HackerDom/checksystem (cit. a p. 9).

[19] *Hetzner Cloud*. URL: https://www.hetzner.com/cloud (cit. a p. 15).

[20] *iCTF Framework*. URL: https://github.com/shellphish/ictf-framework (cit. a p. 8).

[21] *NGINX*. URL: https://www.nginx.com/ (cit. a p. 20).

[22] *PostgreSQL*. URL: https://www.postgresql.org/ (cit. alle pp. 9, 15, 16, 20, 23).

[23] *Prometheus*. URL: https://prometheus.io/ (cit. alle pp. 16, 22, 23).

[24] *RabbitMQ*. URL: https://www.rabbitmq.com/ (cit. alle pp. 20, 21).

[25] *Redis*. URL: https://redis.io/ (cit. alle pp. 15, 16, 20, 23, 26).

[26] *Shellphish*. URL: https://shellphish.net/ (cit. a p. 8).

[27] *Terraform*. URL: https://www.terraform.io/ (cit. alle pp. 8, 15).

[28] *Vagrant*. URL: https://www.vagrantup.com/ (cit. a p. 8).

[30] *wggen*. URL: https://github.com/pomo-mondreganto/wggen (cit. a p. 17).

[31] *WireGuard*. URL: https://www.wireguard.com/ (cit. alle pp. 11, 16–18, 23, 27).