Creazione di un’infrastruttura per le CTF d’Attacco e Difesa

Obiettivo

Realizzare un’infrastruttura per giocare localmente delle CTF AD

Obiettivo

Realizzare un’infrastruttura per giocare localmente delle CTF AD

Pros

- Scenario realistico
- Ottimo per allenamenti
- Tool testing

Cons

- Lunga e difficile da realizzare
- Skill Issues

Obiettivo

Realizzare un’infrastruttura per giocare localmente delle CTF AD

Pros

- Scenario realistico
- Ottimo per allenamenti
- Tool testing

Cons

- Lunga e difficile da realizzare
- Skill Issues

Come realizzare il tutto senza troppe complicazioni?

Obiettivo

Realizzare un’infrastruttura per giocare localmente delle CTF AD

Pros

- Scenario realistico
- Ottimo per allenamenti
- Tool testing

Cons

- Lunga e difficile da realizzare
- Skill Issues

Come realizzare il tutto senza troppe complicazioni?

Risposta: 

Obiettivo: step per la realizzazione

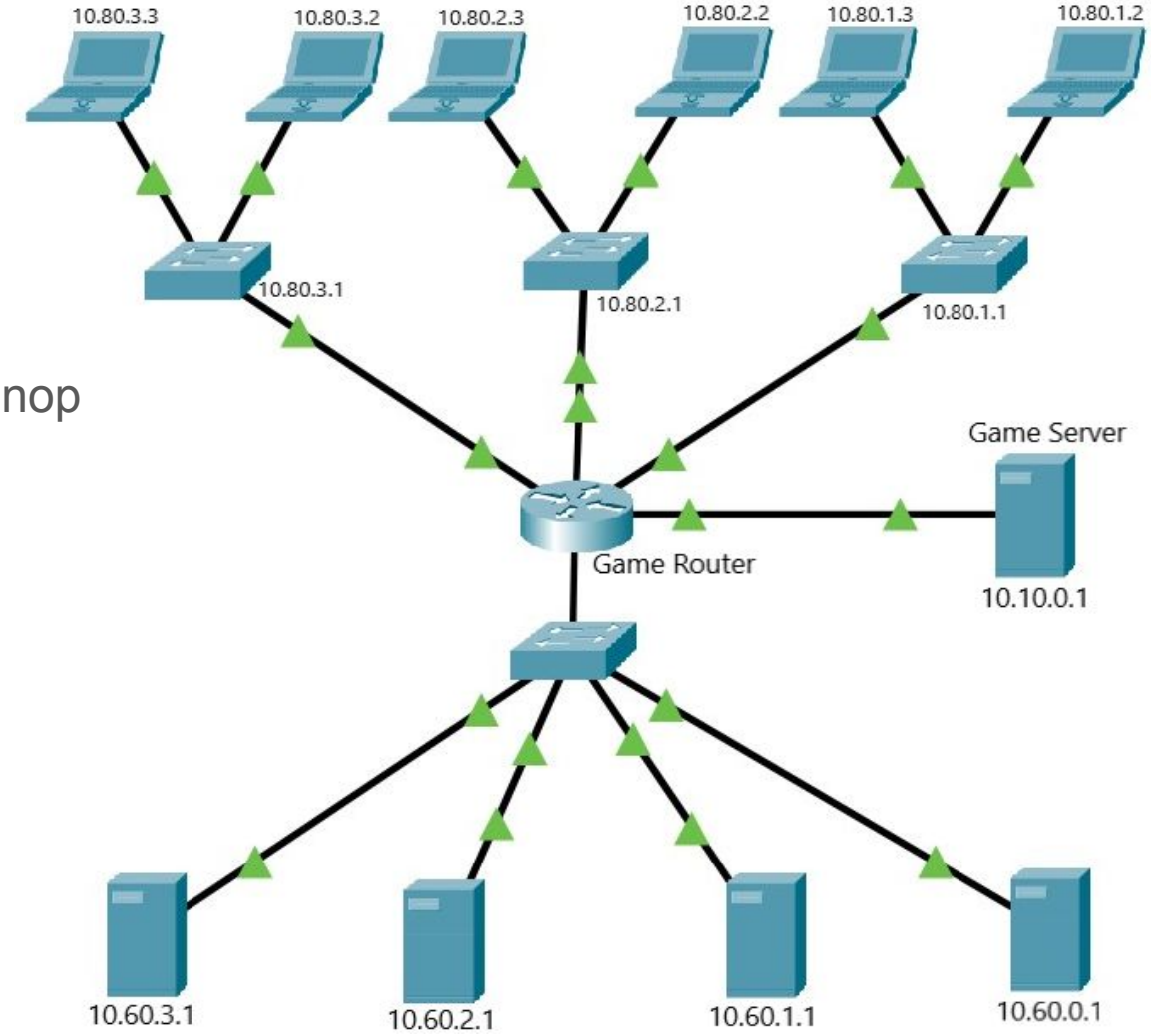
- Progettazione della rete
- Realizzazione di standard per servizi & checker
- Implementazione del game server
- Assemblamento degli step precedenti

Rete di Gara: Requisiti

- Subnet per teams: 10.80.{team_id}.0/24
- Macchine Vulnerabili: 10.60.{team_id}.1/24
- Game Server: 10.10.0.1
- Router per collegare il tutto

Rete di Gara: Progettazione

esempio con 3 team + nop



Rete di Gara: VPN

- istanza di Wireguard server per ogni team
- istanze isolate tra loro
- sfruttiamo l’istanza come router per team (come nella slide precedente)

Rete di Gara: Router

- Alpine per stare leggeri
- un pizzico di iptables
- NAT per garantire anonimità

Checkers

suddividiamo i checker in 3 azioni:

- check sla:

  controllo che il servizio sia raggiungibile e qualche operazione di base

- put flag:

  inserisco una flag nel flag store

- get flag:

  recupero una delle flag inserite all’interno del flag store

Checker: Realizzazione

Realizzandoli in python evitiamo varie complessità
in più passando in input dall'env le 3 possibili azioni rendiamo tutto più semplice:

- Check sla: `ACTION=CHECK_SLA TEAM_ID=0 ROUND=0 ./checker.py`
- Put flag: `ACTION=PUT_FLAG TEAM_ID=0 ROUND=0 FLAG=FLAG ./checker.py`
- Get flag: `ACTION=GET_FLAG TEAM_ID=0 ROUND=0 FLAG=FLAG ./checker.py`

per i risultati dei checker usiamo gli exit code per praticità:

- 101: OK (tutto funziona correttamente)
- 104: DOWN (errore nel servizio, seguito da un log su stderr)
- 110: ERROR (errore nel checker, seguito da un log su stderr)

Game Server: Progettazione

Componenti:

- checkers scheduler
- flag submission api
- flagIDs api
- web ui

Game Server: Progettazione

Componenti:

- checkers scheduler
- flag submission api
- flagIDs api
- web ui

Requisiti del linguaggio:

- Performante
- Facile da scrivere/manutenere

Game Server: Progettazione

Componenti:

- checkers scheduler
- flag submission api
- flagIDs api
- web ui

Requisiti del linguaggio:

- Performante
- Facile da scrivere/manutenere

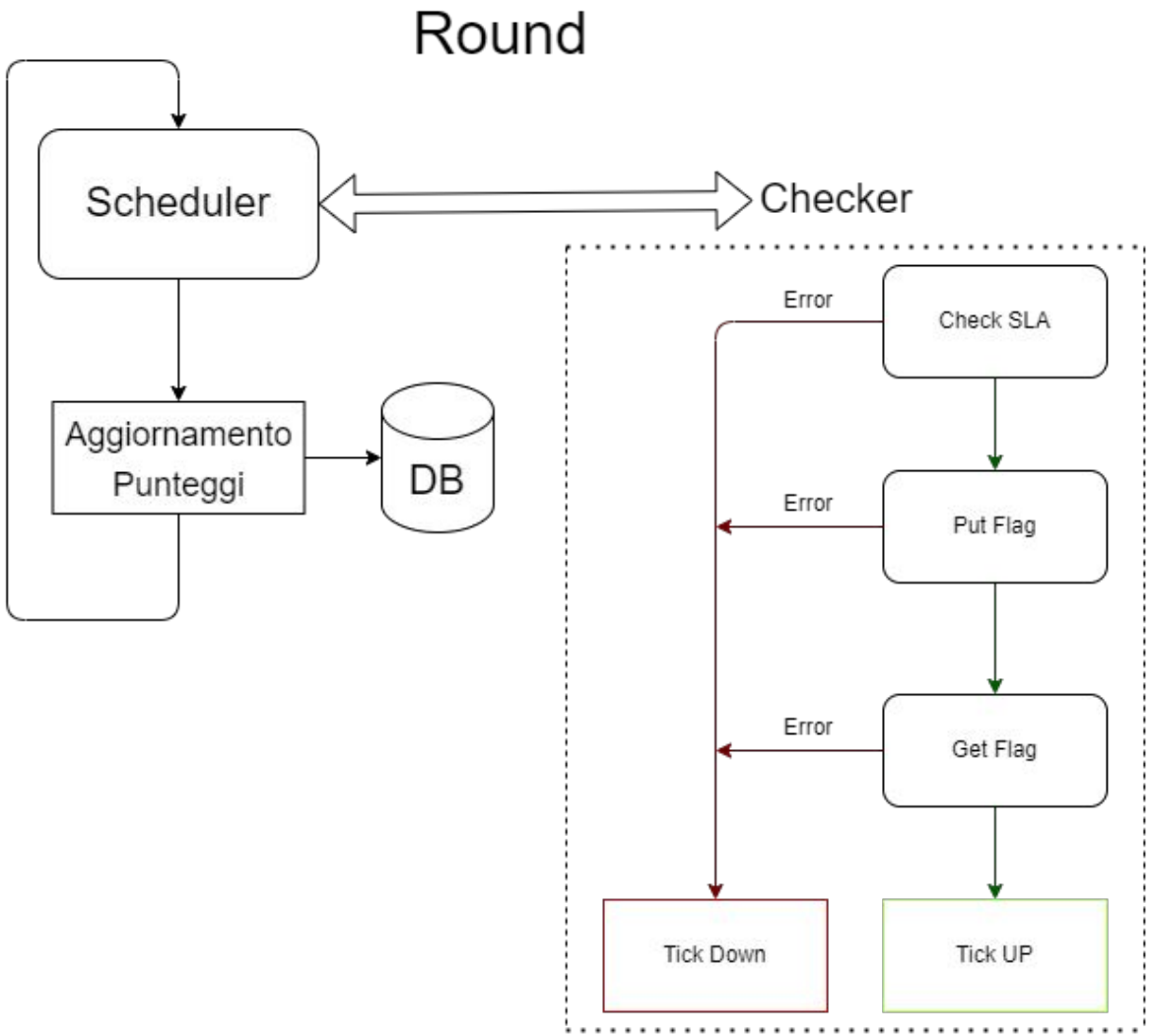
Risposta: 

Game Server: checkers scheduler

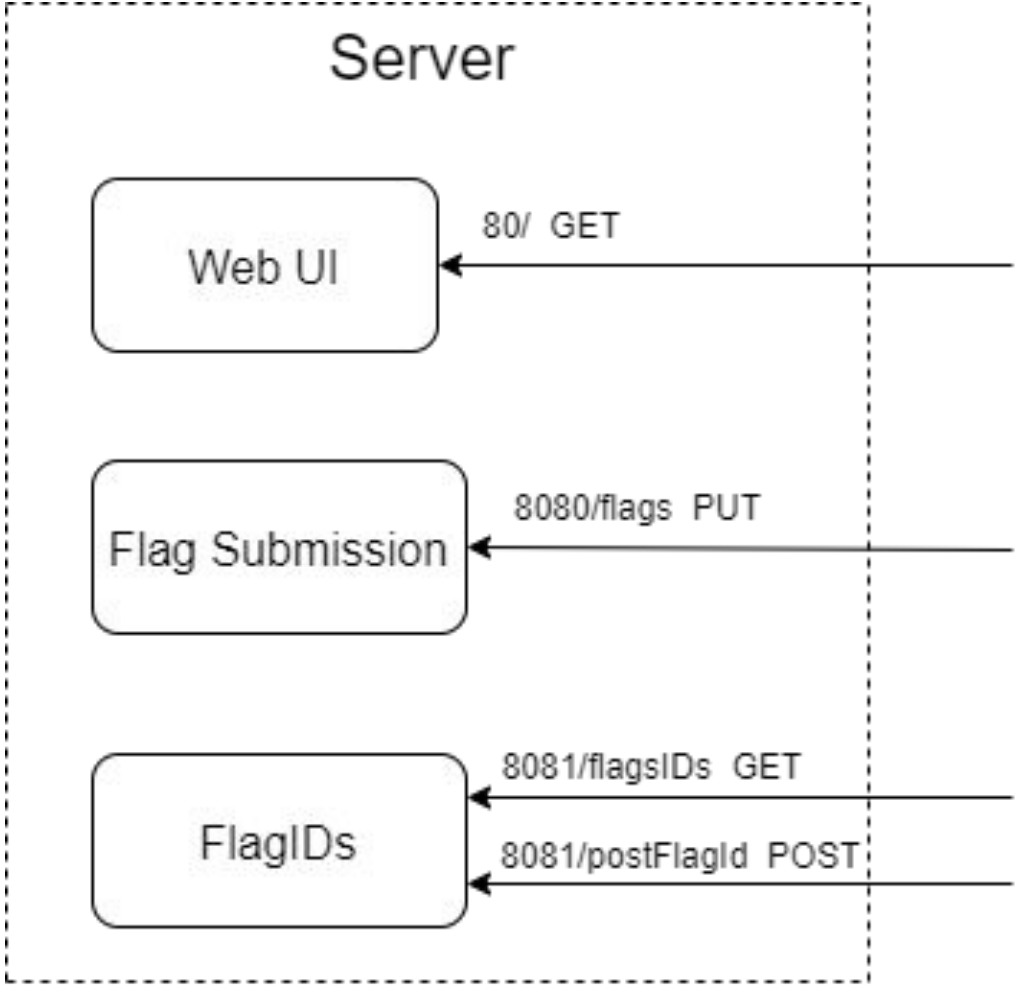
Requisiti:

- ogni round avvia i checker
- controlla le 3 azioni dei checker
- tempo randomizzato dallo scheduler
- aggiorna i punteggi/sla di conseguenza

Game Server: checkers scheduler



Game Server: API



Conclusione: Assemblamento Componenti

Arrivati a questo punto mettiamo tutto assieme

pov: io che assemblo

Conclusione

Già scritto e assemblato!

OASIS:

- Open
- Attack & Defense
- Simple
- Infrastructure
- System

github.com/TheRomanXpl0it/Oasis

Grazie per l’attenzione

Grazie Dario