Treniranje agenata za provođenje napada u simulatoru CCS korištenjem okruženja CyberBatleSim

Braut, Luka

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 659

TRENIRANJE AGENATA ZA PROVOĐENJE NAPADA U SIMULATORU CCS KORIŠTENJEM OKRUŽENJA CYBERBATTLESIM

Luka Braut

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 659

TRENIRANJE AGENATA ZA PROVOĐENJE NAPADA U SIMULATORU CCS KORIŠTENJEM OKRUŽENJA CYBERBATTLESIM

Luka Braut

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 659

Pristupnik:	**Luka Braut (0036528044)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Treniranje agenata za provođenje napada u simulatoru CCS korištenjem okruženja CyberBattleSim**

Opis zadatka:

Kako bi se potencijalne žrtve pripremile za obranu od kibernetičkih napada potrebno je imati emulaciju napadača. Emulaciju napadača trenutno obavljaju ljudi specijalizirani za tu svrhu, međutim, njih je malo i kvaliteta dosta varira. Idealno bi bilo kada bi se mogao automatizirati taj postupak jer bi se na taj način postigla ponovljivost i skalabilnost. U tu svrhu Microsoft je razvio okolinu CyberBattleSim koju je i javno objavio. Ta okolina služi za uvježbavanje i testiranje agenata. U ovom diplomskom radu potrebno je proučiti alat CyberBattleSim, njegove mogućnosti i ograničenja. Potom je potrebno napraviti alat koji će preslikati vježbenu okolinu opisanu u alatu Cyber Conflict Simulator (CCS) u CyberBattleSim. Potom je potrebno u CyberBattleSimu eksperimentirati s autonomnim agentima i njihovim ponašanjima te ih istrenirati u tako zadanoj okolini. Analizirati ograničenja tih agenata u odnosu na stvarna ponašanja napadača. Proučiti je li moguće tako naučene agente koristiti direktno u CCS-u.

Rok za predaju rada: 28. lipnja 2024.

Izražavam svoju zahvalnost mentoru doc. dr. sc. Stjepanu Grošu na vodstvu i doprinosu u izradi ovog diplomskog rada, kao i na cjelokupnom angažmanu tijekom mojega školovanja u području informacijske sigurnosti.

Velika hvala pripada i svim profesorima, asistentima i osoblju fakulteta koji su svojim znanjem, savjetima i otvorenošću doprinijeli mom akademskom razvoju. Također, zahvaljujem svim prijateljima i kolegama sa studija na zajedničkim trenucima učenja, podršci i dragocjenom druženju.

Na kraju, najiskrenije zahvale upućujem djevojci Ani i obitelji: mojoj majci, ocu i bratu, koji su svojim strpljenjem i potporom bili moj najčvršći oslonac tijekom ovih pet godina studiranja.

SADRŽAJ

1. Uvod

U današnjem digitalnom dobu kibernetički napadi postaju sve sofisticiraniji i češći, predstavljajući značajnu prijetnju za sigurnost informatičkih sustava diljem svijeta. Prema izvještaju *Cybersecurity Ventures*, predviđa se da će kibernetički kriminal uzrokovati globalne gubitke od 9.5 trilijuna dolara godišnje u 2024. godini, što čini kibernetičku sigurnost kritičnim pitanjem za organizacije svih veličina [15]. Kako bi se potencijalne žrtve pripremile za obranu od takvih napada, nužno je imati učinkovit sustav za emulaciju napadača. Trenutno, emulaciju napadača obavljaju specijalizirani stručnjaci, no njihov broj je ograničen, a kvaliteta varira. Razlog tome leži u složenosti potrebnog znanja i vještina, brzom razvoju novih tehnologija i metoda napada te visokim troškovima obuke i održavanja stručnjaka na visokoj razini kompetencije. Uz to, varijabilnost kvalitete proizlazi iz različitih pristupa i iskustava pojedinih stručnjaka, kao i iz stalne potrebe za prilagodbom i ažuriranjem znanja kako bi se učinkovito odgovorilo na nove prijetnje i taktike koje koriste napadači. Automatizacija ovog procesa omogućila bi ponovljivost i skalabilnost, čime bi se značajno unaprijedila obrana protiv kibernetičkih prijetnji.

Podržano učenje, grana strojnog učenja, pokazala se kao učinkovita metoda za treniranje autonomnih agenata koji mogu simulirati različite napade i strategije obrane u virtualnim okruženjima. Podržano učenje omogućuje agentima da uče optimalne strategije kroz interakciju s okolinom, čime se postiže visok stupanj prilagodljivosti i učinkovitosti [18]. Ovaj pristup omogućava stvaranje agenata koji mogu reagirati na različite scenarije napada, što ih čini prikladnim za primjenu u kibernetičkoj sigurnosti [16].

U tu svrhu, Microsoft je razvio *CyberBattleSim*, okruženje otvorenog koda koje služi za uvježbavanje i testiranje agenata u simuliranim kibernetičkim okruženjima. CyberBattleSim omogućava simulaciju napada i obrane unutar virtualne mreže čime pomaže u razvoju i testiranju strategija obrane protiv kibernetičkih prijetnji. CyberBattleSim omogućuje istraživanje ponašanja napadača i branitelja te evaluaciju različitih sigurnosnih mjera u kontroliranom okruženju [19].

Cilj ovog rada je istražiti mogućnosti CyberBattleSima u kontekstu treniranja agenata za provođenje napada te razviti alat za preslikavanje vježbenog okruženja opisanog u *Cyber Conflict Simulatoru* (CCS) u CyberBattleSim. Nakon implementacije, cilj je

eksperimentirati s autonomnim agentima u tako zadanoj okolini, analizirati njihova ponašanja te usporediti ograničenja tih agenata sa stvarnim napadačima. Kroz ovaj pristup, istraženo je koliko su agenti trenirani u CyberBattleSimu sposobni replicirati stvarne napade i koje su njihove prednosti i nedostaci u usporedbi s ljudskim napadačima.

Rad je strukturiran na sljedeći način: u drugom poglavlju pruža se pregled podržanog učenja, uključujući osnovne koncepte, algoritme i primjenu u kibernetičkoj sigurnosti. Treće poglavlje posvećeno je CyberBattleSimu gdje će se detaljnije razmotriti njegova arhitektura, komponente te postupak postavljanja i konfiguracije simulacija. Četvrto poglavlje opisuje razvoj alata za preslikavanje okruženja CCS u CyberBattleSim, uključujući ključne razlike između ta dva sustava i tehničke detalje implementacije. U petom poglavlju fokus je na treniranju autonomnih agenata u CyberBattleSimu, s naglaskom na odabir i konfiguraciju agenata, metodologiju treniranja te analizu njihovog ponašanja uz ograničenja i potencijalna poboljšanja. Konačno, šesto poglavlje donosi zaključak uz osvrt na ishod istraživanja.

2. Podržano učenje

Podržano učenje (engl. *reinforcement Learning*) jedan je od tri ključna aspekta strojnog učenja, pored nadziranog (engl. *supervised Learning*) i nenadziranog učenja (engl. *unsupervised Learning*). Omogućava agentima da uče optimalne akcije kroz kontinuiranu interakciju s okolinom kako bi maksimizirali kumulativnu nagradu. Ova metoda učenja se temelji na principu nagrade i kazne, što je vrlo slično načinu na koji ljudi uče iz svojih svakodnevnih iskustava i situacija. Podržano učenje se bitno razlikuje od nadziranog učenja, gdje se model trenira na temelju unaprijed označenih podataka i nenadziranog učenja, koje se bavi prepoznavanjem obrazaca i struktura u podacima bez eksplicitnih oznaka ili nadzora [11].

Primjeri uspješnih primjena podržanog učenja uključuju AlphaGo [7] i AlphaFold [1], dva revolucionarna sustava koje je razvio DeepMind. AlphaGo je postao poznat kada je pobijedio vrhunske igrače *Go* – a, igre poznate po svojoj složenosti i ogromnom broju mogućih poteza. Kombinacijom dubokih neuronskih mreža i podržanog učenja, AlphaGo je uspio razviti strategije koje su nadmašile ljudske sposobnosti. Ovaj uspjeh je bio prekretnica u području umjetne inteligencije, demonstrirajući snagu i potencijal podržanog učenja u rješavanju složenih problema.

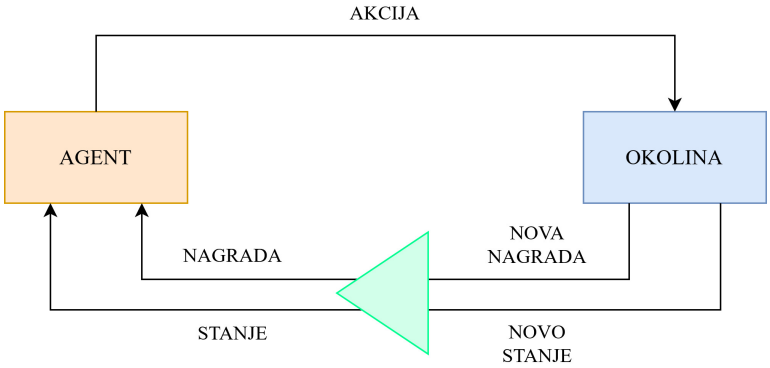
Nakon AlphaGo sustava, DeepMind je razvio AlphaFold, jedan od najsuvremenijih sustava (engl. *state of the art*) koji koristi podržano učenje, a za cilj ima predviđanje struktura proteina. Ovo je bio još jedan veliki iskorak, s obzirom na to da je predviđanje proteinskih struktura od ključne važnosti za biomedicinska istraživanja i razvoj novih lijekova. AlphaFold je pokazao sposobnost točnog predviđanja složenih proteinskih struktura, problem koji je bio otvoren desetljećima. Ovaj napredak nije samo demonstrirao moć održanog učenja u biološkim znanostima, već je i otvorio nove mogućnosti za istraživanje i inovacije u medicini.

2.1. Osnove podržanog učenja

Podržano učenje se temelji na interakciji između agenta i okoline. Agent je entitet koji donosi odluke, dok je okolina sve ono što agent može opažati i na što može utjecati svojim

akcijama. Ključni elementi podržanog učenja uključuju stanja, akcije i nagrade. Stanje predstavlja trenutnu situaciju u kojoj se agent nalazi, akcija je potez koji agent može poduzeti, a nagrada je povratna informacija koju agent prima iz okoline kao rezultat svoje akcije.

Na slici 2.1 prikazana je interakcija između agenta i okoline u podržanom učenju. Agent promatra trenutno stanje okoline i na temelju toga odlučuje koju akciju poduzeti. Nakon što poduzme akciju, okolina se mijenja u novo stanje i dodjeljuje agentu novu nagradu. Agent koristi ovu nagradu kao povratnu informaciju kako bi prilagodio svoje buduće odluke s ciljem maksimiziranja ukupne nagrade tijekom vremena.

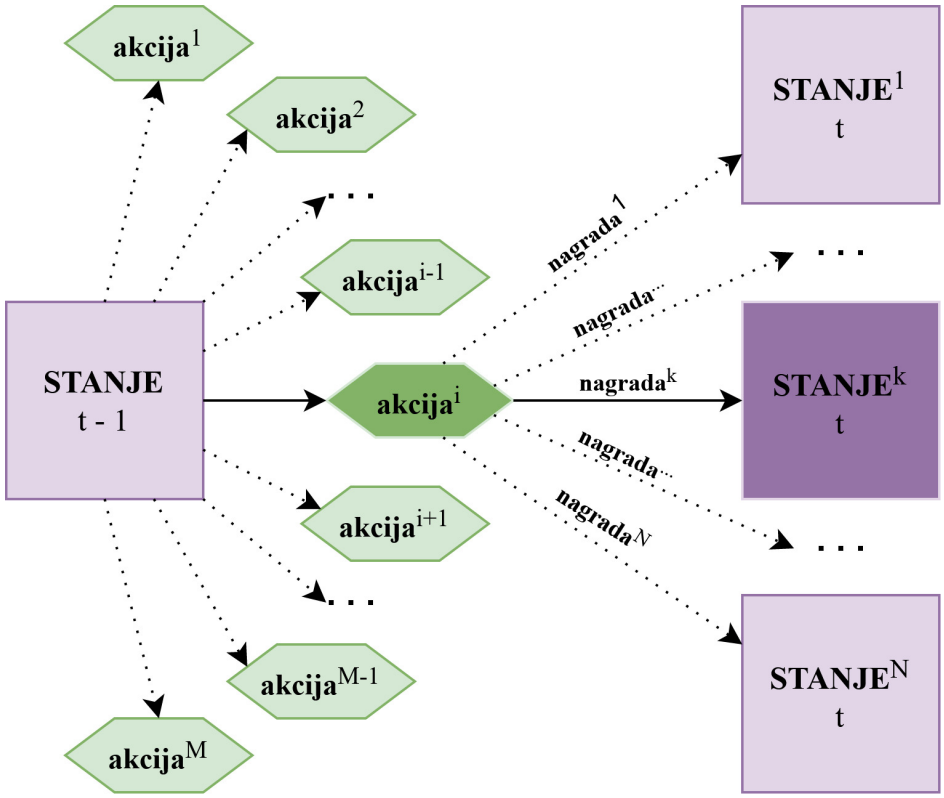


Slika 2.1: Prikaz interakcije između agenta i okoline

Na slici 2.1 može se vidjeti kako agent na temelju opažanja stanja iz okoline poduzima akciju koja rezultira promjenom stanja okoline i povratnom informacijom u obliku nagrade. Ova povratna informacija pomaže agentu da procijeni koliko je njegova akcija bila korisna u postizanju cilja. Agent može obavljati akcije beskonačno dugo, što rezultira problemom koji nazivamo kontinuirani problem. Alternativno, agent može dovršiti svoje zadatke u konačnom broju koraka, kao što je slučaj u CyberBattleSimu, gdje takav problem nazivamo epizodički.

Okolina u kojoj agent djeluje može biti deterministička ili stohastička. U determinističkoj okolini, kada agent poduzme neku akciju, rezultat te akcije uvijek će biti isti, što znači da će prelaziti u isto stanje svaki put kada se ta akcija ponovi u istom početnom stanju. S druge strane, u stohastičkoj okolini, rezultat akcije može varirati zbog prisutnosti nasumičnih elemenata, što znači da isti potez u istom stanju može dovesti do različitih ishoda svaki put kada se ponovi [3].

Slika 2.2 prikazuje dijagram procesa donošenja odluka u podržanom učenju. Iz svakog trenutnog stanja agent može birati između različitih akcija, koje rezultiraju različitim novim stanjima i nagradama. Dijagram ilustrira složenost procesa odlučivanja u podržanom učenju, gdje agent mora uzeti u obzir mnoge moguće ishode kako bi donio optimalne odluke.



Slika 2.2: Dijagram procesa donošenja odluke

Model podržanog učenja može se formalno opisati pomoću Markovljevog procesa odlučivanja (engl. *Markov Decision Process*, MDP). U MDP modelu podržanog učenja, okolina se opisuje kao skup stanja s i akcija a , s prijelaznim vjerojatnostima P i funkcijom nagrade R. Prijelazne vjerojatnosti definiraju vjerojatnost prelaska iz jednog stanja u drugo kao rezultat određene akcije, dok funkcija nagrade specificira trenutnu nagradu koju agent dobiva za poduzetu akciju u danom stanju. Modelu se pridodaje i faktor diskontiranja γ koji balansira između trenutne i buduće nagrade.

Cilj podržanog učenja je pronaći optimalnu politiku π koja maksimizira očekivanu kumulativnu nagradu. Politika π je funkcija koja preslikava stanja u akcije $\pi : S \rightarrow A$. Matematički, ukupna očekivana nagrada može se definirati kao [18]:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Funkcija vrijednosti stanja $V(s)$ pod politikom π definirana je kao očekivana ukupna nagrada počevši od stanja s i slijedeći politiku π:

$$V^{\pi}(s) = \mathbb{E}^{\pi}[G_t \mid S_t = s]$$

Optimalna funkcija vrijednosti stanja $V^*(s)$ zadovoljava Bellmanovu jednadžbu:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V^*(s')]$$

Ove jednadžbe pružaju temelj za izračunavanje optimalne politike i optimalne funkcije vrijednosti, koristeći iterativne metode za ažuriranje procjena vrijednosti stanja i akcija [10].

2.2. Algoritmi podržanog učenja

Cilj podržanog učenja je pronaći politiku (engl. *policy*) kojoj je funkcija preslikavanje stanja u akcije, maksimizirajući kumulativnu nagradu koju agent dobiva tijekom vremena. Optimalna politika je ona koja osigurava najveći mogući povrat za svako stanje. Kumulativna nagrada se često diskontira kako bi se uzela u obzir nesigurnost i vremenska vrijednost nagrade, što se postiže uvođenjem diskontnog faktora γ. Najpoznatiji algoritmi podržanog učenja uključuju Q-učenje, SARSA, duboko Q-učenje (engl. *Deep Q-Learning*, DQL) i naprednije metode poput *Proximal Policy Optimization* (PPO) i *Actor-Critic* metoda.

2.2.1. Q-učenje

Q-učenje (engl. *Q-learning*) je jedan od osnovnih algoritama podržanog učenja koji agentima omogućava da uče optimalne politike bez potrebe za modelom okoline. Ovaj algoritam koristi tablicu Q-vrijednosti za pohranu procijenjenih vrijednosti parova stanja i akcija. Q-vrijednost $Q(s, a)$ predstavlja očekivanu kumulativnu nagradu koju agent može dobiti ako poduzme akciju a u stanju s i potom slijedi optimalnu politiku.

Algoritam iterativno ažurira Q-vrijednosti koristeći Bellmanovu jednadžbu ažuriranja:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

gdje je α stopa učenja, γ diskontni faktor, r nagrada dobivena nakon poduzimanja akcije a u stanju s, s' novo stanje nakon akcije, a a' najbolja akcija u novom stanju [18].

Ovaj pristup omogućava agentu da postepeno poboljšava svoje procjene vrijednosti akcija u različitim stanjima, približavajući se optimalnoj politici kroz iteracije. Q-učenje je *off-policy* algoritam, što znači da uči optimalnu politiku neovisno o politici koja se koristi za istraživanje.

2.2.2. SARSA

SARSA (*State-Action-Reward-State-Action*) je *on-policy* algoritam podržanog učenja koji također koristi Q-vrijednosti za učenje optimalnih akcija. Razlika između SARSA i

Q-učenja je u načinu ažuriranja Q-vrijednosti. U SARSA algoritmu, Q-vrijednosti se ažuriraju na temelju akcija odabranih trenutnom politikom:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

gdje su s i a trenutno stanje i akcija, r nagrada, s' novo stanje, a a' akcija odabrana u novom stanju prema trenutnoj politici [23].

SARSA je prikladniji za situacije gdje je važno slijediti određenu strategiju tijekom učenja, dok je Q-učenje prikladnije za istraživanje optimalnih politika neovisno o strategiji istraživanja. Ovaj algoritam nije dostupan u trenutnoj implementaciji CyberBattleSima.

2.2.3. Duboko Q-učenje

Duboko Q-učenje (engl. *Deep Q-Learning*, DQL) je proširenje Q-učenja koje koristi duboke neuronske mreže za aproksimaciju Q-vrijednosti. Ovaj pristup omogućava rješavanje problema s velikim i kontinuiranim prostorima stanja, gdje bi tablična reprezentacija Q-vrijednosti bila nepraktična.

U DQL algoritmu, mreža Q aproksimira funkciju Q-vrijednosti $Q(s,a;\theta)$, gdje su θ parametri mreže. Algoritam koristi iskustvo ponovnog reproduciranja (engl. *experience replay*) i fiksne ciljne mreže (engl. *fixed target networks*) kako bi poboljšao stabilnost i učinkovitost učenja. Ciljna vrijednost za ažuriranje parametara mreže je:

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-)$$

gdje su θ^- parametri ciljane mreže koji se povremeno ažuriraju kopiranjem parametara iz glavne mreže θ [14].

DQL je revolucionirao područje podržanog učenja, omogućujući rješavanje složenih zadataka poput igranja videoigara na razini koja je usporediva s ljudskom.

2.3. Primjena podržanog učenja u kibernetičkoj sigurnosti

Podržano učenje se pokazalo izuzetno korisnim u području kibernetičke sigurnosti, omogućavajući automatizaciju složenih zadataka poput otkrivanja prijetnji, penetracijskog testiranja i odgovora na napade. Njegova sposobnost učenja iz interakcije s okolinom čini ga idealnim za dinamične i nepredvidljive scenarije koji su često prisutni u kibernetičkom okruženju.

Sustavi za otkrivanje napada (engl. *Intrusion Detection Systems*, IDS) koriste podržano učenje za identifikaciju i odgovaranje na neovlaštene pristupe i napade na računalne mreže. IDS može biti zasnovan na potpisima (engl. *signature*) ili anomalijama, pri čemu podržano

učenje igra ključnu ulogu u analizi obrazaca ponašanja kako bi identificirao anomalije koje ukazuju na potencijalne napade.

Penetracijsko ispitivanje (engl. *penetration testing*, PT) je proces testiranja sigurnosnih ranjivosti sustava. Tradicionalni PT zahtijeva ljudske stručnjake koji ručno provode testove, što može biti skupo i vremenski zahtjevno. Ghanem et al. [6] predložili su za tu problematiku inteligentni automatizirani PT sustav koji koristi RL za poboljšanje performansi i točnosti testiranja u srednjim i velikim mrežama.

Duboko podržano učenje (engl. *Deep Reinforcement Learning*) kombinira podržano učenje s dubokim učenjem za rješavanje složenih problema s velikim prostornim stanjem. Nguyen et al. [16] pokazali su kako takav pristup može biti učinkovit alat za detekciju i prevenciju napada u stvarnom vremenu. Duboko podržano učenje koristi neuronske mreže za aproksimaciju Q-vrijednosti, što omogućava rješavanje problema s kontinuiranim i velikim prostorima stanja. Ovaj pristup je primijenjen na različite scenarije kibernetičke sigurnosti, uključujući zaštitu od raspodijeljenog napada uskraćivanja usluge (engl. *distributed denial of service*, DDoS) i otkrivanje zlonamjernog softvera.

U ovom radu, korišteno je simulacijsko okruženje CyberBattleSim [12] koje omogućava primjenu podržanog učenja za emulaciju napadača. Ovo je još jedna od primjena podržanog učenja u kibernetičkoj sigurnosti.

CyberBattleSim je eksperimentalni projekt koji istražuje kako se autonomni agenti ponašaju u simuliranom poslovnom okruženju mreže. Stoga, on omogućuje simulaciju treniranja agenta napadača i agenta koji se brani u simuliranom okruženju. Njihove akcije uključuju mrežne i računalne naredbe, s ciljem napadača da osvoji udio mreže i ukrade povjerljive informacije, dok je cilj branitelja izbaciti napadače ili ublažiti njihove radnje. Međutim, u ovom radu korišten je isključivo dio funkcionalnosti koji trenira i koristi agenta napadača.

3. CyberBattleSim

U kontekstu rješavanja kompleksnih problema u kibernetičkoj sigurnosti, razvoj naprednih alata za simulaciju i treniranje agenata za emulaciju napadača predstavlja ključni korak. Doprinos u ovom području dao je i razvoj CyberBattleSima, alata koji omogućava istraživačima i stručnjacima za sigurnost da treniraju i testiraju autonomne agente u simuliranim kibernetičkim okruženjima. Ovaj alat otvorenog koda kojeg je razvio Microsoft, nudi platformu za simulaciju različitih scenarija napada i obrane, pružajući dragocjene uvide u ponašanje agenata i učinkovitost sigurnosnih mjera [12].

3.1. Uvod u CyberBattleSim

CyberBattleSim je izgrađen na temelju *OpenAI Gym* sučelja u programskom jeziku Python, što omogućava treniranje agenata s alatima za strojno, odnosno podržano učenje. *OpenAI Gym* je 2021. godine prešao u projekt *Gymnasium* [20], gdje se održava do danas. Platforma simulira računalne mreže koristeći grafove, gdje čvorovi predstavljaju različite računalne sustave, a bridovi predstavljaju mrežne veze među njima. Ovo omogućava stvaranje prilagodljivih i složenih scenarija za treniranje agenata.

OpenAI Gym je sučelje za podržano učenje koje omogućava stvaranje i testiranje agenata u različitim simuliranim okruženjima. Njegova glavna prednost leži u jednostavnosti korištenja i fleksibilnosti, što omogućuje brzo postavljanje eksperimenata i integraciju s različitim algoritmima podržanog učenja. Ova platforma omogućuje modeliranje apstrakcija visoke razine i lako prilagođavanje istraživačkim potrebama, što je ključno za razvoj učinkovitih agenata [2]. CyberBattleSim je, apstraktno gledano, vrsta adaptera koji adaptira mrežnu topologiju, agente kao napadače i branitelje te ostala svojstva kibernetičke sigurnosti u okolinu razumljivu *OpanAi Gymu*.

Razlog implementacije CyberBattleSima kroz simulacijsko okruženje, u odnosu na emulacijsko okruženje, leži u prednostima poput jednostavnosti, brzine, apstraktnosti i veće kontrole. Ove karakteristike čine simulacijska okruženja pogodnijima za eksperimentiranje s podržanim učenjem, omogućujući istraživačima da brzo iteriraju i testiraju različite strategije bez potrebe za velikim resursima koje zahtijevaju emulacijska rješenja.

Apstrakcija smanjuje razinu vjernosti podataka, no omogućava modeliranje aspekata sustava koji su relevantni, kao što je mrežna komunikacija na razini aplikacija umjesto simulacije paketa te zanemarivanje detalja niske razine poput datotečnog sustava ili registra ako nisu nužni [19]. Ključno je što apstrakcijom CyberBattleSima nije vjerno pokazano kako stvarni napadač može izvesti napad, čime se nakon otvaranja koda javnosti moglo educirati napadače kako provesti napad koji do tada nisu kao pojedinci poznavali.

3.2. Arhitektura i komponente CyberBattleSima

Simulacija se odvija na definiranoj mrežnoj topologiji i skupu unaprijed definiranih ranjivosti. Napadač se kreće unutar mreže putem lateralnih pokreta, koristeći definirane ranjivosti. Okruženje stvara simulaciju mreže čvorova povezanih različitim mrežnim protokolima. Svaki čvor u mreži predstavlja jedan računalni sustav, dok veze između čvorova predstavljaju mrežne protokole potrebne za komunikaciju. Primjer takve mreže može uključivati čvorove poput početno zaraženog čvora (klijenta), web stranice i direktorija web stranice (na primjer, *git* repozitorij web stranice), koji su povezani odgovarajućim protokolima. Svaki čvor može imati različite vrste ranjivosti koje napadač može iskoristiti za lateralno kretanje kroz mrežu. Napadač u CyberBattleSimu ima zadatak preuzeti kontrolu nad dijelom mreže iskorištavanjem postojećih ranjivosti. Inicijalno je zamišljen za jednoga agenta s okolinom nakon inicijalnog proboja (engl. *post breach*) napadača u mrežu, no upotrebljiv je i u slučaju kada ima više napadača (engl. *multiagent*), što su pokazali Kunz et al. [9]. Pored trenirajućeg napadača, CyberBattleSim također uključuje branitelja koji ne može biti treniran. Branitelj djeluje prema statičnom probabilističkom modelu. Jedina dostupna akcija branitelja je nasumično odabrati čvor za ponovno postavljanje (engl. *reimaging*), a uspješnost ove akcije određena je konfiguriranom vjerojatnošću. Branitelj je također ograničen kako bi se osigurala određena razina dostupnosti mreže u svakom trenutku. Stoga, branitelj ne može zaključati napadača ponovnim slikanjem svih čvorova odjednom. Četiri osnovne komponente CyberBattleSima su okolina (engl. *environment*), prostor akcija (engl. *action space*), prostor promatranja (engl. *observation space*) i nagrada (engl. *reward*). No kako bi se bolje razumjelo rad komponenti prvo je iznesen model koji leži u pozadini CyberBattleSima.

3.2.1. Model

Model CyberBattleSima definira strukturu i ponašanje mrežnog okruženja u kojem se simulacije odvijaju. U ovom dijelu detaljno ćemo razmotriti ključne komponente modela, počevši od klase NodeInfo, koja predstavlja osnovni građevni blok mreže, a zatim ćemo se

spuštati kroz ostale klase koje se koriste unutar NodeInfo. U daljnjoj razradi rada bit će opisano generiranje okoline koje se u suštini svodi na definiranje varijable čvorova mreže nodes koja je rječnik (engl. *dictionary*) s imenima čvorova kao ključem (engl. *key*) i objektima tipa NodeInfo kao vrijednosti.

Klasa NodeInfo predstavlja čvor u mreži i sadrži informacije o uslugama, ranjivostima, vrijednosti čvora, svojstvima, konfiguraciji vatrozida i statusu čvora, kao što je prikazano u kodu 3.1.

```
class NodeInfo:
  services: List[ListeningService]
  vulnerabilities: VulnerabilityLibrary =
     dataclasses.field(default_factory=dict)
  value: NodeValue = 0
  properties: List[PropertyName] =
     dataclasses.field(default_factory=list)
  firewall: FirewallConfiguration = FirewallConfiguration()
  agent_installed: bool = False
  privilege_level: PrivilegeLevel = PrivilegeLevel.NoAccess
  reimagable: bool = True
  last_reimaging: Optional[datetime] = None
  owned_string: str = ""
  status = MachineStatus.Running
  sla_weight: float = 1.0
```

Kod 3.1: Definicija klase `NodeInfo`

`services` predstavlja popis svih usluga koje su prisutne na čvoru, koje mogu uključivati različite mrežne servise dostupne i potencijalno ranjive na napade. `vulnerabilities` sadrži sve poznate ranjivosti čvora. Na ovo svojstvo je potrebno posebno obratiti pažnju prilikom definiranja zbog ključnosti za simulaciju sigurnosnih prijetnji jer omogućuju napadaču da iskorištava slabe točke u sustavu.

Varijabla `value` označava unutarnju vrijednost čvora, što može odražavati važnost čvora unutar mreže, poput vrijednosti podataka koje čvor sadrži ili njegove uloge u mrežnoj infrastrukturi. `properties` sadrži popis svojstava čvora, uključujući različite karakteristike čvora važne za njegovu funkcionalnost i sigurnost (na primjer naziv operacijskog sustava). `firewall` definira konfiguraciju vatrozida na čvoru koji kontrolira dolazni i odlazni mrežni promet te može blokirati sumnjive aktivnosti ili neovlaštene pristupe.

Varijabla `agent_installed` označava je li napadački agent instaliran na čvoru, omogućujući napadaču daljnje operacije na čvoru, uključujući iskorištavanje ranjivosti i

kretanje kroz mrežu. `privilege_level` označava razinu pristupa koju napadač ima na čvoru, varirajući od potpunog nedostatka pristupa do administratorskih i sistemskih privilegija. `reimagable` označava može li se čvor ponovno postaviti (engl. *reimagable*), što znači vraćanje čvora u početno stanje, ključno za obranu mreže jer omogućuje brzo uklanjanje napadačkih agenata i vraćanje čvora u sigurno stanje prije kompromitiranja.

Varijabla `last_reimaging` bilježi vrijeme posljednjeg ponovnog postavljanja čvora, važno za praćenje učestalosti zaštitnih mjera i procjenu rizika od ponovnog kompromitiranja. `owned_string` je tekst koji označava status vlasništva nad čvorom i može se koristiti za bilježenje informacija o napadačevom vlasništvu ili kontroli nad čvorom. `status` označava trenutni radni status čvora, poput radnog statusa (engl. *running*), zaustavljenog statusa (engl. *stopped*) ili u procesu ponovnog slikanja (engl. *reimaging*), ključno za razumijevanje funkcionalnosti čvora u svakom trenutku. Konačno, `sla_weight` označava težinu usluge u slučaju ne funkcioniranja, što može biti važno za mjerenje utjecaja pada čvora na ukupnu mrežnu uslugu.

Klasa `NodeInfo` koristi nekoliko drugih klasa za definiranje različitih aspekata čvora, uključujući `ListeningService`, `FirewallConfiguration`, `PrivilegeLevel`, `VulnerabilityLibrary` i `MachineStatus`.

Klasa `ListeningService`, prikazana u kodu 3.2, predstavlja mrežnu uslugu koja je dostupna na određenom portu čvora. Ova klasa uključuje informacije o imenu porta, dopuštenim vjerodajnicama za autentifikaciju, statusu usluge i težini usluge u slučaju ne funkcioniranja.

```
class ListeningService:
  name: PortName
  allowedCredentials: List[CredentialID] =
     dataclasses.field(default_factory=list)
  running: bool = True
  sla_weight = 1.0
```

Kod 3.2: Definicija klase `ListeningService`

`name` predstavlja ime porta, a `allowedCredentials` su vjerodajnice koje su dopuštene za autentifikaciju na čvoru tom uslugom, `running` pokazuje je li usluga aktivna, dok `sla_weight` označava težinu usluge u slučaju ne funkcioniranja.

Klasa `FirewallConfiguration`, prikazana u kodu 3.3, definira postavke vatrozida za određeni čvor, uključujući pravila za dolazni i odlazni promet.

```
class FirewallConfiguration:
  outgoing: List[FirewallRule] = field(repr=True,
     default_factory=lambda: [
  FirewallRule("RDP", RulePermission.ALLOW),
  FirewallRule("SSH", RulePermission.ALLOW),
  FirewallRule("HTTPS", RulePermission.ALLOW),
  FirewallRule("HTTP", RulePermission.ALLOW)])

  incoming: List[FirewallRule] = field(repr=True,
     default_factory=lambda: [
  FirewallRule("RDP", RulePermission.ALLOW),
  FirewallRule("SSH", RulePermission.ALLOW),
  FirewallRule("HTTPS", RulePermission.ALLOW),
  FirewallRule("HTTP", RulePermission.ALLOW)])
```

Kod 3.3: Definicija klase `FirewallConfiguration`

U klasi `FirewallConfiguration`, `outgoing` i `incoming` su popisi pravila vatrozida za odlazni i dolazni promet. Svako pravilo definirano je klasom `FirewallRule`, koja specificira port i dopuštenje za taj port.

Klasa `FirewallRule`, prikazana u kodu 3.4, predstavlja pojedinačno pravilo vatrozida koje definira je li promet na određenom portu dopušten ili blokiran.

```
class FirewallRule:
  port: PortName
  permission: RulePermission
  reason: str = ""

class RulePermission(Enum):
  ALLOW = 0
  BLOCK = 1
```

Kod 3.4: Definicija klase `FirewallRule`

U klasi `FirewallRule`, varijabla `port` predstavlja ime porta na kojem se pravilo primjenjuje. To može biti bilo koji mrežni port s imenom koje predstavlja protokol koji se preko njega odvija. Tako na primjer za porta 80 ime porta u CyberBattleSimu biti će `HTTP`, odnosno za port 443 `HTTPS`.

Varijabla `permission` koristi enumeraciju `RulePermission` koja može imati dvije vrijednosti: `ALLOW` ili `BLOCK`. Ove vrijednosti određuju je li promet na definiranom portu dopušten (engl. *allow*) ili blokiran (engl. *block*). Na primjer, ako je postavljeno da je promet

s imenom porta `HTTP` blokiran, sve HTTP (port 80) zahtjeve će vatrozid odbaciti, što može spriječiti neovlaštene pristupe i napade.

Varijabla `reason` je opcionalni tekstualni opis koji može sadržavati razlog za postavljanje pravila. Ovo je korisno za administratore mreže u stvarnom okruženju jer im omogućuje da dokumentiraju svrhu svakog pravila vatrozida, čime se olakšava održavanje i upravljanje mrežnom sigurnošću.

Također, vidljivo je u klasi `FirewallConfiguration` da u slučaju ne definiranja pravila vatrozida ona inicijalno dopuštaju protokole: `RDP`, `SSH`, `HTTPS` i `HTTP`, dok sve ostale blokiraju.

Klasa `PrivilegeLevel`, prikazana u kodu 3.5, predstavlja razine privilegija na određenom čvoru, od bez pristupa do administratorskih i sistemskih privilegija.

```
class PrivilegeLevel(IntEnum):
  NoAccess = 0
  LocalUser = 1
  Admin = 2
  System = 3
  MAXIMUM = 3
```

Kod 3.5: Definicija klase `PrivilegeLevel`

Ova klasa omogućava definiranje i praćenje razine kontrole koju napadač ima nad čvorom.

U klasi `PrivilegeLevel`, varijabla `NoAccess` označava da napadač nema nikakav pristup čvoru. Ovo je početna razina koja implicira da napadač nije uspio prodrijeti u sustav ili dobiti vjerodajnice koje bi mu omogućile bilo kakav pristup, no u CyberBattleSimu podizanjem na ovu razinu (bez) ovlasti povlači da je čvor u vlasništvu napadača i da su mu dostupne lokalne ranjivosti nad tim čvorom.

`LocalUser` razina označava osnovni pristup korisničkom računu na čvoru. Na ovoj razini, napadač može obavljati aktivnosti koje su dostupne običnim korisnicima, ali nema administrativne privilegije koje bi mu omogućile promjene sistemskih postavki ili pristup osjetljivim podacima.

`Admin` razina privilegija pruža napadaču administrativni pristup čvoru. Na ovoj razini, napadač ima znatno veću kontrolu nad sustavom, uključujući na primjer mogućnost instaliranja softvera, promjene konfiguracija te pristup i modifikaciju osjetljivih podataka. Administrativni pristup je često cilj napadača jer omogućuje značajnu kontrolu nad čvorom.

`System` ili `MAXIMUM` razina označava najvišu razinu privilegija, pružajući napadaču potpunu kontrolu nad čvorom. Na ovoj razini, napadač može obavljati sve radnje uključujući one koje su rezervirane za sistemske administratore, kao što su na primjer promjene jezgrinih (engl. *kernel*) postavki, pristup svim datotekama i procesima te

modifikacija sistemskih sigurnosnih postavki. Ovo je najpoželjnija razina privilegija za napadača jer omogućuje neograničen pristup i kontrolu nad čvorom.

Varijabla `VulnerabilityLibrary`, prikazana u kodu 3.6, je rječnik koji pohranjuje informacije o svim poznatim ranjivostima ili značajkama podržanim u simulaciji.

```
VulnerabilityLibrary = Dict[VulnerabilityID, VulnerabilityInfo]
```

Kod 3.6: Definicija varijable `VulnerabilityLibrary`

Ovaj rječnik koristi se kao globalna biblioteka unaprijed popunjena prije početka simulacije.

Klasa `VulnerabilityInfo`, prikazana u kodu 3.7, predstavlja informacije o pojedinoj ranjivosti, uključujući opis, vrstu ranjivosti, ishod iskorištavanja, preduvjete i stope uspješnosti.

```
class VulnerabilityInfo(NamedTuple):
  description: str
  type: VulnerabilityType
  outcome: VulnerabilityOutcome
  precondition: Precondition = Precondition("true")
  rates: Rates = Rates()
  URL: str = ""
  cost: float = 1.0
  reward_string: str = ""

class VulnerabilityType(Enum):
  LOCAL = 1
  REMOTE = 2

class Rates(NamedTuple):
  probingDetectionRate: Probability = 0.0
  exploitDetectionRate: Probability = 0.0
  successRate: Probability = 1.0
```

Kod 3.7: Definicija klase `VulnerabilityInfo` i popratnih klasa

Ova klasa je srž klase `NodeInfo` preko varijable `VulnerabilityLibrary`, a uključuje detaljne informacije o ranjivosti, koje se koriste za definiranje i upravljanje sigurnosnim ranjivostima u simulaciji.

Varijabla `description` je opcionalni opis ranjivosti koji daje dodatne informacije o prirodi i karakteristikama ranjivosti. Varijabla **`type`** koristi enumeraciju `VulnerabilityType` kako bi se odredilo je li ranjivost lokalna (`LOCAL`) ili udaljena

(REMOTE). Lokalna ranjivost se može iskoristiti samo na čvoru gdje je prisutna uz prethodno kompromitiranje čvora, dok se udaljena ranjivost može iskoristiti s drugog čvora unutar mreže.

Varijabla outcome definira ishod iskorištavanja ranjivosti, što može uključivati različite vrste rezultata, poput proboja sigurnosti, krađe podataka ili eskalacije privilegija. Varijabla precondition je logički izraz koji određuje uvjete pod kojima je ranjivost prisutna na čvoru. Ovaj izraz može uključivati različita svojstva čvora, kao što su instalirani softver ili specifične konfiguracije.

precondition je zamišljen da postavlja uvjete na ranjivost čvora ovisno o prisutnosti određenih svojstava (varijable properties). Ovaj pristup omogućuje definiranje globalnih ranjivosti, a zatim, ovisno o tome kako je logički izraz postavljen u odnosu na svojstva čvora, određuje se je li određena globalna ranjivost prisutna na pojedinom čvoru ili nije. Svrha preduvjeta je da se temelje na svojstvima koja su već prisutna u čvoru. Nije moguće da agent postepeno otkriva svojstvo po svojstvo te kada se zadovolje svi uvjeti iz logičkog izraza preduvjeta, napadaču postaje moguće provesti napad. Preduvjeti su uglavnom statični po pitanju svojstava čvora, osim u slučajevima kada se uvodi logika koja kao preduvjet iznosi razine privilegija. Razine privilegija su dinamične i mogu se mijenjati tijekom simulacije, što omogućuje napredniju dinamiku simulacije gdje agent može unaprijediti svoje privilegije i time zadovoljiti preduvjete za iskorištavanje dodatnih ranjivosti.

Svojstvo rates definira stope uspješnosti povezane s ranjivošću, koristeći klasu Rates. Ova klasa sadrži tri varijable: probingDetectionRate, exploitDetectionRate i successRate. probingDetectionRate označava vjerojatnost da će agent branitelj otkriti pokušaj ove ranjivosti, exploitDetectionRate označava vjerojatnost da će iskorištavanje ove ranjivosti biti otkriveno od strane agenta branitelja, a successRate označava vjerojatnost uspješnog iskorištavanja ranjivosti od strane agenta napadača. successRate daje CyberBattleSimu svojstvo stohastičnosti procesa, zato što ako je ova varijabla postavljena na vrijednost manju od jedan, a veću od nule, istom akcijom iz istog stanja okoline neće se prijeći u isto sljedeće stanje, niti će se ostvariti ista nagrada.

Svojstvo URL pruža opcionalni link na dodatne informacije o ranjivosti, što može uključivati tehničke detalje. Svojstvo cost označava trošak povezan s iskorištavanjem ranjivosti, što može odražavati složenost ili potrebne resurse za uspješnu iskorištavanje. Konačno, reward_string je tekstualni opis koji se prikazuje kada je ranjivost uspješno iskorištena, pružajući dodatne informacije ili nagradu za napadača.

Klasa VulnerabilityOutcome u CyberBattleSimu je efektivno prazna klasa koja se koristi kao osnovna klasa za definiranje različitih ishoda iskorištavanja ranjivosti. Ostale klase koje su navedene u kodu 3.8 (VulnerabilityOutcomes listi) nasljeđuju ovu osnovnu klasu i definiraju specifične ishode napada.

```
VulnerabilityOutcomes = Union[LeakedCredentials,
LeakedNodesId, PrivilegeEscalation, AdminEscalation,
SystemEscalation, CustomerData, LateralMove, ExploitFailed]

class CachedCredential(NamedTuple):
  node: NodeID
  port: PortName
  credential: CredentialID


class LeakedCredentials(VulnerabilityOutcome):
  credentials: List[CachedCredential]

  def __init__(self, credentials: List[CachedCredential]):
    self.credentials = credentials


class LeakedNodesId(VulnerabilityOutcome):
  def __init__(self, nodes: List[NodeID]):
    self.nodes = nodes


class PrivilegeEscalation(VulnerabilityOutcome):
  def __init__(self, level: PrivilegeLevel):
  self.level = level

  @property
  def tag(self):
    return f"privilege_{self.level}"


class AdminEscalation(PrivilegeEscalation):
  def __init__(self):
  super().__init__(PrivilegeLevel.Admin)

class LateralMove(VulnerabilityOutcome):
  success: bool
```

Kod 3.8: Definicija popratnih klasa VulnerabilityOutcome klase

Klasa `CachedCredential` je definirana kao `NamedTuple` i predstavlja trojku koja sadrži informacije o čvoru, portu i vjerodajnici. Ova klasa se koristi za pohranu informacija o vjerodajnicama koje su otkrivene tijekom iskorištavanja ranjivosti.

Klasa `LeakedCredentials` nasljeđuje `VulnerabilityOutcome` i koristi instancu `CachedCredential` za pohranu vjerodajnica koje su otkrivene tijekom iskorištavanja ranjivosti. Konstruktor klase prima listu vjerodajnica i pohranjuje ih u varijablu `credentials`.

Klasa `LeakedNodesId` također nasljeđuje `VulnerabilityOutcome` i koristi se za pohranu informacija o otkrivenim čvorovima. Konstruktor klase prima listu identifikatora čvorova i pohranjuje ih u varijablu `nodes`.

Klasa `PrivilegeEscalation` nasljeđuje `VulnerabilityOutcome` i koristi se za pohranu informacija o eskalaciji privilegija. Konstruktor klase prima razinu privilegija (engl. *privilege level*) i pohranjuje je u varijablu `level`. Klasa također sadrži svojstvo (engl. *property*) `tag`, koje vraća tekst predstavljen razinom privilegija, koristeći prefiks `privilege_`.

Klasa `AdminEscalation` nasljeđuje `PrivilegeEscalation` i predstavlja specifičan tip eskalacije privilegija na administratorsku razinu. Konstruktor klase automatski postavlja razinu privilegija na `PrivilegeLevel.Admin` pozivom konstruktora nad-klase s odgovarajućom razinom privilegija.

Klasa `SystemEscalation` definirana je na isti način kao `AdminEscalation`, ali postavlja razinu privilegija na `PrivilegeLevel.System`. Ova klasa omogućuje modeliranje najviših razina privilegija koje napadač može dobiti.

Klasa `CustomerData` nasljeđuje `VulnerabilityOutcome` i koristi se za označavanje ishoda u kojem napadač dobiva pristup podacima korisnika na ciljanom čvoru. Ova klasa ne sadrži dodatna svojstva ili metode, ali označava specifičan tip uspješnog iskorištavanja koji uključuje krađu osjetljivih podataka.

Klasa `LateralMove` koristi se za označavanje lateralnog pokreta unutar mreže, gdje napadač uspijeva premjestiti svoju poziciju s jednog čvora na drugi. Varijabla `success` označava uspješnost ovog pokreta. Ako je `success` postavljen na `True`, to znači da je napadač uspješno izvršio lateralni pokret i preuzeo kontrolu nad ciljnim čvorom.

Klasa `ExploitFailed` nasljeđuje `VulnerabilityOutcome` i koristi se za situacije u kojima iskorištavanje ranjivosti nije uspjelo. Ova klasa ne sadrži dodatna svojstva ili metode, ali jasno označava da napad nije bio uspješan, što može utjecati na daljnje akcije napadača i ukupnu strategiju napada.

3.2.2. Okolina

Okolina ima svojstva statičnosti, diskretnosti, determinističnosti i djelomične vidljivosti (engl. *partially observability*). Glavna posljedica statičnosti je ograničenje okoline da tijekom simulacije ne može nastati novi čvorovi, stoga nakon pokretanja ne mogu postojati akcije koje dopunjuju mrežnu topologiju. Diskretnost označava da su moguće akcije i stanja jasno definirane i ograničene na određeni skup vrijednosti, propisanih unutar CyberBattleSima kako bi agenti mogli opažati stanje mreže kroz niz diskretnih, specifičnih informacija. Svojstvo determinističnosti upućuje da ako akcija kao rezultat ima definirano curenje informacija o x čvoru, nemoguće je da poduzimanje te akcije rezultira curenje informacija o y čvoru ili nekim drugim rezultatom. Četvrto svojstvo djelomične vidljivosti nužno je da agent napadač nema mogućnost uvida u cijelu topologiju mreže odmah, nego time što djelomično vidi samo dio topologije, poduzima akcije kojima kroz simulaciju postepeno razotkriva ostatak čvorova i njihov ranjivosti u mreži.

3.2.3. Prostor akcija

Prostor akcija sastoji se od tri vrste akcija: lokalnog napada (engl. *local attack*), udaljenog napada (engl. *remote attack*) i akcije autentificiranog povezivanja (engl. *authenticated connection*). Ove akcije omogućuju napadaču u CyberBattleSimu da se kreće kroz mrežu, iskorištava ranjivosti i preuzima kontrolu nad čvorovima.

Lokalni napad koristi se za iskorištavanje ranjivosti unutar čvora koji je već kompromitiran. Ova vrsta napada omogućuje napadaču da dobije dodatne informacije ili vjerodajnice koje mogu biti korisne za daljnje napade. Na primjer, naredba za lokalni napad može izgledati kako je prikazano u kodu 3.9.

```
c2.run_attack('Pocetno zarazeni cvor',
   'PretraziPovijestPreglednika')
```

Kod 3.9: Lokalni napad

U ovoj naredbi `Pocetno zarazeni cvor` predstavlja ime čvora na kojem se napad izvodi, dok `PretraziPovijestPreglednika` označava specifičnu ranjivost ili akciju koju napadač koristi za iskorištavanje. Argument `Pocetno zarazeni cvor` omogućuje napadaču da nacilja specifični čvor unutar mreže, dok `PretraziPovijestPreglednika` određuje specifičnu tehniku koja se koristi za dobivanje dodatnih informacija ili vjerodajnica.

Udaljeni napad omogućuje napadaču da iskoristi ranjivosti na drugom čvoru unutar mreže. Ova akcija zahtijeva da napadač već ima određeni nivo pristupa mreži kako bi mogao identificirati i iskoristiti ranjivosti na udaljenom čvoru. Primjer naredbe za udaljeni

prikazan je u kodu 3.10.

```
c2.run_remote_attack('Pocetno zarazeni cvor', 'Web stranica',
   'SkeniranjeIzvornogKodaStranice')
```

Kod 3.10: Udaljeni napad

Ovdje `Pocetno zarazeni cvor` predstavlja čvor s kojeg napadač započinje napad, `Web stranica` je ciljni čvor napada, a `SkeniranjeIzvornogKodaStranice` označava specifičnu tehniku napada koja se koristi za iskorištavanje ranjivosti na ciljnom čvoru. `SkeniranjeIzvornogKodaStranice` određuje metodu iskorištavanja koja se koristi za dobivanje dodatnih informacija ili vjerodajnica s udaljenog čvora.

Akcija autentificiranog povezivanja omogućuje napadaču da koristi spremljene vjerodajnice za lateralno kretanje i preuzimanje kontrole nad udaljenim čvorom. Ova akcija zahtijeva prethodno stečene vjerodajnice koje omogućuju napadaču da se autentificira na ciljanom čvoru. Naredba za ovu akciju može izgledati kako je prikazano u kodu 3.11.

```
c2.connect_and_infect('Pocetno zarazeni cvor', 'Web stranica',
   'Port', 'PrimjerVjerodajnica')
```

Kod 3.11: Autentificirano povezivanje

U ovoj naredbi `Pocetno zarazeni cvor` je čvor s kojeg napadač započinje povezivanje, `Web stranica` je ciljani čvor, `Port` predstavlja mrežni port na kojem se vrši povezivanje, a `PrimjerVjerodajnica` označava vjerodajnice koje napadač koristi za autentifikaciju. Port određuje specifičnu ulaznu točku za mrežnu komunikaciju, a `PrimjerVjerodajnice` omogućuje autentifikaciju na ciljanom čvoru, pružajući napadaču mogućnost preuzimanje kontrole.

3.2.4. Nagrada

Nakon svakog koraka akcije napadač dobiva nagrade koje su određene vrstom akcije, rezultatom akcije i vrijednošću čvora na kojem je akcija izvršena kako je definirano u CyberBattleSim okruženju. Isječak koda kojim su definirane nagrade u CyberBattleSima je prikazan kodom 3.12.

```
NEW_SUCCESSFULL_ATTACK_REWARD = 7
NODE_DISCOVERED_REWARD = 5
CREDENTIAL_DISCOVERED_REWARD = 3
PROPERTY_DISCOVERED_REWARD = 2
```

Kod 3.12: Definirane nagrade u CyberBattleSimu

U CyberBattleSimu, napadač može dobiti nekoliko vrsta nagrada temeljenih na uspješno izvršenim akcijama. Prva vrsta nagrade je za uspješno izvršenje lokalnog ili udaljenog napada po prvi put nakon što je ciljani čvor zadnji put osvježen. Ova nagrada je definirana kao `NEW_SUCCESSFULL_ATTACK_REWARD` i iznosi 7 bodova. S naglaskom da se trošak napada oduzima od `NEW_SUCCESSFULL_ATTACK_REWARD` nagrade, što znači da neto nagrada može varirati ovisno o trošku specifične akcije.

Nagrada za otkrivanje novog čvora u mreži definira se kao `NODE_DISCOVERED_REWARD` i iznosi 5 bodova. Ovo potiče napadača na istraživanje mreže i otkrivanje novih čvorova koje može potencijalno kompromitirati. Osim toga, postoji i nagrada za otkrivanje novih vjerodajnica, definirana kao `CREDENTIAL_DISCOVERED_REWARD`, koja iznosi 3 boda. Otkrivanje vjerodajnica omogućuje napadaču da pristupi dodatnim resursima unutar mreže, čime se povećava njegova sposobnost za provođenje daljnjih napada.

Konačno, otkrivanje novih svojstava čvora također donosi nagradu, definiranu kao `PROPERTY_DISCOVERED_REWARD`, koja iznosi 2 boda. Ova nagrada potiče napadača na detaljno istraživanje svakog čvora kako bi prikupio što više informacija koje mogu biti korisne za buduće napade.

Ovaj sustav nagrađivanja omogućuje napadaču da uči iz svojih akcija i prilagođava svoje strategije kako bi maksimizirao svoje uspjehe u budućim napadima. Međutim, uz ove nagrade, CyberBattleSim također uključuje i kazne ili negativne nagrade za određene radnje koje napadač može poduzeti tijekom simulacije. Kazne su osmišljene kako bi demotivirale napadača od poduzimanja sumnjivih, neefikasnih ili neispravnih radnji, pružajući negativnu povratnu informaciju i smanjujući ukupnu nagradu napadača. Kazne su u CyberBattleSimu implementirane kroz klasu `Penalty` prikazanu u kodu 3.13.

```
class Penalty:
  SUSPICIOUSNESS = -5.0
  SCANNING_UNOPEN_PORT = -10.0
  REPEAT = -1
  LOCAL_EXPLOIT_FAILED = -20
  FAILED_REMOTE_EXPLOIT = -50
  MACHINE_NOT_RUNNING = 0
  WRONG_PASSWORD = -10
  BLOCKED_BY_LOCAL_FIREWALL = -10
  BLOCKED_BY_REMOTE_FIREWALL = -10
  INVALID_ACTION = -1
```

Kod 3.13: Definirane kazne u CyberBattleSimu

`SUSPICIOUSNESS` predstavlja kaznu od -5 bodova za generičku sumnjivost, čime se

smanjuje ukupna nagrada napadača za radnje koje bi mogle biti prepoznate kao sumnjive od strane obrambenih sustava.

Ako napadač pokuša uspostaviti vezu na port koji nije otvoren, dobiva kaznu `SCANNING_UNOPEN_PORT` od -10 bodova. Ova kazna potiče napadača da pažljivije bira ciljeve za napad temeljem prethodno prikupljenih informacija o otvorenim portovima.

Ponavljanje iste ranjivosti donosi kaznu `REPEAT` od -1 bod, što smanjuje ukupnu nagradu napadača za neuspješno ponavljanje istih napada. Neuspješan lokalni napad rezultira kaznom `LOCAL_EXPLOIT_FAILED` od -20 bodova, dok neuspješan udaljeni napad donosi veću kaznu `FAILED_REMOTE_EXPLOIT` od -50 bodova, zbog većeg rizika i složenosti ovih napada.

Ako napadač pokuša izvršiti akciju na čvoru koji nije u radnom stanju, kazna `MACHINE_NOT_RUNNING` iznosi 0 bodova, što implicira da takva akcija nema posljedica na ukupnu nagradu. Pokušaj povezivanja s neispravnom lozinkom donosi kaznu `WRONG_PASSWORD` od -10 bodova, dok blokiranje prometa zbog lokalnog ili udaljenog vatrozida rezultira kaznama `BLOCKED_BY_LOCAL_FIREWALL` i `BLOCKED_BY_REMOTE_FIREWALL`, obje iznose -10 bodova.

Konačno, ako napadač pokuša izvršiti nevaljanu akciju, poput pokretanja napada s čvora koji nije pod njegovom kontrolom, dobiva kaznu `INVALID_ACTION` od -1 bod, pod uvjetom da opcija `throws_on_invalid_actions` nije postavljena na `True`.

Kombinacija nagrada i kazni u CyberBattleSimu omogućuje napadačima da uče iz svojih akcija i prilagođavaju svoje strategije kako bi maksimizirali uspjeh u budućim napadima, dok istovremeno smanjuju rizik od neuspjeha i detekcije.

3.2.5. Prostor promatranja

U CyberBattleSimu, prostor promatranja omogućuje napadačkom agentu da dobije uvid u stanje mreže tijekom simulacije. Ključni elementi prostora promatranja uključuju otkrivene čvorove (engl. *discovered nodes*), kompromitirane čvorove koje agent posjeduje (engl. *owned nodes*), otkrivene vjerodajnice (engl. *discovered credentials*), dostupne napade (engl. *available attacks*) na svaki čvor i razinu eskalacije (engl. *escalation level*) svakog pojedinog čvora.

Agent ne vidi sve čvorove u mrežnoj topologiji, već samo otkrivene čvorove, što mu omogućuje da stekne pregled nad mrežom i identificira potencijalne mete za buduće napade. Otkriveni čvorovi su oni koje je agent otkrio kroz prethodne akcije, bilo kroz pretraživanje mreže ili iskorištavanje ranjivosti. Kompromitirani čvorovi koje agent posjeduje su oni nad kojima je agent već preuzeo kontrolu. Ovi čvorovi predstavljaju strateške točke s kojih agent može pokretati daljnje napade i širenje unutar mreže.

Osim toga, agent ima pristup otkrivenim vjerodajnicama koje su ključne za provođenje

napada na druge čvorove. Vjerodajnice omogućuju agentu autentifikaciju i pristup novim resursima unutar mreže, čime se povećava njegova sposobnost za lateralno kretanje i eskalaciju napada.

Dostupni napadi na svaki čvor predstavljaju set akcija koje agent može poduzeti kako bi iskoristio ranjivosti na određenom čvoru. Ove informacije omogućuju agentu da planira svoje napade temeljeno na specifičnim ranjivostima koje su identificirane na ciljanim čvorovima.

Naposljetku, razina eskalacije svakog pojedinog čvora označava stupanj kontrole koji agent ima nad tim čvorom. Razina eskalacije može varirati od osnovne kontrole do potpune dominacije nad čvorom kroz četiri razine, što agentu omogućuje izvođenje sve sofisticiranijih i štetnijih napada.

Prostor promatranja u CyberBattleSimu stoga pruža agentu sve potrebne informacije za donošenje informiranih odluka i učinkovito planiranje napada unutar simuliranog mrežnog okruženja.

3.3. Postavljanje i konfiguracija simulacija

Postavljanje i konfiguracija CyberBattleSima uključuje nekoliko ključnih koraka, počevši od preuzimanja i postavljanja alata, konfiguracije razvojnog okruženja te kreiranja vlastite mrežne topologije i pokretanja simulacije.

3.3.1. Preuzimanje i postavljanje CyberBattleSima

Prvi korak u postavljanju CyberBattleSima je preuzimanje alata s *git* repozitorija. CyberBattleSim je projekt otvorenog koda koji se može preuzeti i instalirati kao što je prikazano kodom 3.14 u terminalu. Za ovu naredbu potrebno je imati instaliran i pravilno postavljen `git`.

```
git clone https://github.com/microsoft/CyberBattleSim.git
```

Kod 3.14: Preuzimanje CyberBattleSima

Nakon preuzimanja koda, u ovom radu korišten je *Docker* [4] za pokretanje na operacijskom sustavu *Windows* putem *Windows Subsystem for Linux* – a. Nadalje, korišten je *VS Code* [13] kao razvojno okruženje, a posebno njegov dodatak (engl. *plugin*) *Dev Containers* za razvoj unutar *Docker container* – a. Za daljnje korake podrazumijevat će se uspješno postavljanje ovih sustava.

Nakon toga otvaranje CyberBattleSima u okolini izvodi se na sljedeći način:

1. otvaranje *VS Code* razvojnog okruženja

2. unutar *VS Code* otvorite preuzeti direktorij `CyberBattleSim`

3. kliknite na ikonu `Open a Remote Window` u donjem lijevom kutu *VS Code* prozora

4. odaberite opciju `Reopen in Container`.

VS Code će sada automatski prepoznati konfiguracijske datoteke za *Docker* i postaviti razvojno okruženje unutar *Docker container* – a. Ova radnja prilikom inicijalnog pokretanja može potrajati više od sat vremena ovisno o brzini mreže, pošto se moraju unutar *Docker container* – a instalirati biblioteke koje stoje u pozadini CyberBattleSima.

Kao konfiguracija kreiranja unutar *Dev Containers* – a može se koristiti datoteka `devcontainer.json` koja se nalazi unutar direktorija `.devcontainer`. Ovaj direktorij može se opcionalno stvoriti prije pokretanja ili se automatski stvara s konfiguracijom prilikom prvog pokretanja. Izmijenjena konfiguracija prikazana je kodom 3.15

```
{
  "name": "Existing Dockerfile",
  "build": {
    "context": "..",
    "dockerfile": "../Dockerfile"
  },
  "features": {
    "ghcr.io/devcontainers/features/nvidia-cuda:1": {}
  },
  "runArgs": ["--gpus", "all"]
}
```

Kod 3.15: Konfiguracijska datoteka `devcontainer.json`

Parametar `runArgs` sadrži dvije ključne stavke: `"--gpus"` i `"all"`. Korištenjem ovih argumenata postiže se da svi dostupni grafički procesori na sustavu domaćina postanu dostupni unutar *Docker container* – a. Ova mogućnost je posebno korisna jer biblioteke koje koristi CyberBattleSim imaju podršku za paralelizaciju uz pomoć grafičkih procesora. Kada su grafički procesori dostupni, simulacije mogu iskoristiti dodatnu računalnu snagu koju oni pružaju, što dovodi do značajnog ubrzanja izvođenja simulacija. U nekim slučajevima, ovisno o vrsti i specifikacijama grafičkog procesora, može se postići ubrzanje i više do tri puta u odnosu na izvođenje bez grafičkih procesora.

3.3.2. Kreiranje vlastite mrežne topologije

Nakon što je razvojno okruženje postavljeno, može se kreirati vlastita mrežna topologija za simulaciju. Mrežna topologija se definira u *Python* skripti. Na primjer, kreirajmo

jednostavnu topologiju s nekoliko čvorova i ranjivosti, pri čemu je u kodu 3.16 prikazan samo jedan čvor radi sažetosti.

```
from cyberbattle.simulation import model as m
from cyberbattle.simulation.model import NodeID, NodeInfo,
   VulnerabilityID, VulnerabilityInfo
from typing import Dict, Iterator, cast, Tuple

nodes = {"Pocetno␣zarazeni␣cvor": m.NodeInfo(
    services=[],
    firewall=m.FirewallConfiguration(
      outgoing=[m.FirewallRule("HTTPS", m.RulePermission.ALLOW)]
    ),
    properties=["Windows11"],
    value=0,
    vulnerabilities=dict(
      PretraziPovijestPreglednika=m.VulnerabilityInfo(
        description="Pretrazite␣povijest␣preglednika␣za␣popis␣web␣
           stranica␣kojima␣je␣pristupljeno",
        type=m.VulnerabilityType.LOCAL,
        outcome=m.LeakedNodesId(["Web␣stranica"]),
        reward_string="Povijest␣web␣preglednika␣otkrila␣je␣URL␣web␣
           stranice␣od␣interesa",
        cost=1.0)),
    agent_installed=True)}

global_vulnerability_library: Dict[VulnerabilityID,
   VulnerabilityInfo] = dict([])

ENV_IDENTIFIERS = m.infer_constants_from_nodes(
  cast(Iterator[Tuple[NodeID, NodeInfo]],
  list(nodes.items())),
  global_vulnerability_library)

def new_environment() -> m.Environment:
  return m.Environment(network=m.create_network(nodes),
    vulnerability_library=global_vulnerability_library,
    identifiers=ENV_IDENTIFIERS)
```

Kod 3.16: Skraćena `myEnv.py` datoteka

Ovaj kod uključuje (engl. *import*) potrebne module za definiciju mrežnih čvorova i ranjivosti. Modul `cyberbattle.simulation` uključen je kao `m` i koristi se za pristup različitim klasama i funkcijama koje su ključne za definiranje mrežne topologije. Uključuju se klase `NodeID`, `NodeInfo`, `VulnerabilityID`, `VulnerabilityInfo` te potrebni tipovi iz `typing` modula, kao što su `Dict`, `Iterator`, `cast` i `Tuple`.

U definiciji čvorova (`nodes`), definiramo čvor nazvan `Pocetno zarazeni cvor` s nekoliko ključnih atributa, uključujući vatrozid konfiguraciju, svojstva (u ovom slučaju `Windows11`), ranjivosti te svojstvo da je agent napadač instaliran. Među atributima je i vrijednost čvora koja je nula zato što je ovaj čvor početno zaražen te njegova vrijednost ne utječe na daljnje radnje unutar simulacije. Definirana ranjivost `PretraziPovijestPreglednika` je lokalna ranjivost koja otkriva informacije o web stranicama pristupljenim putem preglednika.

Nakon definiranja čvorova, kreira se globalna biblioteka ranjivosti (`global_vulnerability_library`) koja je trenutno prazna. Zatim, pomoću funkcije `m.infer_constants_from_nodes`, definiramo konstante okruženja (`ENV_IDENTIFIERS`) koje identificiraju različite entitete u mreži.

Funkcija `new_environment` kreira novo simulacijsko okruženje pomoću definirane mreže čvorova i globalne biblioteke ranjivosti. Ova funkcija vraća instancu klase `Environment` koja se koristi za pokretanje simulacija.

Ova datoteka `myEnv.py` je kreirana u direktoriju `cyberbattle/samples/my_env`.

Nakon kreiranja `myEnv.py` datoteke, potrebno je modificirati datoteku `cyberbattle/__init__.py` kako bi se nova topologija registrirala kako je prikazano u kodu 3.17.

```
from .samples.my_env import myEnv

if 'CyberBattleMy-v0' in registry.env_specs:
  del registry.env_specs['CyberBattleMy-v0']

register(
  id='CyberBattleMy-v0',
  cyberbattle_env_identifiers=myEnv.ENV_IDENTIFIERS,
  entry_point='cyberbattle._env.cyberbattle_my:CyberBattleMy',
  kwargs={'defender_agent': None,
    'attacker_goal': AttackerGoal(own_atleast=6),
    'defender_goal': DefenderGoal(eviction=True)
  }
)
```

Kod 3.17: Isječak nadodan u `cyberbattle/__init__.py`

Ovaj kod uključuje novo kreiranu `myEnv.py` datoteku. Ako već postoji registracija za `CyberBattleMy-v0`, ona se uklanja. Zatim se nova simulacija registrira s odgovarajućim identifikatorima okruženja, početnom točkom (engl. *entry point*) te ciljevima napadača i branitelja. Registracija omogućuje CyberBattleSimu da prepozna i koristi novu mrežnu topologiju. Također je potrebno kreirati novu datoteku `cyberbattle_my.py` u direktoriju `cyberbattle_env` s kodom 3.18.

```
from ..samples.my_env import myEnv
from . import cyberbattle_env

class CyberBattleMy(cyberbattle_env.CyberBattleEnv):
  def __init__(self, **kwargs):
    super().__init__(
    initial_environment=myEnv.new_environment(),
    **kwargs
    )
```

Kod 3.18: Kreirana datoteka `cyberbattle_my.py`

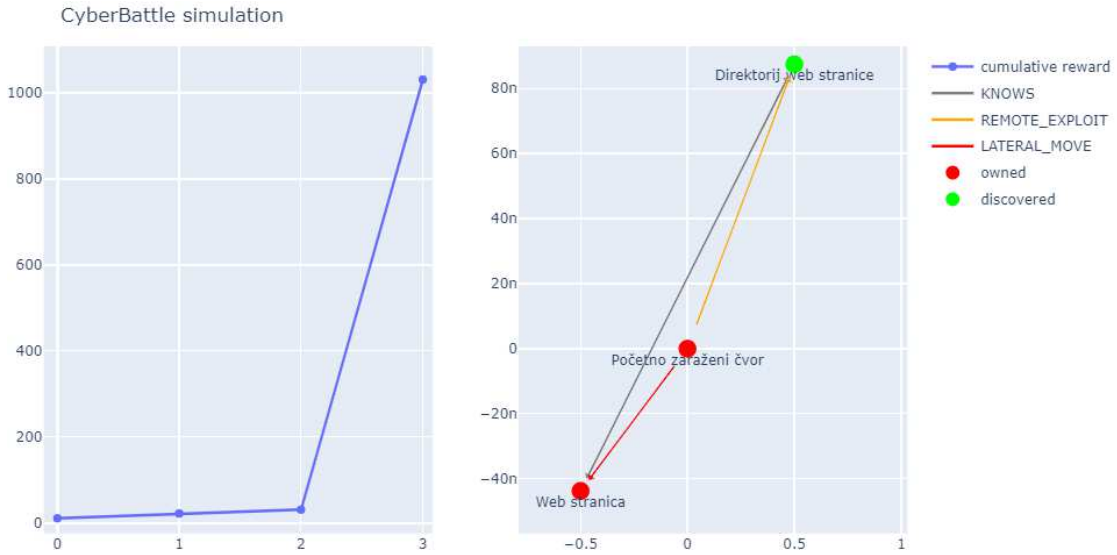
Ovaj kod uključuje `myEnv` modul i definira novu klasu `CyberBattleMy` koja nasljeđuje `CyberBattleEnv`. Korištenjem konstruktora nadklase, `CyberBattleMy` se inicijalizira s novim okruženjem simulacije definiranim u `myEnv`.

Nakon što su svi ovi koraci izvršeni, moguće je kreirati *Jupyter* bilježnicu za pokretanje i testiranje nove mrežne topologije u CyberBattleSimu, što će biti detaljno objašnjeno u nastavku rada.

Konačan rezultat treniranja i pokretanja agenta kroz bilježnicu nad gore izvedenim primjerom je prikazan slikom 3.1.

Na slici su prikazani rezultati simulacije u dva grafikona. Lijevi grafikon prikazuje kumulativnu nagradu koju agent postiže tijekom simulacije. Kumulativna nagrada prikazana je plavom linijom koja raste kako agent napreduje kroz mrežu, otkriva nove čvorove i uspješno izvodi napade. Na grafu se može vidjeti značajan porast kumulativne nagrade u trećem koraku simulacije, što sugerira uspješno iskorištavanje ranjivosti ili otkriće važnih informacija.

Desni grafikon prikazuje mrežnu topologiju i interakcije agenta s okolinom. Čvorovi u mreži su predstavljeni krugovima, pri čemu su crveni čvorovi `owned` oni koje je agent kompromitirao, dok su zeleni čvorovi `discovered` oni koje je agent otkrio, ali nisu još kompromitirani. Linije između čvorova predstavljaju različite tipove akcija koje je agent



Slika 3.1: Izlazni grafovi CyberBattleSima

poduzeo:

- sive linije (`KNOWS`) označavaju poznate veze između čvorova
- narančaste linije (`REMOTE_EXPLOIT`) označavaju uspješne udaljene napade
- crvene linije (`LATERAL_MOVE`) označavaju lateralne pokrete agenta unutar mreže.

Na grafu se vidi kako je agent krenuo s `Pocetno zarazeni cvor` (u sredini), uspješno izveo lateralni pokret prema `Web stranica` (lijevo) te izveo udaljeni napad prema `Direktorij web stranice` (gore desno), koristeći ranjivosti. Kroz ovu interakciju, agent je otkrio i kompromitirao ključne čvorove u mreži, što je rezultiralo povećanjem kumulativne nagrade prikazane na lijevom grafu.

4. Razvoj alata za preslikavanje okruženja

U kontekstu kibernetičke sigurnosti, emulacija ili simulacija napadača postaje sve važnija kako bi se bolje razumjele potencijalne prijetnje i unaprijedila obrana sustava. Ručno provođenje penetracijskih testova od strane stručnjaka je skupo, vremenski zahtjevno i podložno varijacijama u kvaliteti [17]. Stoga se sve više pažnje posvećuje razvoju automatiziranih alata koji mogu simulirati napade, omogućujući skalabilnost i ponovljivost testiranja. Microsoftov alat CyberBattleSim, detaljnije objašnjen u prethodnim poglavljima, predstavlja takvu okolinu, pružajući platformu za treniranje i testiranje autonomnih agenata u simuliranim mrežnim okruženjima.

Kako bi se dodatno unaprijedila sposobnost testiranja i treniranja, potrebno je integrirati različite simulacijske okoline. U ovom kontekstu, razvijen je alat za preslikavanje okruženja iz CCS – a u CyberBattleSim. CCS je specifičan simulator koji pruža detaljne scenarije i topologije za testiranje sigurnosnih sustava. U ovom poglavlju opisan je postupak preslikavanja okruženja iz CCS – a u CyberBattleSim, uključujući opis i funkcionalnosti CCS – a te implementaciju procesa preslikavanja.

4.1. Opis i funkcionalnosti Cyber Conflict Simulatora

Cyber Conflict Simulator je napredan alat dizajniran za simulaciju kompleksnih kibernetičkih napada i obrambenih scenarija. Omogućuje korisnicima stvaranje realističnih mrežnih okruženja i simuliranje različitih vrsta napada kako bi se bolje razumjele ranjivosti i testirale obrambene strategije [21].

Jedna od ključnih funkcionalnosti CCS – a je mogućnost definiranja različitih vrsta računala i uređaja unutar mreže. Svaki uređaj može imati pridružene oznake kao što su `Machine`, `Asset` i `Operating System`, što omogućuje detaljno modeliranje karakteristika svakog čvora u mreži. Uređaji su prikazani pomoću ikona koje označavaju stanje napajanja, prisutnost povezanih elemenata poput datoteka ili usluga te veze s mrežom ili korisnicima [8]. Oznake određuju atribute koji su prisutni na čvoru, pa će tako na primjer

kombinacija: `Machine`, `Asset`, `Operating System`, `Backup Server` i `File System` imati sve atribute `Machine`, `Asset` i `Operating System` sa nadodanim atributima ove kombinacije oznaka.

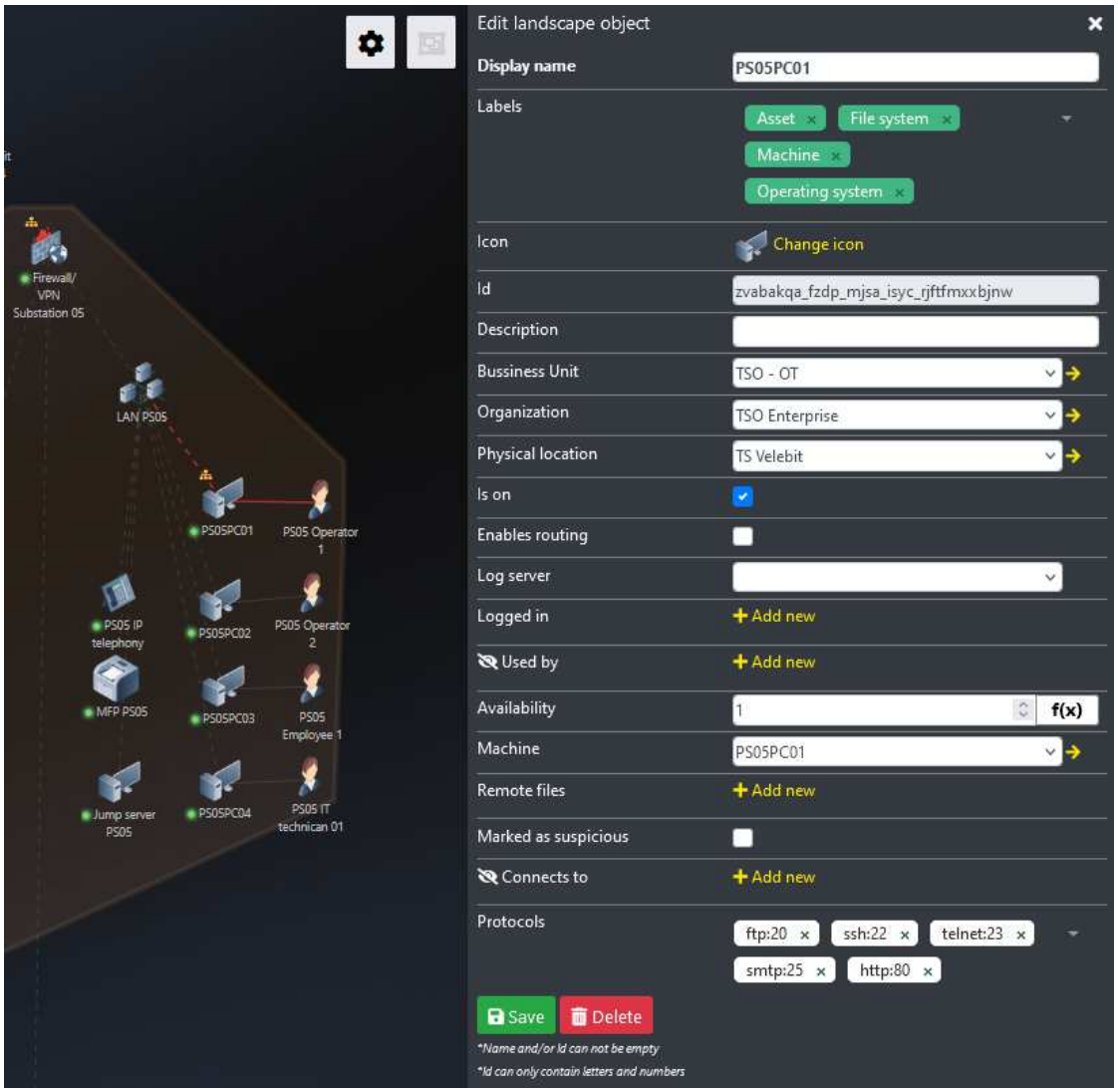
U CCS – u su veze između računala označene čvrstim linijama ako računalo koristi korisnik ili isprekidanim linijama ako je računalo povezano na mrežu bez korisnika. Na slici 4.1 prikazan je isječak dijela ekrana iz simulatora pri izmjeni mrežne topologije. Svako računalo ima atribute kao što su naziv, oznake, ikona, ID, opis, poslovna jedinica, organizacija, fizička lokacija, status uključenosti, mogućnost usmjeravanja prometa, server za bilježenje (engl. *log server*), trenutni korisnik ili korisnici koji ga koriste, dostupnost, popis protokola i port, vrijeme potrebno za ponovnu instalaciju, veličina, implementirane sigurnosne kontrole, kontrolne stanice, kontrolni protokoli i smjer, funkcionalnosti, korisnički računi, konfiguracija posrednik (engl. *proxy*) poslužitelja, akcije pri izmjeni, sadržaj memorije, datum ažuriranja softvera, verzija softvera, informacije o softveru, datotečni sustavi, datoteke i instalirani softver.

Osim detaljnog modeliranja pojedinih uređaja, aktera, organizacija, datoteka i ostale topologije, CCS omogućava simulaciju različitih vrsta napada. Na primjer, može se simulirati *phishing* napad, gdje se stvara e-pošta s zlonamjernim privitkom ili poveznicom na zlonamjernu web stranicu. Također, moguće je simulirati napade poput infekcije web stranice, krađe vjerodajnica, eskalacije privilegija, skeniranja mreže i mnoge druge. Svaka vrsta napada može imati specifične preduvjete i ishode.

CCS također omogućava definiranje i upravljanje sigurnosnim kontrolama kao što su vatrozidi, antivirusne zaštite, kontrola pristupa i druge sigurnosne mjere. Ovo omogućava korisnicima da testiraju učinkovitost različitih sigurnosnih postavki i strategija obrane protiv simuliranih napada. Kontrole se mogu definirati na razini pojedinačnih uređaja ili mrežnih segmenata (na primjer uz pomoć vatrozida).

Jedna od važnih funkcionalnosti CCS – a je mogućnost upravljanja vjerodajnicama. Spremišta vjerodajnica omogućavaju pohranu i upravljanje korisničkim računima, lozinkama i drugim osjetljivim informacijama. Vjerodajnice se koriste za autentifikaciju korisnika i kontrolu pristupa različitim resursima unutar mreže. Upravljanje vjerodajnicama je ključno za simulaciju napada koji uključuju krađu ili zloupotrebu vjerodajnica te za testiranje sigurnosnih mjera koje ih štite.

Na slici 4.1 vidljivo je da se izmjena topologije provodi jednostavno kroz grafičko sučelje, a cijela interakcija sa simulatorom prilikom opisivanja topologije te napada, ali i tijekom same simulacije vrši se kroz `HTTPS` protokol unutar internet preglednika. Nakon što je opisana topologija i tijek napada kroz *Editor* CCS – a (dio simulatora za izradu topologije), može se koristiti u *Simulator* dijelu CCS – a koji omogućava kontrolu nad simulacijom, uključujući pokretanje i zaustavljanje simulacija, praćenje rezultata te



Slika 4.1: Isječak dijela ekrana iz CCS – a

interakciju s različitim elementima unutar simulacije. Simulacija može uključivati više igrača, pri čemu svaki igrač može kontrolirati jednu ili više topologija, odnosno aktera [8]. Ovo omogućava stvaranje složenih scenarija gdje različiti timovi mogu testirati svoje sigurnosne strategije i obrane protiv napada u realnom vremenu.

Tijekom same simulacije moguće je da fizička osoba upravlja napadačem kao jednim od aktera unutar simulacije, ili da se iskoristi dio CCS – a u kojemu se unaprijed definira akcijske sekvence (engl. *action sequences*). Akcijske sekvence definiraju scenarij slijeda radnji koje su smještene u vremenu simulacije i mogu ovisiti jedna o drugoj te dopuštaju da se unaprijed pripremi napad unutar simulacije.

4.2. Razlika između CCS – a i CyberBattleSima

CyberBattleSim je alat fokusiran na simulaciju kibernetičkih napada i obrambenih strategija unutar simuliranih mrežnih okruženja. Iako dijeli sličnosti s CCS – om u smislu simulacije mrežnih napada, postoje značajne razlike u pristupu i funkcionalnostima ovih alata.

CCS omogućava vrlo detaljno modeliranje mrežnih okruženja s naglaskom na realistične scenarije i detaljne sigurnosne postavke. Svaki čvor u mreži može imati pridružene specifične atribute, poput verzije softvera, postavki vatrozida, konfiguracija posrednik poslužitelja i drugih karakteristika koje odražavaju stvarne mrežne okoline. Napadi u CCS – u mogu uključivati kompleksne scenarije s višestrukim preduvjetima i specifičnim ishodima, omogućujući korisnicima da detaljno testiraju i analiziraju sigurnosne mjere.

S druge strane, CyberBattleSim koristi apstraktniji pristup, fokusirajući se na ključne elemente potrebne za treniranje agenata za strojno učenje. Simulirana mrežna okruženja u CyberBattleSimu predstavljena su pomoću grafova, gdje su čvorovi zamišljeni da predstavljaju računala ili uređaje, a veze između njih predstavljaju mrežne veze. Ovaj pristup omogućava brzu iteraciju i eksperimentiranje s različitim strategijama napada i obrane, ali žrtvuje određeni stupanj detalja u usporedbi s CCS – om.

Jedna od ključnih prednosti CyberBattleSima je njegova integracija s alatima za strojno učenje, što omogućava treniranje agenata koji mogu autonomno istraživati mrežna okruženja, otkrivati ranjivosti i razvijati strategije napada.

Implementacija preduvjeta za napade također se razlikuje između ova dva alata. U CCS – u, preduvjeti za napade mogu biti vrlo specifični i uključivati detaljne uvjete poput prisutnosti određenih datoteka, specifičnih konfiguracija uređaja, ili prisutnosti određenih usluga. U CyberBattleSimu, preduvjeti su obično apstraktniji, fokusirajući se na dostupnost određenih resursa ili pristupačnih vjerodajnica. Ovaj pristup omogućava fleksibilniju simulaciju, ali može zahtijevati dodatnu prilagodbu kako bi se postigli specifični scenariji kao što će biti prikazano u nastavku. Posebno je za naglasiti da CyberBattleSim nema mehanizam kojim se postavlja preduvjet za neku ranjivost ovisno o prethodnoj ranjivosti nad čvorom ako taj čvor nije kompromitiran, a kada je kompromitiran to se postiže isključivo podizanjem ovlasti. Nadalje, nije moguće postaviti preduvjet za ranjivost u ovisnosti o stanju nekog drugog čvora u mreži.

Ova svojstva ograničavaju preslikavanje složenih funkcionalnosti modeliranih CCS – om na apstraktne funkcionalnosti modelirane CyberBattleSimom.

4.3. Implementacija preslikavanja okruženja

Potpuna implementacija preslikavanja okruženja iz CCS – a u CyberBattleSim zahtijeva detaljno razumijevanje oba alata i njihovu sposobnost da modeliraju različite aspekte mrežnih okruženja i napadačkih scenarija. Analizom ograničenja CyberBattleSima u odnosu na CCS – a, bez izmjene nad CyberBattleSimom, nije moguće posve preslikati okolinu iz jednog alata u drugi. Također, dok je CyberBattleSim otvorenog koda, CCS je zatvorenog koda (engl. *closed source*) i nije posve javno poznat njegov slijed radnji i otvaranje novih ranjivosti. Iz tih razloga u nastavku rada bit će prikazano preslikavanje ranjivosti vezanih isključivo uz scenarij *phishing* napada. Ovo će se pokazati kao dokaz koncepta (engl. *proof of concept*, PoC) da je moguće preslikati funkcionalnost (pa makar dio) iz CCS – a u CyberBattleSim na upotrebljiv način.

Cilj je preslikati *JSON* (*JavaScript Object Notation*) datoteku koju CCS generira prilikom preuzimanja definirane okoline van simulatora u CyberBattleSim *JSON* okolinu. CyberBattleSim okolinu prima u *Python* obliku. Kao konačni izlaz preslikavanja transformirana okolina u opisnik *Python* CyberBattleSim okoline, sličnom ranije objašnjenoj datoteci `myEnv.py` u kodu 3.16.

Implementaciji je pristupljeno u dva koraka:

1. preslikavanje CyberBattleSim *JSON* topologije u CyberBattleSim okolinu
2. preslikavanje CCS *JSON* topologije u CyberBattleSim *JSON* topologiju.

Prilikom izvršavanja ove funkcionalnosti, koraci će se izvoditi obrnutim redom, odnosno prvo će se preslikati CCS *JSON* topologija u CyberBattleSim *JSON* topologiju, a zatim će se CyberBattleSim *JSON* topologija preslikati u CyberBattleSim okolinu koje je bez izmjena upotrebljiva unutar CyberBattleSim simulacije. CyberBattleSim *JSON* je osmišljen kao nadodan posredni tip podatka kako prilikom preslikavanja funkcionalnosti u kodu ne bi trebalo paziti na sintaksu CyberBattleSim okoline, već se isključivo posvetiti koje svojstvo CCS – a preslikati u koje svojstvo CyberBattleSima. Također, ovaj način implementacije pospješuje čitljivost i preglednost koda. Primjer CyberBattleSim *JSON* – a za jedan čvor vidljiv je u kodu 4.1. Za oba preslikavanja implementirana su kroz *Python* skripte iz razloga dobre podrške *Pythona* kao programskog jezika za *JSON* format.

```
{
  "services": [],
  "vulnerabilities": {
    "PretraziPovijestPreglednika": {
      "description": "Pretrazite␣povijest␣preglednika␣za␣popis␣web␣
         stranica␣kojima␣je␣pristupljeno",
```

```
      "type": "LOCAL",
      "outcome": {
        "LeakedNodesId": ["Web␣stranica"]
      },
      "cost": 1.0,
      "reward_string": ="Povijest␣web␣preglednika␣otkrila␣je␣URL␣
         web␣stranice␣od␣interesa"
    },
  },
  "value": 0,
  "properties": ["Windows11"],
  "firewall": {
    "incoming": [],
    "outgoing": [
      {
        "port": "HTTPS",
        "permission": "ALLOW"
      }
    ]
  },
  "agent_installed": True,
}
```

Kod 4.1: Primjer CyberBattleSim *JSON* – a

4.3.1. Preslikavanje CyberBattleSim *JSON* u CyberBattleSim okolinu

Ova faza uključuje čitanje *JSON* datoteke i kreiranje *Python* objekata koji predstavljaju čvorove, usluge, ranjivosti i druge aspekte mreže. Implementacije ove funkcionalnosti nalazi se u klasi `NodeWriter`, čija funkcija `write_services` je prikazana u kodu 4.2.

```
class NodeWriter:
  def write_services(self, node):
    self.write_line('services=[', 2)

    if 'services' not in node:
      self.write('],')
      return

    for service in node['services']:
      self.write(f'm.ListeningService("{service["name"]}"', 3)
```

```
      if 'allowedCredentials' in service:
        self.write(f',␣
           allowedCredentials=["{service["allowedCredentials"]}"]')
      if 'running' in service:
        self.write(f',␣running={service["running"]}')
      if 'sla_weight' in service:
        self.write(f',␣sla_weight={service["sla_weight"]}')

      self.write(')')
      if service != node['services'][-1]:
        self.write(',')
      self.write_line()
    self.write_line('],', 2)
```

Kod 4.2: Preslikavanje usluga iz CyberBattleSim *JSON* – a u CyberBattleSim *Python* okolinu

Funkcija `write_services` iz *JSON* strukture čita podatke o uslugama i generira odgovarajuće *Python* objekte u CyberBattleSim formatu. Ovaj postupak uključuje provjeru prisutnosti usluga, njihovih atributa poput `allowedCredentials`, `running` i `sla_weight` te njihovo pravilno formatiranje unutar izlaznog *Python* koda.

Sličan postupak primjenjuje se i za preslikavanje ranjivosti. Funkcija `write_vulnerabilities` čita informacije o ranjivostima iz *JSON* – a i generira *Python* objekte koji predstavljaju te ranjivosti u CyberBattleSim okruženju. Ova funkcija omogućava iteraciju kroz sve ranjivosti definirane u *JSON* – u te njihovo pravilno formatiranje i generiranje odgovarajućih *Python* objekata. Proces uključuje provjeru prisutnosti i ispravnosti različitih atributa kao što su opis, vrsta, preduvjeti i stope uspjeha te njihovo preslikavanje u odgovarajuće formate koje koristi CyberBattleSim.

Na sličan način preslikan je i ostatak okoline. Glavna funkcija ove klase `write_nodes_py_env` poziva sve ostale pomoćne funkcije kao što su `write_services` i `write_vulnerabilities` te izvodi samo pretvaranje.

Ove funkcije zajedno omogućavaju pretvorbu cijelog *JSON* opisa mreže u skup *Python* objekata koji se mogu koristiti unutar CyberBattleSim okruženja, čime se omogućava izvođenje simulacija na temelju definiranih mrežnih topologija i scenarija.

4.3.2. Preslikavanje *CCS JSON* u CyberBattleSim *JSON*

Preslikavanje CCS *JSON* formata u CyberBattleSim *JSON* format uključuje više koraka gdje se relevantne informacije iz CCS – a prilagođavaju CyberBattleSimu. Prvi korak u ovom procesu je inicijalizacija i konfiguracija konvertera koji će upravljati cijelim

postupkom. Klasa `ConverterToCyberBattleSim` upravlja preslikavanjem atributa čvorova. Unutar ove klase definirane su metode koje prepoznaju oznake povezane s čvorovima te prema njima dodaju odgovarajuće atribute.

```
class ConverterToCyberBattleSim:
  def __init__(self, input_nodes, output_nodes):
    self.input_nodes = input_nodes
    self.output_nodes = output_nodes

  def add_attributes_by_label(self, node, converted_node):
    labels = set(node['labels'])

    if {'os', 'asset', 'machine'} <= labels:  # 14
      self.add_attributes_by_os_asset_machine(node, converted_node)

    elif {'external_zone', 'physical_zone'} <= labels:  # 15
      converted_node['do_not_convert'] = True

    elif {'exploit', 'file'} <= labels:  # 16
      self.add_attributes_by_exploit_file(node, converted_node)

    def add_attributes_by_exploit_file(self, node, converted_node):
    add_vulnerability_infect_website(node, self.input_nodes,
       self.output_nodes)
    self.add_attributes_by_file(node, converted_node)
```

Kod 4.3: Dio klase `ConverterToCyberBattleSim`

Metoda `add_attributes_by_label` koristi oznake (engl. *labels*) kako bi identificirala i dodala odgovarajuće atribute čvorovima. Na primjer, ako čvor ima oznake koje uključuju `os`, `asset` i `machine`, poziva se metoda `add_attributes_by_os_asset_machine` za dodavanje specifičnih atributa. Ova metoda brine o tome da se svi relevantni podaci pravilno preslikaju u CyberBattleSim *JSON* format.

S druge strane, ako čvor ima oznake `'exploit'` i `'file'`, poziva se metoda `add_attributes_by_exploit_file` koja dodaje ranjivosti čvoru pomoću funkcije `add_vulnerability_infect_website`. Ova metoda omogućuje precizno definiranje iskoristivih ranjivosti unutar mreže, što je ključno za simulaciju napadačkih scenarija.

Vidljivo je da se nad čvorom koji ima oznake `external_zone` i `physical_zone` unutar njegovog rječnika postavlja ključ `do_not_convert` po kojemu se na kraju pretvorbe odbaci sve čvorove koje nije potrebno preslikati u CyberBattleSim okolinu.

Dodatno, metoda `add_vulnerability_send_mail` prikazan kodom 4.4, ilustrira kako se specifične ranjivosti dodaju čvorovima. Ova metoda omogućuje dodavanje ranjivosti slanja elektroničke poruke (`SendMail`) i osigurava da svi relevantni atributi budu ispravno konfigurirani.

```
def add_vulnerability_send_mail(node, output_nodes):
  output_node = output_nodes.get(node['id'])
  vulnerabilities = output_node.setdefault('vulnerabilities', {})
  if vulnerabilities.get('SendMail') is not None:
    print("SendMail vulnerability already exists")
    return

  firewall = output_node.setdefault('firewall', {'incoming': [],
     'outgoing': []})
  firewall['outgoing'].append({'port': 'C-send-mail',
     'permission': 'ALLOW'})

  vulnerabilities['SendMail'] = {
    'description': 'Send prepared email',
    'type': 'LOCAL',
    'precondition': 'ADMINTAG',
    'outcome': {
      'LeakedCredentials': [
        {'node': work_station, 'port': 'C-send-mail',
           'credential': "send-mail-" + work_station}
        for work_station in node['work_stations']
      ]
    },
    'reward_string': 'Mail sent'
  }
```

Kod 4.4: Metoda `add_vulnerability_send_mail`

U metodi `add_vulnerability_send_mail` prvo se provjerava postoji li već ranjivost `SendMail` kako bi se izbjeglo udvostručavanje. Ako ranjivost ne postoji, dodaje se nova ranjivost s atributima kao što su opis, tip (`LOCAL`), preduvjet (`ADMINTAG`) i ishod (`LeakedCredentials`). Također, konfigurira se vatrozid da dopušta odlazni promet na portu `C-send-mail`, čime se osigurava da čvor može slati elektroničku poruku kao dio simulacije.

Ova metoda poziva se isključivo pri kombinaciji oznaka `actor` i `person`, dakle nad čvorom aktera. Akter se u CyberBattleSimu smatra ravnopravnim čvorom po svojstvima s

računalom, pa tako posjeduje na primjer i vatrozid. Makar to nije intuitivno i realno ponašanje, ne postoji drugačiji način da se u CyberBattleSimu preslikaju akteri. No pažljivim konfiguriranjem čvora može se postići da agent pri korištenju okoline postupa očekivano te pronalazi i iskorištavanja ranjivosti aktera, kao i drugih entiteta iz CCS – a koji nisu računala. Vidljivo je da ranjivost po tipu odgovara lokalnoj ranjivosti, što znači da akter prvom mora biti kompromitiran. Nadalje, u preduvjetu ima zadanu razinu privilegija na minimalno razinu administratora, što znači da postoji ranjivost koje će prvom podignuti razinu privilegija, odnosno radnja koje je preduvjet ovoj ranjivosti. Konkretno u ovom slučaju to je ranjivost opisana metodom `add_vulnerability_create_spearphishing_mail_with_link`. Rezultat ove akcije je otkrivanje vjerodajnice za port `C-send-mail` na čvoru ili čvorovima kojima akter radi, no efektivno u simulaciji ovime će biti omogućeno da napadač nakon iskorištavanja ove ranjivosti može kompromitirati računalo na kojem akter radi. Da bi se napadač mogao povezati uz pomoć vjerodajnica na čvor mora imati kompromitirano barem jedno računalo s otvorenim `outgoing` portom za koji je vjerodajnica namijenjena, zbog čega je vatrozid na akter čvoru konfiguriran na taj način.

Ova metoda jasno prikazuje kako se specifične sigurnosne ranjivosti mogu dodati i konfigurirati unutar čvora, prilagođavajući se potrebama simulacije. Kroz ove korake, preslikavanje CCS *JSON* – a u CyberBattleSim *JSON* omogućuje potpuno modeliranje mrežnih elemenata i njihovih sigurnosnih svojstava, čime se omogućava daljnja upotreba u simulacijama.

5. Treniranje autonomnih agenata u CyberBattleSimu

U ovom poglavlju istražuje se proces treniranja autonomnih agenata unutar CyberBattleSim okruženja. Fokus je na odabiru i konfiguraciji agenta, metodologiji treniranja te analizi ponašanja agenata tijekom simulacija, što će biti pokazano kroz *Jupyter* bilježnicu. Definiraju se dostupni algoritmi unutar CyberBattleSima te na posljetku iznose ograničenja te prijedlog potencijalnih poboljšanja.

5.1. Odabir i konfiguracija agenta

CyberBattleSim nudi nekoliko različitih algoritama za treniranje agenata, uključujući *Credential Lookup*, duboko Q-učenje, tablično Q-učenje (engl. *Tabular Q-Learning*) i *Random Policy* [22]. Svaki od ovih algoritama ima svoje prednosti i ograničenja, ovisno o specifičnostima problema i veličini prostora stanja i akcija.

Credential Lookup je algoritam koji se fokusira na iskorištavanje dostupnih vjerodajnica. Ovaj agent preferira korištenje postojećih vjerodajnica prije nego što pokuša druge dostupne akcije. Ova strategija može biti vrlo učinkovita kada su vjerodajnice često dostupne i omogućavaju brzi napredak kroz mrežu.

Prethodno objašnjeno duboko Q-učenje koristi duboku neuronsku mrežu za aproksimaciju Q-vrijednosti. Ovaj pristup može biti vrlo učinkovit u velikim prostorima stanja i akcija zahvaljujući sposobnosti neuronskih mreža da generaliziraju preko sličnih stanja.

Tablično Q-učenje je tradicionalni klasični algoritam Q-učenja koji koristi tabličnu reprezentaciju za spremanje Q-vrijednosti. Ovaj algoritam je prikladniji za manje prostore stanja i akcija.

Random Policy agent nasumično bira valjane akcije u svakom vremenskom koraku. Iako može izgledati primitivno, agent ove vrste može postići iznenađujuće dobre rezultate u scenarijima gdje su važeće akcije ograničene.

Za ovu problematiku, odabran je algoritam dubokog Q-učenja. Duboko Q-učenje

omogućuje efikasno upravljanje velikim prostorima stanja i akcija, što je ključno s obzirom na velik broj čvorova koji se mogu pojaviti u mreži (1019 ulaznih čvorova u početnoj korištenoj CCS topologiji) i veliki broj dostupnih akcija (njih 44 u CCS – u). Osim toga, dubokog Q-učenje može generalizirati preko sličnih stanja, što omogućuje agentu da učinkovitije uči optimalne strategije u složenim mrežnim scenarijima. Ova sposobnost dubokog učenja da prepozna obrasce i prilagodi se promjenama u okruženju dodatno motivira njegov odabir.

5.2. Metodologija treniranja i analiza ponašanja agenata

Za treniranje agenata u CyberBattleSimu korištena je *Jupyter* bilježnica zbog njezine dobre interaktivnosti i mogućnosti lakog ponovnog pokretanja već istreniranog agenta. *Jupyter* bilježnice omogućuju fleksibilnost pri izvođenju eksperimenata, prilagodbu parametara učenja i trenutni pregled rezultata, što je ključno za iterativni proces treniranja i analize ponašanja agenata.

Agent uči optimalnu politiku kroz simulaciju, prolazeći epizodu po epizodu i učeći izravno tijekom interakcije s okolinom za koju nemaju unaprijed definiran model. Algoritam koji se koristi za treniranje je Q-učenje, gdje agent iterativno poboljšava svoje procjene q-vrijednosti, na temelju kojih kasnije može odrediti optimalnu politiku. Proces učenja započinje inicijalizacijom q-vrijednosti (obično postavljene na 0). Tijekom svake epizode, agent koristi epsilon-pohlepnu politiku koja omogućuje balansiranje između istraživanja i iskorištavanja naučenog. U epsilon posto slučajeva, agent odabire akciju nasumično, dok u preostalim slučajevima bira najbolju akciju prema trenutnoj politici. Na početku treniranja, epsilon je postavljen na visoku vrijednost kako bi agent mogao temeljito istraživati okruženje, a kasnije se smanjuje kako bi agent više koristio naučenu politiku.

U kodu 5.1 prikazan je isječak *Jupyter* bilježnica u kojoj se trenira agenta dubokim Q-učenjem.

```
gymid = "CyberBattleMy-v0"
iteration_count = 300
training_episode_count = 400

my_env = gym.make(gymid)

ep = w.EnvironmentBounds.of_identifiers(
  maximum_node_count=120,
  maximum_total_credentials=100,
  identifiers=my_env.identifiers
)
```

```
dqn_learning_run = learner.epsilon_greedy_search(
  cyberbattle_gym_env=my_env,
  environment_properties=ep,
  learner=dqla.DeepQLearnerPolicy(
    ep=ep,
    gamma=0.03,
    replay_memory_size=20000,
    target_update=5,
    batch_size=512,
    learning_rate=0.01
  ),
  episode_count=training_episode_count,
  iteration_count=iteration_count,
  epsilon=0.95,
  epsilon_exponential_decay=4000,
  epsilon_minimum=0.15,
  verbosity=Verbosity.Quiet,
  render=False,
  plot_episodes_length=False,
  title="MyDQL"
)
```

Kod 5.1: Isječak iz *Jupyter* bilježnice `notebook_my.ipynb`

Isječak počinje definiranjem identifikatora okruženja (`gymid`) koje će se koristiti za treniranje agenata. Također se postavljaju parametri za broj iteracija treniranja(`iteration_count`) i broj epizoda treniranja (`training_episode_count`).

Potom se inicijalizira okruženje za treniranje agenata pomoću *OpenAI Gym* sučelja, stvarajući instancu okruženja definiranu identifikatorom `gymid`.

Nadalje se definiraju granice okruženja (`EnvironmentBounds`) koje određuju maksimalni broj čvorova (`maximum_node_count`) i maksimalni broj vjerodajnica (`maximum_total_credentials`). Također, `identifiers` se koriste za definiranje specifičnih identifikatora okruženja.

U posljednjem dijelu isječka koda poziva se funkcija `epsilon_greedy_search` koja pokreće proces treniranja koristeći *epsilon-greedy* strategiju istraživanja. Parametri ove funkcije uključuju [5]:

- `cyberbattle_gym_env=my_env`: Definira okruženje za treniranje. Ovo je instanca okruženja koje je kreirana pomoću *OpenAI Gym* sučelja.

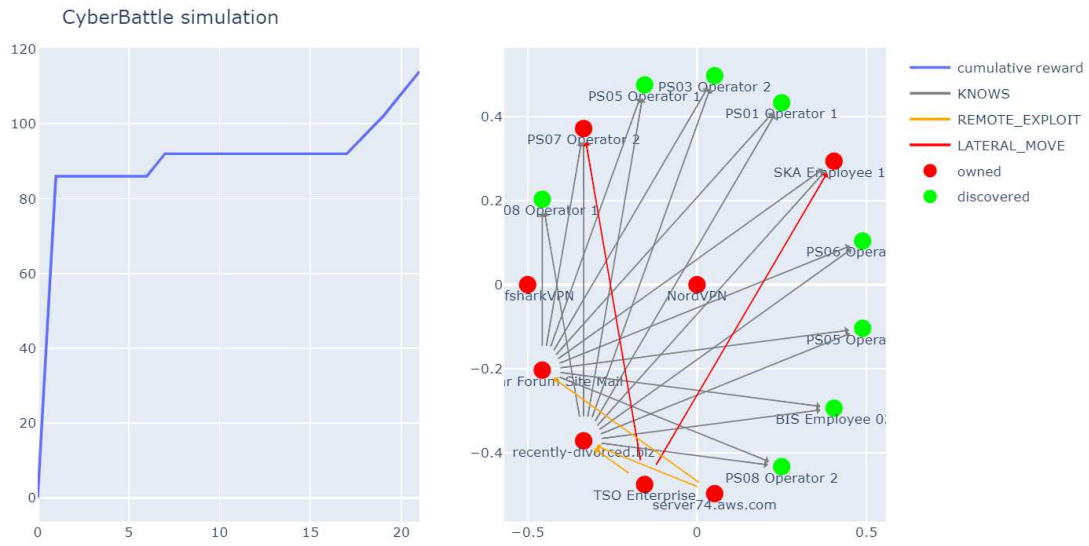
- `environment_properties=ep`: Postavlja granice okruženja. Ove granice definiraju maksimalne dopuštene resurse koje agent može koristiti tijekom simulacije, kao što su maksimalni broj čvorova u mreži i maksimalni broj vjerodajnica koje agent može otkriti. Također, pomažu u kontroliranju složenosti okruženja i osiguravaju da simulacija ostane unutar prihvatljivih parametara.
- `learner=dqla.DeepQLearnerPolicy`: Odabire algoritam dubokog Q-učenja za treniranje agenta. Ovaj algoritam omogućava agentima da nauče optimalne strategije kroz iskustvo stečeno tijekom simulacije.
- `gamma=0.03`: γ diskontni faktor koji određuje koliko agent vrednuje buduće nagrade u odnosu na trenutne nagrade. Niži γ faktor znači da agent više cijeni trenutne nagrade nego buduće, dok viši γ faktor znači da agent više uzima u obzir buduće nagrade pri donošenju odluka.
- `replay_memory_size=20000`: Veličina memorije za ponavljanje iskustava. Memorija omogućuje agentu da pohranjuje svoja prošla iskustva (stanja, akcije, nagrade i sljedeća stanja) i koristi ih za treniranje modela. Ova tehnika pomaže u stabilizaciji treniranja jer omogućuje agentu da uči iz različitih dijelova svog iskustva umjesto da se oslanja samo na najnovije informacije.
- `target_update=5`: Frekvencija ažuriranja ciljne mreže. Ciljna mreža se povremeno ažurira kako bi se uskladila s trenutnom mrežom za treniranje, što pomaže u stabilizaciji treniranja i sprječavanju oscilacija u procesu učenja.
- `batch_size=512`: Veličina grupe za treniranje, odnosno broj iskustava koja se koriste u svakom koraku ažuriranja mreže. Korištenje velikih grupa za treniranje može poboljšati učinkovitost učenja jer omogućuje agentu da uči iz većeg broja primjera u svakom koraku.
- `learning_rate=0.01`: Stopa učenja koja određuje brzinu prilagodbe agenta na nove informacije. Viša stopa učenja znači brže prilagođavanje, ali može dovesti do nestabilnosti, dok niža stopa učenja može omogućiti stabilnije učenje, ali sporije prilagođavanje.
- `episode_count=training_episode_count`: Ukupan broj epizoda treniranja. Epizoda predstavlja jednu cjelovitu sekvencu interakcija agenta s okolinom, od početnog do završnog stanja. Broj epizoda treniranja određuje koliko će puta agent proći kroz cijeli proces učenja, što omogućuje ponavljanje i usavršavanje strategija.
- `iteration_count=iteration_count`: Maksimalni broj iteracija po epizodi. Iteracija predstavlja pojedinačnu akciju agenta unutar jedne epizode. Ovaj parametar ograničava broj koraka koje agent može poduzeti unutar jedne epizode,

čime se kontrolira duljina svake epizode i omogućuje postavljanje jasnih granica za proces učenja.

- `epsilon=0.95`: Početna vrijednost epsilon parametra koji kontrolira omjer istraživanja i iskorištavanja ranjivosti. Epsilon određuje koliko često agent bira nasumične akcije umjesto optimalnih akcija prema trenutnoj politici.
- `epsilon_exponential_decay=4000`: Faktor eksponencijalnog raspadanja epsilon vrijednosti, koji smanjuje epsilon nakon svake epizode kako bi agent postupno prelazio s istraživanja na iskorištavanje naučenog.
- `epsilon_minimum=0.15`: Minimalna vrijednost epsilon parametra. Postavlja donju granicu za epsilon, osiguravajući da agent nikada ne prestane potpuno istraživati okruženje.
- `verbosity=Verbosity.Quiet`: Postavka koja kontrolira količinu ispisa tijekom treniranja.
- `render=False`: Određuje hoće li se vizualizacije prikazivati tijekom treniranja. Onemogućavanje vizualizacija može ubrzati proces treniranja, dok omogućavanje vizualizacija može pomoći u vizualnoj analizi ponašanja agenta.
- `plot_episodes_length=False`: Postavka za iscrtavanje duljine epizoda.
- `title="MyDQL"`: Naslov koji se koristi za identifikaciju eksperimenta.

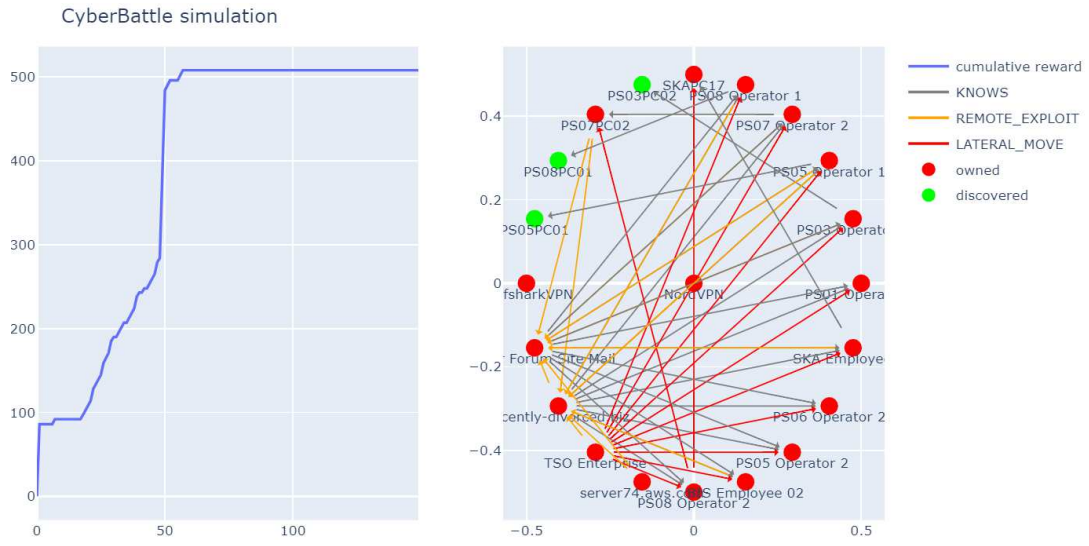
Treniranje na definiranom broju epizoda i iteracija može potrajati više od 80 minuta unatoč korištenju paralelizacije grafičkih procesora. Nakon treniranja agenta, agenta je moguće pokrenuti nad istom okolinom da izvodi akciju po akciju te ispisuje koje radnje je poduzeo. Agenta je moguće upotrijebiti i u drugim okolinama što je pokazano u dokumentaciji CyberBattleSima [19], no svakako najbolje rezultate agent postiže u istoj okolini u kojoj je i treniran. Ispis akcija nije inicijalno podržan unutar CyberBattleSima, no za potrebe ovog rada je implementiran kako bi bilo jasno koje akcije agent provodi u kojem trenutku.

Prilikom pokretanja agenta uočen je ispravan slijed akcija s početno zaraženih čvorova. Pa tako prva uspješna akcija koju agent provodi, odnosno akcija za koju je primljena nagrada je iskorištavanje udaljene ranjivosti `InfectWebsite` nad čvorom koje predstavlja web stranicu u CCS – u. Nakon toga su uočene uspješne akcije `Recon_actor` te kasnije `SpearPhishingMailWithLink`. U CCS – u napadač prvo mora provesti akciju `Recon` nad organizacijom i onda ovisno o tome za koje aktere unutar organizacije napadač dozna iz tog napada, može provesti `Recon` nad njima. U CyberBattleSimu nije izvedivo ovo ograničenje. Stanje u okolini nakon ovih akcija prikazano je slikom 5.1.



Slika 5.1: Izlazni grafovi CyberBattleSima nakon akcije `SpearPhishingMailWithLink`

Akcija `SpearPhishingMailWithLink` kreira *phishing* elektroničku poruku te napadač potom izvodi akciju `SendMail`, kojom šalje elektroničku poruku akteru nad kojim je ova akcija provedena. Nakon toga postanu mu vidljivi svi čvorovi računala na kojima akter radi. Sljedeće vidljivo unutar simulacije je da napadač ponavlja ovaj obrazac kako bi otkrio i čvorove drugih aktera. Nadalje, uspijeva provesti `InspectSystem` te potom `NetworkScan` nad računalima koje je otkrio tijekom provedbe prethodnih obrazaca. Na slici 5.2 vidljivo je konačno stanje nakon pokaznog pokretanja agenta nad okolinom.



Slika 5.2: Izlazni grafovi CyberBattleSima nakon završetka simulacije

Agent se prilikom svakog pokretanja, ovisno o načinu treniranja, može ponašati ponešto drugačije. Pa je tako uočeno da tijekom epizoda treniranja agent uspije provesti napad koji uspije provesti akciju `UploadFile` nad čvorom datoteke koja ima veću vrijednost za napadača (lako uočljivo preko naglog skoka primljene nagrade).

5.3. Ograničenja i potencijalna poboljšanja

Iako CyberBattleSim predstavlja snažan alat za simulaciju kibernetičkih napada, postoje određena ograničenja koja utječu na njegovu primjenu i učinkovitost. U ovom poglavlju raspravljat će se o glavnim ograničenjima trenutne verzije CyberBattleSima i predložiti potencijalna poboljšanja koja bi mogla unaprijediti njegove funkcionalnosti, posebno za preslikavanje okoline iz CCS – a.

Jedno od značajnih ograničenja CyberBattleSima je nedostatak mehanizma dinamičkog preduvjeta za ranjivosti. Kako bi se okolina iz CCS – a mogla kvalitetno prenijeti u CyberBattleSim, mora biti omogućeno napraviti slijed radnji koje zajedno postižu cilj (na primjer kompromitiraju čvor) kao preduvjet jedna drugoj. Trenutno u CyberBattleSimu postoji mehanizam podizanja privilegija, koji je upotrebljiv isključivo nakon što je čvor kompromitiran i limitiran na četiri stupnja. Posebno bi bilo korisno da funkcionalnost preduvjeta u logičkom izrazu može sadržavati uvjet na drugi čvor u mreži, što je u CCS – u čest slučaj. Primjer akcije koja zahtjeva ovu logiku je preduvjet da napadač prvo mora provesti akciju `InfectWebsite` i tek onda može napraviti elektroničku poruku s zaraženom stranicom kao prilogom.

Također bilo bi korisno unutar svojstva čvora imati mogućnost definiranja verzije. U stvarnim mrežama kao i u CCS – u, čvorovi često prolaze kroz različite faze konfiguracija i ažuriranja, što može utjecati na njihove ranjivosti i sigurnosne postavke. Trenutna verzija CyberBattleSima ne omogućuje modeliranje različitih verzija svojstava čvorova, što ograničava realističnost simulacija.

Poboljšanje bi također bilo omogućiti spremanje istreniranog agenta te njegovo ponovno učitavanje po potrebi. Ova funkcionalnost omogućila bi korisnicima da nastave testiranje agenta nakon pauze ili ponovnog pokretanja CyberBattleSima, čime bi se uštedjelo vrijeme i računalni resursi potrebni za ponovno treniranje od početka.

Postoje i specifična ograničenja u vezi s kapacitetom simulacije, kao što su maksimalni broj čvorova i vjerodajnica. Trenutna postavka u CyberBattleSimu omogućuje maksimalno 100 čvorova i 1000 vjerodajnica. Ovo je vrlo značajno ograničenje, posebno kada se radi o simulaciji složenih mreža koje imaju stotine ili tisuće čvorova. Na primjer, najosnovnija topologija u nekim stvarnim scenarijima može imati 430 čvorova, što znatno premašuje trenutna ograničenja CyberBattleSima. Kada se pokušava zaobići ovo ograničenje mijenjanjem vrijednosti u izvornom kodu na veće vrijednosti, može doći do grešaka poput `MemoryError`. Ova greška ukazuje na to da trenutačna implementacija nije optimizirana za rad s velikim mrežama te da su potrebne optimizacije kako bi se poboljšala skalabilnost i učinkovitost. Zbog ovog ograničenja u 1019 različitih čvorova u ulaznoj topologiji CCS – a preslikano je 58 čvorova, pritom su zadržani čvorovi koji potencijalno mogu sudjelovati u

phishing napadu, koji je proveden kao dokaz koncepta.

Još jedno ograničenje pretvorbe okoline iz CCS – a u CyberBattleSim je potreba da se 44 vrste različitih napada i ishoda napada dostupnih u CCS – u mora preslikati u 8 ishoda CyberBattleSima. Prethodno navedena poboljšanja poput implementacije mehanizma dinamičkih preduvjeta na čvor za pravilnu funkcionalnost morala bi dopuniti i moguće ishode akcija, no usprkos tome postoji i potencijal za dopunom postojećih ishoda akcija kao što je ishod `CustomerData` koji ne donosi dodatnu nagradu napadaču, a mogao bi.

Nadalje CyberBattleSim ne zna za vremensku i prostornu domenu unutar simulacije, što ograničava prijenos pune funkcionalnosti iz CCS – a unutar kojeg svaka radnja ima pripadno trajanje i može imati za preduvjet da se lokacija aktera (na primjer napadača) poklapa s lokacijom čvora. Također postoji i mogućnost preduvjeta akcije s obzirom na vještina aktera kojima se provode, no pošto CyberBattleSim emulira napadača za pretpostaviti je da sofisticirani napadači današnjice imaju većinu vještina potrebnih za provedbu napada. Prilikom preslikavanja funkcionalnosti vezane uz prostornu domenu nisu preslikane, dok vremensku domenu je moguće nadodati svakoj od provedenih akcija, ako znamo koliko traje (definirano u CCS – u) te koje joj akcije prethode (izlaz CyberBattleSima).

Unatoč ovim ograničenjima, dio okoline koji je uspješno preslikan iz CCS – a u CyberBattleSim te korištenje istreniranog agenta nad njom, dokazuje da je provedeni napad moguće iskoristiti i unutar samog CCS – a. Razlog tome je što slijed akcija koje agent obavlja u CyberBattleSimu može biti uspješno obavljen i u CCS – u ako je okolina pravilno preslikana, uz potrebu smještaja u vremenu.

Usporedba agenata treniranih u CyberBattleSimu sa stvarnim napadačima otkriva nekoliko ključnih razlika, ali i sličnosti koje se mogu iskoristiti za poboljšanje strategija kibernetičke sigurnosti. Agenti trenirani metodama dubokog podržanog učenja imaju potencijal da otkriju ranjivosti i pronađu načine proboja u mrežama koje ljudski napadači možda ne bi primijetili, osobito u složenim mrežama s više stotina čvorova. Ovi agenti mogu sistematski istraživati mrežna okruženja, testirati različite kombinacije napada i koristiti povratne informacije kako bi kontinuirano poboljšavali svoje strategije. Ovaj pristup može dovesti do otkrivanja ne intuitivnih puteva napada i novih ranjivosti koje su promakle ljudima koji emuliraju napadača ili stvarnom napadaču.

S druge strane, stvarni napadači često imaju specifične ciljeve i fokusiraju se na minimalizaciju rizika otkrivanja. Ljudski napadači provode napade s određenom svrhom, izbjegavajući nepotrebne akcije koje bi mogle povećati šanse da budu otkriveni. Oni koriste svoje iskustvo, intuiciju i znanje o specifičnim sustavima kako bi ciljano i efikasno proveli napad, dok trenirani agenti često slijede optimizacijske algoritme koji ne uzimaju u obzir aspekt skrivanja svojih aktivnosti.

Zbog tih razlika, najbolji pristup može biti kombinacija obje vrste, ljudskih napadača i treniranih agenata, u hibridnom modelu. Ovaj model bi koristio agente trenirane u CyberBattleSimu za prepoznavanje mogućih puteva napada i proboja u mrežama, dok bi ljudski napadači ili ljudi koji emuliraju napadača filtrirali i procijenili te puteve u kontekstu specifičnih ciljeva napada i rizika od otkrivanja. Agenti bi mogli identificirati širok spektar potencijalnih ranjivosti i strategija napada, dok bi ljudi koji emuliraju napadača procijenili koje od tih strategija najbolje odgovaraju njihovim ciljevima i koje su najsigurnije za implementaciju bez otkrivanja, kako bi bili što bliži pravim napadačima.

6. Zaključak

U današnjem digitalnom dobu kibernetički napadi postaju sve sofisticiraniji i češći, predstavljajući značajnu prijetnju za sigurnost informatičkih sustava diljem svijeta. Ovaj rad je pokazao kako se CyberBattleSim može koristiti za treniranje autonomnih agenata koji simuliraju različite napade unutar definiranih okruženja. Kroz detaljan pregled i razvoj alata za preslikavanje CCS okoline u CyberBattleSim okolinu, omogućeno je stvaranje realističnih scenarija za treniranje.

Podržano učenje, kao grana strojnog učenja, omogućuje agentima da kroz interakciju s okolinom uče optimalne strategije. U radu je prikazano kako agenti, trenirani uz pomoć podržanog učenja, mogu identificirati sigurnosne propuste unutar simuliranih mreža i prilagoditi svoje strategije za učinkovito provođenje napada. Unatoč ograničenjima kao što su broj čvorova u mreži i nedostatak detalja okoline unutar CyberBattleSim simulacije, naučeni agenti pružaju vrijedne uvide u moguća ponašanja napadača.

Konačno, eksperimentalni rezultati pokazuju da su agenti sposobni prepoznati i iskoristiti ranjivosti u simuliranim mrežama, što je ključno za automatiziranu emulaciju napadača u okolini kao što je CCS. Emulacija napadača omogućava potencijalnim žrtvama da se bolje pripreme za obranu od kibernetičkih napada, odnosno doprinosi pripremljenosti na stvarne napade.

Konačno, rad naglašava mogućnost daljnjeg razvoja i poboljšanja CyberBattleSima kako bi se povećala razina detalja okoline, a time i vjernost treniranih agenata u simulatoru.

LITERATURA

[1] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, stranice 1–3, 2024.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, i Wojciech Zaremba. Openai gym, 2016.

[3] Thomas Carr. Deterministic vs. stochastic policies in reinforcement learning. *Baeldung*, 2024. dostupno na `https://www.baeldung.com/cs/rl-deterministic-vs-stochastic-policies` [pristup 10.6.2024.].

[4] Inc. Docker. Docker documentation. `https://docs.docker.com/`, 2024. [pristup 13.6.2024.].

[5] Jonathan Esteban. *Simulating Network Lateral Movements through the CyberBattleSim Web Platform*. Doktorska disertacija, Massachusetts Institute of Technology, 2022.

[6] Mohamed C Ghanem i Thomas M Chen. Reinforcement learning for intelligent penetration testing. U *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, stranice 185–192. IEEE, 2018.

[7] Scott R Granter, Andrew H Beck, i David J Papke Jr. Alphago, deep learning, and the future of the human microscopist. *Archives of pathology & laboratory medicine*, 141 (5):619–621, 2017.

[8] K. Grubešić. Izgradnja složenog kibernetičkog poligona za vježbe napada i obrane. Magistarski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2022. URL `https://urn.nsk.hr/urn:nbn:hr:168:341208`. Diplomski rad.

[9] Thomas Kunz, Christian Fisher, James La Novara-Gsell, Christopher Nguyen, i Li Li. A multiagent cyberbattlesim for rl cyber operation agents. U *2022 International*

Conference on Computational Science and Computational Intelligence (CSCI), stranice 897–903. IEEE, 2022.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, i Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9(1):381–386, 2020.

[12] Microsoft. Gamifying machine learning for stronger security and ai models, 2021. URL `https://www.microsoft.com/en-us/security/blog/2021/04/08/gamifying-machine-learning-for-stronger-security-and-ai-models/`. [pristup 5.6.2024.].

[13] Microsoft. Visual studio code. `https://code.visualstudio.com/`, 2024. [pristup 13.6.2024.].

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[15] Steve Morgan. Cybercrime to cost the world $9.5 trillion usd annually in 2024. *Cybersecurity Ventures*, 2023. dostupno na `https://cybersecurityventures.com/cybercrime-to-cost-the-world-9-trillion-annually-in-2024/` [pristup 8.6.2024.].

[16] Thanh Thi Nguyen i Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3779–3795, 2021.

[17] Xue Qiu, Shuguang Wang, Qiong Jia, Chunhe Xia, i Qingxin Xia. An automated method of penetration testing. U *2014 IEEE Computers, Communications and IT Applications Conference*, stranice 211–216. IEEE, 2014.

[18] Richard S Sutton i Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2nd izdanju, 2018.

[19] Microsoft Defender Research Team. Cyberbattlesim. `https://github.com/microsoft/cyberbattlesim`,

2021. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.

[20] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, i Omar G. Younis. Gymnasium, Ožujak 2023. URL `https://zenodo.org/record/8127025`.

[21] Utilis. Cyber conflict simulator (ccs), 2021. URL `https://ccs.utilis.biz/`. [pristup 14.6.2024.].

[22] Erich Walter, Kimberly Ferguson-Walter, i Ahmad Ridley. Incorporating deception into cyberbattlesim for autonomous defense. *arXiv preprint arXiv:2108.13980*, 2021.

[23] Dongbin Zhao, Haitao Wang, Kun Shao, i Yuanheng Zhu. Deep reinforcement learning with experience replay based on sarsa. U *2016 IEEE symposium series on computational intelligence (SSCI)*, stranice 1–6. IEEE, 2016.

Treniranje agenata za provođenje napada u simulatoru CCS korištenjem okruženja CyberBattleSim

Sažetak

Ovaj rad istražuje treniranje agenata za provođenje napada u *Cyber Conflict Simulatoru* (CCS) korištenjem okruženja *CyberBattleSim*. Započinje s pružanjem pregleda podržanog učenja, uključujući osnovne koncepte, algoritme i njihovu primjenu u kibernetičkoj sigurnosti. Razmatraju se arhitektura, komponente te postupak postavljanja i konfiguracije simulacija u *CyberBattleSimu*. Poseban naglasak stavljen je na razvoj alata za preslikavanje okruženja iz CCS – a u CyberBattleSim, ističući ključne razlike između ta dva sustava te tehničke detalje implementacije. Na kraju, dan je pregled treniranja autonomnih agenata u *CyberBattleSimu*, uključujući odabir i konfiguraciju agenata, metodologiju treniranja te analizu njihovog ponašanja s posebnim naglaskom na ograničenja i potencijalna poboljšanja.

Ključne riječi: Podržano učenje, CyberBattleSim, Cyber Conflict Simulator, Automatizacija napada

Training agents for attack execution in CCS simulator using CyberBattleSim environment

Abstract

This thesis investigates the training of agents for conducting attacks in the *Cyber Conflict Simulator* (CCS) using the *CyberBattleSim* environment. It begins with an overview of reinforcement learning, including basic concepts, algorithms, and their application in cybersecurity. The architecture, components, and the process of setting up and configuring simulations in CyberBattleSim are examined. Special emphasis is placed on the development of a tool for mapping environments from CCS to *CyberBattleSim*, highlighting the key differences between the two systems and the technical details of the implementation. Finally, a review is given of training autonomous agents in *CyberBattleSim*, including the selection and configuration of agents, training methodology, and the analysis of their behavior with a focus on limitations and potential improvements.

Keywords: Reinforcement Learning, CyberBattleSim, Cyber Conflict Simulator, Attack Automation