



Penetration Testing Report

ANALISI E SPERIMENTAZIONE DI PIATTAFORME
PER CYBER BATTLE E NETWORK ATTACK SIMULATION:
PENETRATION TESTING SIMULATO DI CUSTOM NETWORK

Giorgio Colella (0522501752)
A.A. 2024/2025



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Indice

1 Executive Summary	1
2 Engagement Highlights	1
2.1 Configurazione e Strumenti Utilizzati	1
2.2 Regole di Ingaggio	2
2.3 Obiettivi del Penetration Testing	2
2.4 Fasi della metodologia	3
3 Vulnerability Report	4
4 Remediation Report	5
5 Findings Summary	6
6 Detailed Summary	8
7 Riferimenti	17
8 Appendici	18
.1 CyberBattleSim (Microsoft)	18
.1.1 Funzionamento e architettura	18
.1.2 Implementazione	19
.2 CybORG (CAGE Challenge)	25
.2.1 Architettura ibrida: simulazione ed emulazione	25
.2.2 Implementazione	26
.3 Network Attack Simulator (NASim)	31
.3.1 Funzionamento e architettura	31
.3.2 Implementazione	32
.4 Infection Monkey	34
.4.1 Architettura	34
.4.2 Funzionamento Pratico di Infection Monkey	35
.5 Tentativi e Processo di Apprendimento dei Tool	38
.5.1 CyberBattleSim	38
.5.2 CybORG	39
.5.3 NetworkAttackSimulator	40
.6 Confronto Trai Tool	40
.7 Installazione tool	41
.7.1 CyberBattleSim	41
.7.2 CybORG Challenge 4	42
.7.3 NetworkAttackSimulation (Nasim)	43
.7.4 Infection Monkey	44
.8 Script Forniti	46

1 Executive Summary

In questo documento viene presentata l'analisi di sicurezza di una Custom Network e cioè di una rete fittizia la quale viene simulata usando lo strumento CyberBattleSim. Quest'ultimo è simulatore ad alto livello sviluppato da Microsoft che rappresenta la rete aziendale come un grafo con nodi vulnerabili ed utilizza agenti di Reinforcement Learning per condurre sia la difesa che l'attacco della rete. Un agente di reinforcement learning è un'entità che interagisce con un ambiente, impara tramite ricompense e ottimizza le sue azioni per massimizzare un obiettivo nel tempo. In questo caso, l'ambiente permette di definire nodi che rappresentano gli host ed hanno proprietà come sistema operativo ma anche ruoli utente (con tanto di credenziali) e servizi offerti i quali presentano anche vulnerabilità customizzabili. Sono presenti diversi tipi di attacchi che permettono di ottenere credenziali, accesso alla macchina, privilege escalation e così via. Gli agenti d'attacco forniti sono, per esempio, DQN (Deep Q-Learning Network) e la versione tabulare di Q-Learning i quali sono algoritmi di RL e sfruttano formule matematiche diverse per l'apprendimento. Questi agenti sono parametrizzabili e quindi adattabili a diverse situazioni e possono anche essere allenati. Esistono poi anche agenti di difesa stocastici che viaggiano all'interno della rete alla ricerca di macchine compromesse e, appena ne individuano una, cercano di ripristinarla rimuovendo il controllo che l'attaccante ha ottenuto su di essa. L'ambiente risulta quindi astratto perché non solo la topologia di rete non è dinamica ma tutti i concetti presenti sono definibili fino ad un certo livello di dettaglio: gli agenti si muovono di nodo in nodo e se rispettano alcuni prerequisiti effettuando le azioni giuste riescono ad ottenere l'accesso al nodo ma non vengono effettivamente eseguiti comandi complessi come quelli, per esempio, del famoso tool NMAP (strumento open source per la scansione di reti, utilizzato per individuare host attivi, porte aperte e servizi in esecuzione). Inoltre gli host non sono effettivamente host simulati ma sono nodi che presentano degli attributi tipici di un host che viene quindi così astratto. Questa da un lato è una limitazione e dall'altro è uno strumento che permette di: effettuare test rapidi focalizzandosi sulla strategie di attacco più che sulla tecnica ("mi concentro su firewall?", "faccio privilege escalation o lateral movement?" invece di "uso questo tool oppure quest'altro?") e di avere un ambiente controllato e riproducibile dove poter allenare agenti in vista di task più complesse. L'obiettivo di questo documento è quello di comprendere le potenzialità di questo tool e se esse possono essere utili per un pentester. Inoltre, il tool verrà anche confrontato con altri che offrono funzionalità simili. Nel documento viene analizzata una rete fittizia dalla quale sono state estratte diverse informazioni come host attivi, servizi erogati e vulnerabilità. Per queste ultime poi sono state proposte delle mitigazioni. L'agente è riuscito a scoprire ed attaccare 11 dei 15 nodi della rete nei diversi test ed ha scoperto proprietà, servizi e vulnerabilità come RDPBrute di livello Alto e WebRCE di livello Critico. **Và comunque precisato che le vulnerabilità sono fittizie poichè simulate ma cercano comunque di rifarsi al mondo reale.**

2 Engagement Highlights

2.1 Configurazione e Strumenti Utilizzati

Le analisi effettuate durante il Penetration Testing hanno richiesto i seguenti strumenti ed ambienti:

- **Kali Linux**: utilizzato per l'esecuzione dei simulatori CyberBattleSim, CybORG e NetworkAttackSimulator;
- **Ubuntu**: utilizzato per l'esecuzione dello strumenti di pentesting automatizzato Infection Monkey;
- **CyberBattleSim**: simulatore di Cyber Battle sviluppato da Microsoft;
- **CybORG Challenge 4**: ambiente Gym di simulazione, realistico e nato per diverse challenge di network attacking and defending;
- **Network Attack Simulator (Nasim)**: ambiente Gym di simulazione realistico ma semplice e leggero;
- **Jupyter**: ambiente interattivo open source che permette di scrivere ed eseguire codice;
- **Overleaf**: editor online per scrivere documenti LaTeX, usato soprattutto per articoli scientifici, tesi e report tecnici.

2.2 Regole di Ingaggio

Per quanto riguarda le Regole di Ingaggio:

- per gli ambienti, essi sono simulati quindi tutto è permesso considerando che non ci sono rischi essendo gli ambienti non reali e ripristinabili;
- per gli ambienti reali, tutto ciò che non fa parte della Nat Network, in cui sono presenti le macchine virtuali, va escluso da attività di testing;
- verrà utilizzato il framework FGPT (Framework Generale per il Penetration Testing);
- il testing sarà Black Box perchè è effettuato dagli agenti RL che partono senza avere alcuna conoscenza dell'ambiente in cui si trovano;
- non si rende necessario NDA (Non-Disclosure Agreement) essendo gli ambienti simulati ed i dati completamente fintizi;

2.3 Obiettivi del Penetration Testing

L'obiettivo è quello di determinare la sicurezza di una rete fittizia in modo da testare l'efficacia degli ambienti simulati nel rappresentare il mondo reale. Questo permetterà di capire quanto sia utile per un pentester avvalersi anche di ambienti simulati e quanto ancora questi simulatori debbano evolvere.



Figura 1: Screenshot del sistema operativo Kali emulato tramite Oracle VirtualBox

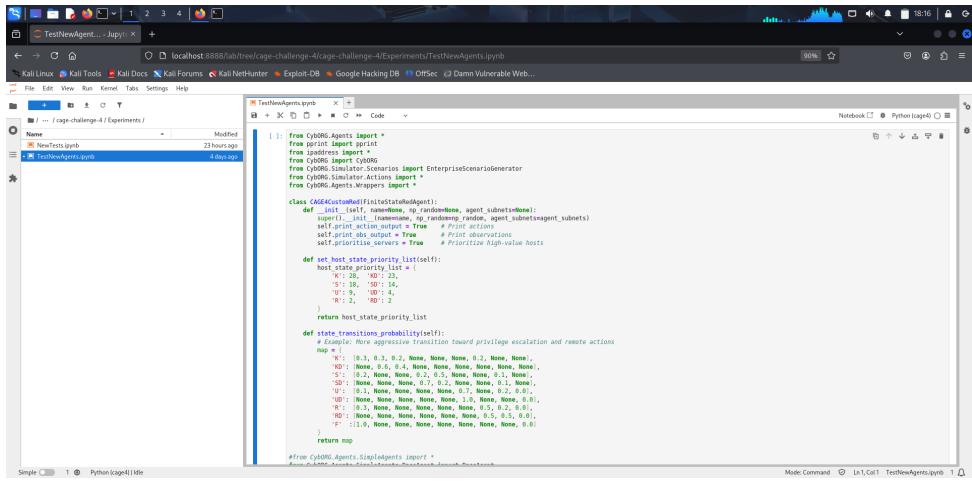


Figura 2: Screenshot dell’ambiente interattivo di sviluppo Jupyter

2.4 Fasi della metodologia

La metodologia adottata per l’attività di penetration testing prevede le seguenti fasi:

- Target Scoping:** definire i confini del test modellando una rete aziendale fittizia in CyberBattleSim e individuando gli asset critici (host, servizi e vulnerabilità simulate).
- Information Gathering:** raccolta di dati dalla rete per cercare di capire la sua infrastruttura, azione compiuta dall’agente.outcome).
- Target Discovery:** identificare gli host attivi e raggiungibili tramite scanning.
- Enumerating Target:** scoprire servizi e versioni sui nodi identificati.
- Vulnerability Mapping:** individuare ciascuna vulnerabilità nota ed associarle ad i diversi host attivi.
- Target Exploitation:** verificare l’effettivo sfruttamento di ciascuna vulnerabilità e l’impatto sul sistema.

7. **Post-Exploitation:** valutare possibili movimenti laterali, privilege escalation e persistenza dopo l'exploit iniziale.
8. **Reporting:** produrre il documento tecnico finale, completo di tabelle, grafici e raccomandazioni.



Figura 3: Fasi della metodologia

Il presente documento contiene la fase di Reporting mentre il Penetration Testing Narrative conterrà la documentazione dello svolgimento di tutte le altre fasi.

3 Vulnerability Report

Le vulnerabilità individuate dall'agente potrebbero essere categorizzate in:

- **Credential Issues Sensitive Information Disclosure** (gestione inadeguata delle credenziali come password deboli, hardcoded o accessibili in file non protetti e esposizione involontaria di dati sensibili) : RDPBruteCreds, SMBcreds, WebAdminCreds, WeakAdmin, GitSMBcreds, App1DBCreds, App2FileCreds, MailLeakCreds, MgmtCreds, SQLiCreds, GitHistory;
- **Remote Access Misconfiguration e Remote Code Execution** (configurazioni errate di servizi remoti che consentono accesso non autorizzato o privilegiano troppo) : WeakAdmin, RDPBrute, SMBWeakShare, MgmtExploit, WebExploit, App2APIExploit,;
- **Privilege Escalation** (escalation da utente normale a privilegi elevati): App1PrivEsc, App2PrivEsc, WeakAdmin;
- **Data Breach** (enorme furto di dati): SQLiDataBreach;

Sono state individuate **25 vulnerabilità** da parte dell'agente divise nelle seguenti gravità:

- **Gravità Critica (2)**: RDPBrute e WebAdminCreds;
- **Gravità Alta (14)**: SMBcreds, WebExploit, WebAdminCreds, WeakAdmin, BackupSQLCreds, Web3SSRF, Web3BackupCreds, App1PrivEsc, App2APIExploit, App2PrivEsc, MgmtExploit, MgmtCreds, SQLi, SQLiCreds e SQLiDataBreach;

- **Gravità Media (6)**: App2FileCreds, App1DBCreds, MailLeakCreds, GitSMB-Creds, SMBWeakShare e RDPBruteCreds;
- **Gravità bassa (3)**: MailLeak, GitHistory e BackupExploit.

4 Remediation Report

L'analisi condotta tramite simulazione ha messo in evidenza diverse vulnerabilità critiche nei sistemi esposti. Tali vulnerabilità, identificate tramite servizi come SSH, SMB, SQL, HTTP e RDP, compromettono seriamente la sicurezza dell'intera infrastruttura. Si raccomanda di applicare le seguenti contromisure tecniche per mitigare i rischi:

- **Hardening dei servizi esposti via rete**, in particolare SMB, SQL e HTTP, disabilitando funzionalità inutili e aggiornando le configurazioni di sicurezza con policy di accesso minime e regole firewall dedicate.
- **Protezione dei file di backup e delle credenziali**, rilevanti in vulnerabilità come *BackupSQLCreds*, *Web3BackupCreds* e *MgmtCreds*. È fondamentale criptare tutti i backup e rimuovere file contenenti credenziali plaintext o storici Git esposti.
- **Applicazione di filtri sugli input API e SQL**, come prevenzione contro *App2APIExploit* e *SQLi*. È essenziale usare ORM sicuri, prepared statements e sanitizzazione approfondita degli input lato server.
- **Segmentazione della rete e accessi privilegiati**, per isolare i server compromessi e ridurre il rischio di movimento laterale. Le vulnerabilità *App2PrivEsc*, *App1PrivEsc* e *MgmtExploit* suggeriscono escalation possibili in ambienti insufficientemente compartmentalizzati.
- **Limitazione dell'esposizione SMTP e auditing delle comunicazioni**, per evitare leakage informativi come nel caso di *MailLeak* e *MailLeakCreds*. È opportuno introdurre cifratura TLS obbligatoria e filtri su parole chiave sensibili nei log e nei messaggi.
- **Revisione delle impostazioni del Domain Controller**, dove la presenza combinata di *WeakAdmin*, *RDPBrute* e *RDPBruteCreds* ha portato a un completo takeover. È necessario limitare le connessioni RDP, adottare autenticazione multi-fattore e bloccare gli account dopo ripetuti tentativi errati.
- **Monitoraggio continuo e detection automatica**, con strumenti di anomaly detection e SIEM, per individuare pattern simili a quelli emulati durante la simulazione. L'iniezione SQL, l'accesso a file di backup o Git e l'abuso dei protocolli SMB e RDP devono generare alert tempestivi.
- **Ripensamento delle policy di gestione delle credenziali**, con introduzione di vault centralizzati, rotazione periodica e audit automatico di chiavi e password esposte (come nei repository Git o nelle directory di backup).
- **Aggiornamento continuo del software**, inclusi OS (Linux e Windows) e applicazioni custom (come server API e database), per correggere vulnerabilità conosciute sfruttate in attacchi simulati come *Web3SSRF* e *SQLiDataBreach*.

- **Formazione del personale IT e DevSecOps**, al fine di diffondere consapevolezza sui rischi legati a repository non protetti, configurazioni errate e gestione inadeguata delle chiavi di accesso.

Nel complesso, la simulazione ha dimostrato che una rete apparentemente segmentata può comunque essere compromessa da una concatenazione di vulnerabilità elementari, sfruttate in modo coerente. Si consiglia pertanto di adottare un approccio proattivo alla sicurezza, integrando test automatici, best practice di sviluppo sicuro e monitoraggio centralizzato.

5 Findings Summary

Le vulnerabilità riscontrate sono classificate in base alla gravità, suddivise su una scala con quattro livelli di gravità:

- **Critical**: Rischio critico, vulnerabilità molto grave per il sistema.

$$(9.0 \leq \text{CVSS}^3 < 10)$$

- **High**: Rischio elevato, vulnerabilità grave per il sistema.

$$(7.0 \leq \text{CVSS}^3 < 9.0)$$

- **Medium**/: Rischio medio, vulnerabilità abbastanza grave per il sistema.

$$(4.0 \leq \text{CVSS}^3 < 7.0)$$

- **Low**: Rischio basso, vulnerabilità non particolarmente grave per il sistema.

$$(0.1 \leq \text{CVSS}^3 < 4.0)$$

Vulnerabilità Per Gravità

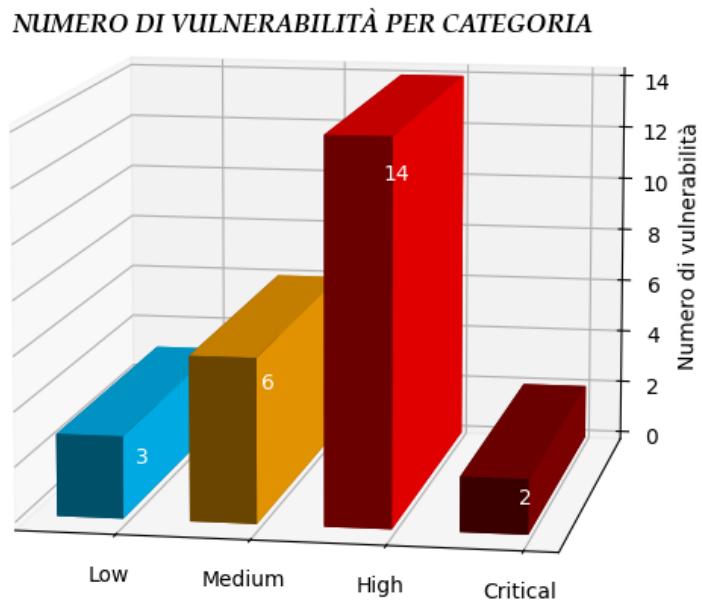


Figura 4: Numero di vulnerabilità per categoria

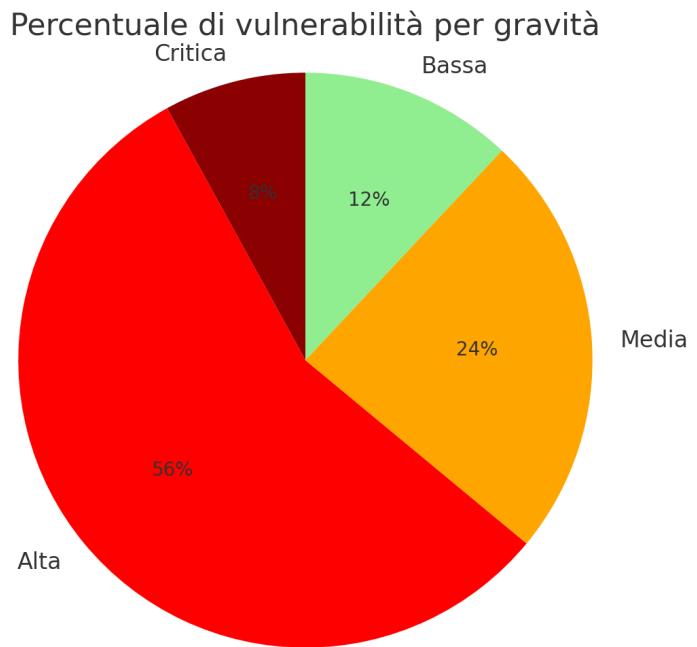


Figura 5: Numero di vulnerabilità per categoria

Vulnerabilità per Host

Nodo	OS	Servizi aperti	Vulnerabilità
ws_entry	Linux		WeakAdmin
web1	Linux	HTTP, GIT	WebExploit, WebAdminCreds, GitHistory, GitSMBCCreds
file1	Windows	SMB	SMBWeakShare, SMBCCreds
app2	Linux	SSH	App2APIExploit, App2FileCreds, App2PrivEsc
app1	Linux	SSH	App1PrivEsc, App1DBCCreds
backup1	Windows	SMB	BackupExploit, BackupSQLCCreds
sql1	Windows	SQL	SQLi, SQLiCreds, SQLiDataBreach
mail1	Linux	SMTP	MailLeak, MailLeakCreds
dc1	Windows	RDP	RDPBrute, RDPBruteCreds, WeakAdmin
mgmt1	Linux	SSH	MgmtExploit, MgmtCreds
web3	Windows	HTTP	Web3SSRF, Web3BackupCreds

6 Detailed Summary

RDPBrute	CVE CVE-2019-0708
Critical	
Descrizione:	Attacco brute-force sul servizio RDP per ottenere accesso non autorizzato.
Impatto:	L'attaccante può ottenere accesso remoto completo al nodo tramite forza bruta sul servizio RDP.
Soluzione:	Abilitare l'autenticazione a due fattori, blocco degli IP e controlli su RDP.
Metodo di detection:	Tentativi ripetuti di autenticazione automatizzati nel simulatore.
Nodi coinvolti:	'dc1'

Tabella 2: Scheda vulnerabilità RDPBrute

RDPBruteCreds	CVE Nessuna CVE specifica identificata
Medium	
Descrizione:	Uso di credenziali RDP comuni note per ottenere accesso remoto.
Impatto:	Compromissione delle credenziali dell'amministratore via file di configurazione o brute-force
Soluzione:	Rimuovere credenziali statiche e gestirle in modo sicuro (come vault)
Metodo di detection:	Accesso a file non protetti contenenti password in chiaro.
Nodi coinvolti:	‘dc1’

Tabella 3: Scheda vulnerabilità RDPBruteCreds

MailLeak	CVE Nessuna CVE specifica identificata
LOW	
Descrizione:	Intercettazione di e-mail confidenziali all'interno dell'host
Impatto:	Accesso a messaggi email interni tramite vulnerabilità nel servizio SMTP.
Soluzione:	Configurare correttamente SMTP con accessi autenticati e log delle attività.
Metodo di detection:	Lettura simulata di comunicazioni SMTP in chiaro o da log.
Nodi coinvolti:	‘mail1’

Tabella 4: Scheda vulnerabilità MailLeak

SMBWeakShare	CVE CVE-1999-0519
Medium	
Descrizione:	Accesso non autorizzato a share SMB non protetti.
Impatto:	Accesso a file riservati tramite condivisioni SMB pubbliche non protette
Soluzione:	Disabilitare SMBv1, impostare ACL rigorose e proteggere le cartelle condivise.
Metodo di detection:	Simulazione tramite CyberBattleSim.
Nodi coinvolti:	‘file1’

Tabella 5: Scheda vulnerabilità SMBWeakShare

SMBcreds	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Credenziali SMB trovate o brute-forzate per accedere alla macchina.
Impatto:	L'attaccante ottiene credenziali sensibili tramite file SMB o condivisioni mal configurate.
Soluzione:	Cifrare i file contenenti segreti e limitare accessi.
Metodo di detection:	Download e lettura file tramite SMB simulata.
Nodi coinvolti:	'file1'

Tabella 6: Scheda vulnerabilità SMBcreds

WebExploit	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Sfruttamento di vulnerabilità comuni su server web per ottenere accesso.
Impatto:	Esecuzione di codice remoto o esposizione di file tramite il servizio web.
Soluzione:	Sanitizzazione input, aggiornamenti framework e WAF.
Metodo di detection:	Invio di payload simulati che bypassano filtri.
Nodi coinvolti:	'web1'

Tabella 7: Scheda vulnerabilità WebExploit

WebAdminCreds	CVE Nessuna CVE specifica identificata Critical
Descrizione:	Recupero di credenziali admin da configurazioni esposte o file.
Impatto:	Accesso non autorizzato tramite file visibili contenenti credenziali di amministratore.
Soluzione:	Evitare la memorizzazione in chiaro di credenziali su webserver.
Metodo di detection:	Analisi file di configurazione in directory web.
Nodi coinvolti:	'web1'

Tabella 8: Scheda vulnerabilità WebAdminCreds

GitSMBcreds	CVE Nessuna CVE specifica identificata
Medium	
Descrizione:	Credenziali SMB estratte da repository Git esposti.
Impatto:	Le credenziali SMB pubblicate in repository possono essere utilizzate per movimenti laterali.
Soluzione:	Audit dei repository e revoca delle credenziali esposte.
Metodo di detection:	Parsing automatico dei repository tramite CyberBattle-Sim.
Nodi coinvolti:	‘web1‘

Tabella 9: Scheda vulnerabilità GitSMBcreds

GitHistory	CVE Nessuna CVE specifica identificata
LOW	
Descrizione:	Informazioni sensibili recuperate dalla cronologia Git.
Impatto:	Espone cronologie con codice sensibile o configurazioni obsolete.
Soluzione:	Rimuovere cartelle .git e riscrivere la storia Git se necessario.
Metodo di detection:	Accesso simulato a ‘.git‘ e analisi file ‘config‘, ‘HEAD‘, ‘logs‘.
Nodi coinvolti:	‘web1‘

Tabella 10: Scheda vulnerabilità GitHistory

MailLeakCreds	CVE Nessuna CVE specifica identificata
Medium	
Descrizione:	Leaking delle credenziali di alcuni account di email.
Impatto:	Recupero di credenziali da email inviate internamente senza cifratura.
Soluzione:	Usare TLS per SMTP e strumenti di DLP per bloccare invii sensibili.
Metodo di detection:	Parsing dei messaggi SMTP acquisiti con sniffing simulato.
Nodi coinvolti:	‘mail1‘

Tabella 11: Scheda vulnerabilità MailLeakCreds

BackupExploit	CVE Nessuna CVE specifica identificata
LOW	
Descrizione:	Perdita di confidenzialità e modifica dell'integrità dei file di backup.
Impatto:	Accesso non autorizzato al servizio di backup, permettendo lettura o modifica dei dump.
Soluzione:	Segmentare il traffico di backup, cifrare i dati e autenticare i client.
Metodo di detection:	Accesso SMB e analisi cartelle condivise contenenti backup.
Nodi coinvolti:	'backup1'

Tabella 12: Scheda vulnerabilità BackupExploit

WeakAdmin	CVE Nessuna CVE specifica identificata
HIGH	
Descrizione:	Un modulo remoto tenta di ottenere privilegi tramite credenziali deboli di amministrazione.
Impatto:	Un attaccante con accesso limitato può ottenere privilegi elevati (Admin o SYSTEM) sfruttando configurazioni errate.
Soluzione:	Applicare la regola del minimo privilegio e utilizzare credenziali forti, evitando configurazioni predefinite.
Metodo di detection:	Simulazione brute-force tramite CyberBattleSim.
Nodi coinvolti:	'ws _e tryedc1'

Tabella 13: Scheda vulnerabilità WeakAdmin

BackupSQLCreds	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Dump di tabelle SQL con presenza di backup non crittografati e pieni di dati sensibili.
Impatto:	Le credenziali SQL sono incluse nei dump o nei file di configurazione del sistema di backup
Soluzione:	Rimuovere le credenziali in chiaro dai backup e usare variabili temporanee sicure
Metodo di detection:	Analisi contenuti dei dump SQL ottenuti tramite SMB.
Nodi coinvolti:	‘backup1’

Tabella 14: Scheda vulnerabilità BackupSQLCreds

Web3SSRF	CVE CVE-2025-2691 HIGH
Descrizione:	Server-Side Request Forgery per interagire con altri host interni.
Impatto:	L’attaccante può sfruttare la web app per inviare richieste arbitrarie a sistemi interni (SSRF).
Soluzione:	Validare gli URL richiesti lato server e isolare la rete interna.
Metodo di detection:	Invio di URL controllati verso localhost, 169.254.x.x, internal/.
Nodi coinvolti:	‘web3’

Tabella 15: Scheda vulnerabilità Web3SSRF

Web3BackupCreds	CVE CVE-2025-21384 HIGH
Descrizione:	Recupero di credenziali da file di backup accessibili.
Impatto:	Backup visibile via HTTP contenente credenziali sensibili.
Soluzione:	Rimuovere i backup dalle directory pubbliche e controllare i permessi di accesso.
Metodo di detection:	Accesso HTTP simulato a /backup/, /config.bak, /users.db.
Nodi coinvolti:	‘web3’

Tabella 16: Scheda vulnerabilità Web3BackupCreds

App1PrivEsc	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Privilege escalation da utente ad amministratore.
Impatto:	Escalation dei privilegi da utente normale a root sul server app1 tramite vulnerabilità locali.
Soluzione:	Aggiornare i pacchetti, rimuovere SUID non necessari e monitorare i binari eseguibili.
Metodo di detection:	Simulazione di exploit noti su binari scrivibili o configurazioni errate
Nodi coinvolti:	‘app1‘

Tabella 17: Scheda vulnerabilità App1PrivEsc

App1DBCreds	CVE Nessuna CVE specifica identificata MEDIUM
Descrizione:	Estrazione credenziali database da log o file.
Impatto:	Esposizione di credenziali del database da file locali o variabili d'ambiente non protette.
Soluzione:	Proteggere file di configurazione e usare meccanismi sicuri per lo storage di segreti.
Metodo di detection:	Lettura di file config, .env o backup di app1 tramite accesso SSH simulato.
Nodi coinvolti:	‘app1‘

Tabella 18: Scheda vulnerabilità App1DBCreds

App2APIExploit	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Sfruttamento API vulnerabile per accesso o comando remoto.
Impatto:	Accesso abusivo all'API esposta da app2 che consente interazioni privilegiate
Soluzione:	Limitare gli endpoint pubblici, usare token sicuri e validare le richieste.
Metodo di detection:	Scansione endpoint REST e invio di richieste alterate.
Nodi coinvolti:	‘app2‘

Tabella 19: Scheda vulnerabilità App2APIExploit

App2FileCreds	CVE Nessuna CVE specifica identificata
Medium	
Descrizione:	Recupero credenziali da file nel server applicativo.
Impatto:	Credenziali hardcoded o file .env accessibili da utenti remoti tramite app2.
Soluzione:	Isolare le variabili sensibili in vault e limitare permessi di lettura ai file.
Metodo di detection:	Accesso file tramite exploit API o SSH, analisi di chiavi e password.
Nodi coinvolti:	‘app2‘

Tabella 20: Scheda vulnerabilità App2FileCreds

App2PrivEsc	CVE Nessuna CVE specifica identificata
HIGH	
Descrizione:	Privilege escalation locale su appserver.
Impatto:	L’attaccante ottiene privilegi SYSTEM su app2 sfruttando vulnerabilità locali.
Soluzione:	Monitorare escalation locali e correggere configurazioni di sistema.
Metodo di detection:	Esecuzione di script automatici per privilege escalation simulata.
Nodi coinvolti:	‘app2‘

Tabella 21: Scheda vulnerabilità App2PrivEsc

MgmtExploit	CVE Nessuna CVE specifica identificata
HIGH	
Descrizione:	Sfruttamento remoto su nodo di management.
Impatto:	L’exploit sfrutta il server di gestione per ottenere il controllo completo della rete.
Soluzione:	Segmentare il nodo di gestione e ridurre la superficie d’attacco con hardening.
Metodo di detection:	Esecuzione exploit via SSH dopo compromissione credenziali.
Nodi coinvolti:	‘mgmt1‘

Tabella 22: Scheda vulnerabilità MgmtExploit

MgmtCreds	CVE Nessuna CVE specifica identificata HIGH
Descrizione:	Credenziali di management trapelate da file o rete.
Impatto:	Compromissione di credenziali amministrative su un nodo di gestione 'mgmt'
Soluzione:	Rotazione periodica delle credenziali e accesso condizionato tramite PAM.
Metodo di detection:	Raccolta di file authorized_keys, .bash_history, e chiavi SSH.
Nodi coinvolti:	'mgmt1'

Tabella 23: Scheda vulnerabilità MgmtCreds

SQLi	CVE CVE-2022-3323 HIGH
Descrizione:	SQL Injection per ottenere accesso o manipolare dati.
Impatto:	Possibilità di eseguire query arbitrarie sul database tramite SQL Injection.
Soluzione:	Usare query parametrizzate e WAF per filtrare le richieste.
Metodo di detection:	Invio di payload SQL su input non sanitizzati.
Nodi coinvolti:	'sql1'

Tabella 24: Scheda vulnerabilità SQLi

SQLiCreds	CVE CVE-2021-36385 HIGH
Descrizione:	Estrazione credenziali utente via SQL Injection.
Impatto:	Estrazione di credenziali di accesso al database tramite SQL Injection.
Soluzione:	Segmentazione dei privilegi nel DBMS e hashing delle password.
Metodo di detection:	Estrazione dati da tabelle users, credentials, login in risposta a query iniettate.
Nodi coinvolti:	'sql1'

Tabella 25: Scheda vulnerabilità SQLiCreds

SQLiDataBreach	CVE CVE-2025-1094 HIGH
Descrizione:	Ottenimento illecito di dati sensibili dal database via SQLi.
Impatto:	Dati sensibili del database vengono esfiltrati sfruttando vulnerabilità SQLi.
Soluzione:	Log delle query anomale e cifratura dei dati sensibili lato database.
Metodo di detection:	Simulazione di dump massivo dei dati tramite SQL Injection.
Nodi coinvolti:	‘sql1’

Tabella 26: Scheda vulnerabilità SQLiDataBreach

7 Riferimenti

- <https://github.com/microsoft/CyberBattleSim>
- <https://github.com/cage-challenge/CybORG>
- <https://github.com/Jjschwartz/NetworkAttackSimulator>
- <https://networkattacksimulator.readthedocs.io/en/latest/tutorials/scenarios.html>
- <https://cage-challenge.github.io/cage-challenge-4/pages/>
- https://cage-challenge.github.io/cage-challenge-4/pages/tutorials/03_Actions/B_Blue_Actions/1_Monitor/
- https://cage-challenge.github.io/cage-challenge-4/pages/tutorials/03_Actions/C_Red_Actions/1_Discover_Remote_Systems/
- <https://cage-challenge.github.io/cage-challenge-4/pages/reference/agents/FiniteStateRedAgent/>
- https://cage-challenge.github.io/cage-challenge-4/pages/reference/agents/red_overview/
- <https://www.guardicore.com/infectionmonkey/>
- <https://github.com/guardicore/monkey>
- <https://tajsecguy.medium.com/integrating-adversary-emulation-using-infection-monkey-3a2f3a2a2a2a>
- https://www.researchgate.net/publication/354065901_CybORG_A_Gym_for_the_Development_of_Autonomous_Cyber_Agents
- <https://arxiv.org/html/2310.07745v2#:~:text=unprocessed\%20dictionary\%20observations,been\%20run\%20by\%20The\%20Technical>

- <https://www.scirp.org/journal/paperinformation?paperid=141811#:~:text=Currently\%2C\%20the\%20Microsoft\%20CyberBattleSim\%20environment, and\%20applicability\%20of\%20CyberBattleSim\%20research>
- <https://www.microsoft.com/en-us/security/blog/2021/04/08/gamifying-machine-learning#:~:text=Today\%2C\%20we\%E2\%80\%99d\%20like\%20to\%20share,com\%2Fmicrosoft\%2FCyberBattleSim>
- <https://www.cybersecurity360.it/soluzioni-aziendali/cyberbattlesim-il-tool-per-s #:~:text=In\%20poche\%20parole\%2C\%20CyberBattleSim\%20\%C3\%A8,attraverso\%20i\%20cosiddetti\%20movimenti\%20laterali>
- <https://arxiv.org/abs/2108.13980>

8 Appendici

I simulatori di cybersicurezza dedicati al penetration testing forniscono ambienti controllati in cui testare tecniche di attacco informatico in modo sicuro e ripetibile. Questi strumenti permettono a ricercatori e sviluppatori di agenti autonomi di esercitare strategie di attacco o difesa senza rischi per sistemi reali. In particolare, con l'arrivo del RL in ambito di sicurezza sono nati diversi gym simulativi ispirati a reti enterprise reali, utili per addestrare agenti automatici in scenari di attacco. Esamineremo tre di questi simulatori: CyberBattleSim di Microsoft, CybORG (sviluppato nell'ambito delle CAGE Challenges) e Network Attack Simulator (NASim). Verranno analizzati per architettura, modalità di funzionamento, approccio al pentesting (black box vs. white box), tecnologie utilizzate, supporto all'apprendimento per rinforzo, esempi d'uso, seguiti da un confronto diretto sui principali parametri (facilità d'uso, realismo, astrazione, personalizzazione e pro e contro).

.1 CyberBattleSim (Microsoft)

.1.1 Funzionamento e architettura

L'architettura di CyberBattleSim rappresenta la rete come un grafo di nodi connessi, ognuno avente proprietà (come sistema operativo, servizi) e vulnerabilità predefinite. Il simulatore è parametrizzato da una topologia fissa e da un insieme di vulnerabilità note che l'agente attaccante può sfruttare per muoversi lateralmente. In ogni simulazione, si assume che l'attaccante abbia già compromesso inizialmente un nodo della rete da cui inizia l'esplorazione e conquista di altri host. L'ambiente è parzialmente osservabile: l'agente non conosce a priori l'intera mappa di rete, ma deve scoprirla progressivamente dai nodi compromessi, riflettendo un approccio **black box** in cui il pentester non possiede conoscenza interna iniziale. Le azioni disponibili includono attacchi locali (su nodi già ottenuti), attacchi remoti su nodi adiacenti e movimenti laterali (come stabilire connessioni usando credenziali rubate). Ogni vulnerabilità è modellata in modo astratto tramite una probabilità di successo e un effetto deterministico se sfruttata (come ottenere accesso root, estrarre credenziali e così via). Non viene eseguito codice malevolo reale; l'intero attacco è simulato a livello logico. Ad esempio, un exploit SMB è rappresentato da condizioni (sistema vulnerabile, porte aperte, credenziali valide) e da un esito positivo con una certa probabilità. Un semplice agente difensore stocastico è incluso per modellare

la reazione di un sistema di difesa: con una certa probabilità il defender rileva l'attività anomala e mitiga l'attacco, ad esempio ripristinando (re-imaging) i nodi compromessi. Questa mitigazione è anch'essa astratta e avviene nell'arco di più step simulativi, durante i quali l'attaccante può essere rallentato. In sintesi, l'architettura di CyberBattleSim è pensata per catturare i concetti chiave di un attacco in rete senza scendere al livello di pacchetti di rete o exploit reali. Inoltre, dal punto di vista tecnologico, CyberBattleSim è implementato in Python e offre un'interfaccia simile a OpenAI Gym. Ciò consente di interagire con il simulatore tramite i tipici metodi `reset()` e `step()`.

1.2 Implementazione

La definizione della rete è partita dalla definizione dei diversi nodi la cui classe `NodeInfo` può prendere diversi parametri che definiscono in maniera astratta diversi elementi:

- Servizi come RDP con tanto di credenziali di accesso;
- Vulnerabilità;
- Valore del nodo in caso di conquista il quale influenza il valore della reward;
- Una lista di proprietà customizzabili tipiche di una rete come sistema operativo della macchina ma anche ruoli ed altro;
- Regole per il Firewall;
- Se questo è il nodo di partenza dell'agente attaccante;
- Se la macchina è attiva;
- Se la macchina può essere re-immaginata da un agente difensivo per eliminare l'influenza dell'agente attaccante;
- Stringa da stampare nel caso in cui il Red Agent riesca a conquistare il nodo.

Ecco quindi la definizione di un nodo che offre il servizio SMB con credenziali root:toor, firewall che blocca tutto il traffico in entrata tranne quello SMB e possiede diverse proprietà e vulnerabilità:

```
"ws4": NodeInfo(
    services=[ListeningService("SMB", allowedCredentials=["root:toor"])],
    firewall=FirewallConfiguration(
        incoming=[FirewallRule("SMB", ALLOW)],
        outgoing=default_allow_rules
    ),
    value=1.0,
    properties=["os:Windows", "role:workstation", "service:smb"],
    vulnerabilities={
        "SMBWeakShare": vuln_smb,
        "SMBCreds": vuln_smb_creds,
    }
),
```

Listing 1: Definizione del nodo "ws4" con servizi

Si è passati poi alla definizione delle vulnerabilità usando la classe **VulnerabilityInfo** che può prendere diversi parametri che definiscono in maniera astratta diversi elementi:

- Stringa che descrive la vulnerabilità;
- Tipo di vulnerabilità che può essere Locale o Remota;
- L'outcome e ciò cosa succede se la vulnerabilità viene sfruttata che descriveremo a breve;
- Precondizione per lo sfruttamento o esistenza della vulnerabilità;
- Costo dello sfruttare la vulnerabilità;
- Messaggio di reward per il corretto sfruttamento della vulnerabilità.

CyberBattleSim fornisce inoltre diversi tipi di outcome per l'effettivo sfruttamento di una vulnerabilità:

- **LateralMove**: rappresenta un movimento laterale riuscito verso un nodo target nella rete.
- **CustomerData**: indica che l'agente ha ottenuto accesso a dati sensibili (es. dati clienti) sul nodo compromesso.
- **PrivilegeEscalation**: rappresenta un'escalation dei privilegi dell'agente sul nodo (da utente a admin o system).
- **SystemEscalation**: escalation specifica al livello di privilegio **SYSTEM**, il massimo possibile su Windows.
- **AdminEscalation**: escalation al livello di **Administrator** locale sul nodo compromesso.
- **ProbeSucceeded**: l'agente ha ottenuto proprietà o informazioni sul nodo target (come OS, ruolo, servizi).
- **ProbeFailed**: il tentativo di probing non ha avuto successo e quindi nessuna informazione è stata acquisita.
- **ExploitFailed**: il tentativo di sfruttare la vulnerabilità è fallito e quindi nessun accesso o effetto è stato ottenuto.
- **LeakedCredentials**: l'agente ha ottenuto una o più credenziali valide da usare in attacchi futuri.
- **LeakedNodesId**: l'agente ha scoperto l'identità (ID) di altri nodi nella rete grazie all'exploit.

```

vuln_smb_creds = VulnerabilityInfo(
    description="SMB share creds leak",
    type=VulnerabilityType.REMOTE,
    outcome=LeakedCredentials(credentials=[
        CachedCredential(node="backup1", port="SMB", credential=""
                          root:toor")
    ]),
    precondition=Precondition("service:smb"),
    cost=1.4,
    reward_string="Backup server SMB root password leaked"
)

```

Listing 2: Esempio di vulnerabilità definita in CyberBattleSim

Successivamente si definiscono gli edges (archi) tra i diversi nodi definiti il che creerà i path che l'agente potrà seguire. Poi si utilizza NetworkX di Python per creare il grafo che verrà passato alla classe relativa alla creazione dell'ambiente e cioè **CyberBattleEnv** a cui è possibile fornire l'agente attaccante ed anche quello difensivo. Per l'agente attaccante, in questo momento, si definisce l'obiettivo (**AttackerGoal**) che è espresso in "quantità di rete conquistata": se si scrive "**own_atleast_percent=1.0**" si sta dicendo che l'agente vincerà solo quando controllerà l'intera rete. Questo è proprio il nostro caso.

Per contrastare l'agente attaccante è stato scelto l'agente difensivo built-in **Scan And Reimage Compromised Machines** che rappresenta un *difensore reattivo e probabilistico* che si occupa di monitorare la rete e di ripristinare i nodi compromessi ed il cui comportamento è determinato da tre parametri chiave:

- **probability=0.7**: indica la **probabilità** che l'agente esegua un'azione di difesa (scan e re-image) durante un turno. Nel caso specifico, l'azione verrà eseguita nel 70% dei casi in cui è permessa.
- **scan_capacity=3**: definisce il **numero massimo di nodi** che l'agente può ispezionare in un singolo ciclo di scansione. In questo esempio, l'agente può controllare fino a tre nodi per volta.
- **scan_frequency=7**: rappresenta l'**intervallo di tempo (in turni)** tra due scansioni consecutive. Un valore di 7 indica che il difensore effettuerà una scansione ogni 7 time step.

Questo tipo di agente è utile per testare la robustezza delle strategie di attacco in presenza di una difesa attiva, anche se limitata. È particolarmente adatto per ambienti simulati in cui si vogliono valutare le interazioni dinamiche tra attaccanti e difensori automatici.

Infine andiamo a definire l'agente DQN con i seguenti parametri:

- **gamma=0.95**: fattore di sconto per le ricompense future (valori vicini a 1 danno più importanza al lungo termine).
- **replay_memory_size=10000**: numero massimo di esperienze memorizzate per il training;
- **target_update=100**: frequenza, in passi, con cui viene aggiornata la rete target (update meno frequenti migliorano la stabilità).

- **batch_size=64**: numero di esperienze campionate a ogni passo di apprendimento che impatta la stabilità e la velocità del training.
- **learning_rate=0.001**: velocità di aggiornamento dei pesi (valori piccoli favoriscono una convergenza più stabile).

```
policy = DeepQLearnerPolicy(
    ep=ep,
    gamma=0.95,
    replay_memory_size=10000,
    target_update=100,
    batch_size=64,
    learning_rate=0.001,
)
```

Listing 3: Configurazione della policy Deep Q-Learning in CyberBattleSim

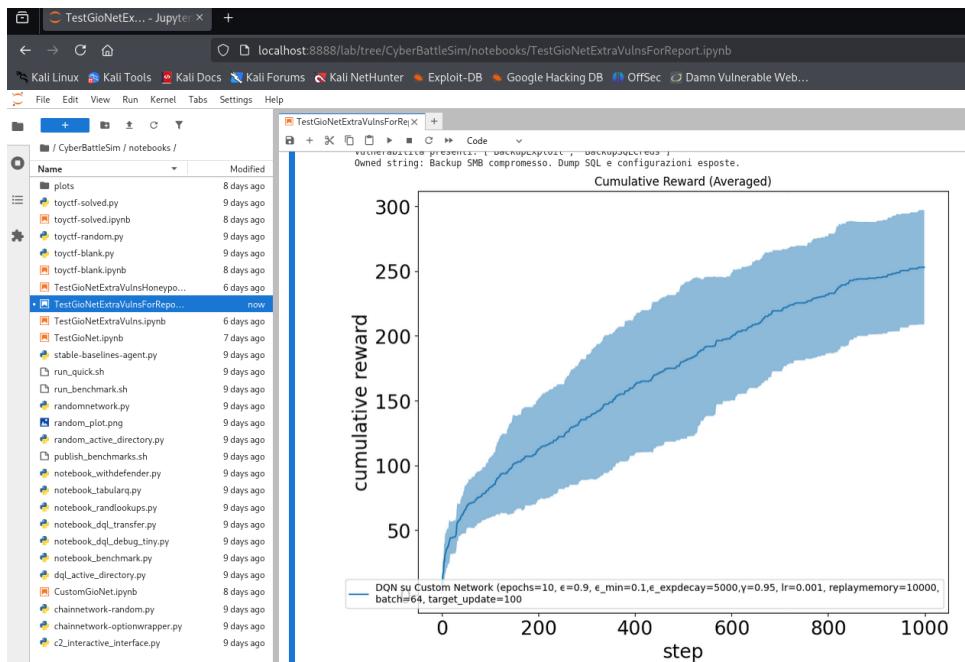


Figura 6: Reward cumulativo di DQN nell’ambiente simulato con 10 episodi e 1000 iterazioni per ognuno

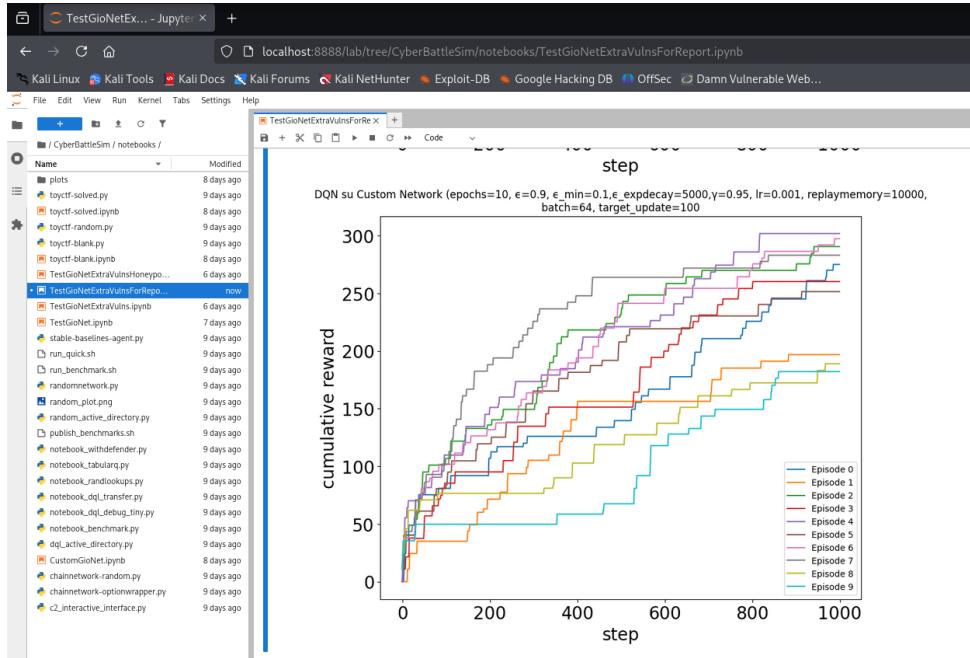


Figura 7: Reward per episodio di DQN nell’ambiente simulato con 10 episodi e 1000 iterazioni per ognuno e con agente difensivo

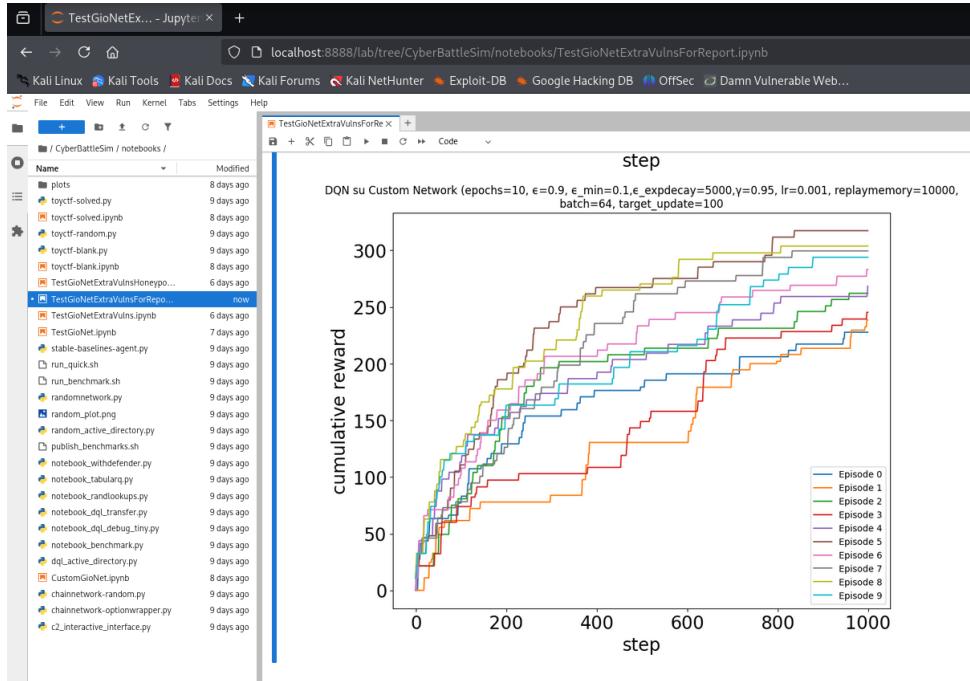


Figura 8: Reward per episodio di DQN nell’ambiente simulato con 10 episodi e 1000 iterazioni per ognuno e nessun agente difensivo

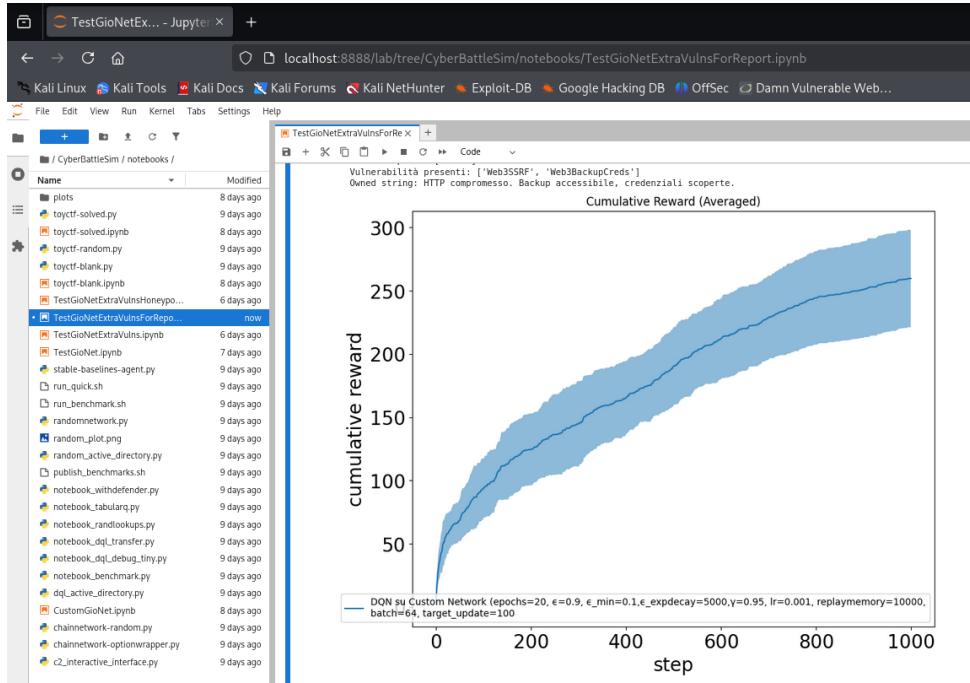


Figura 9: Reward cumulativo di DQN nell’ambiente simulato con 20 episodi e 1000 iterazioni per ognuno e con agente difensivo

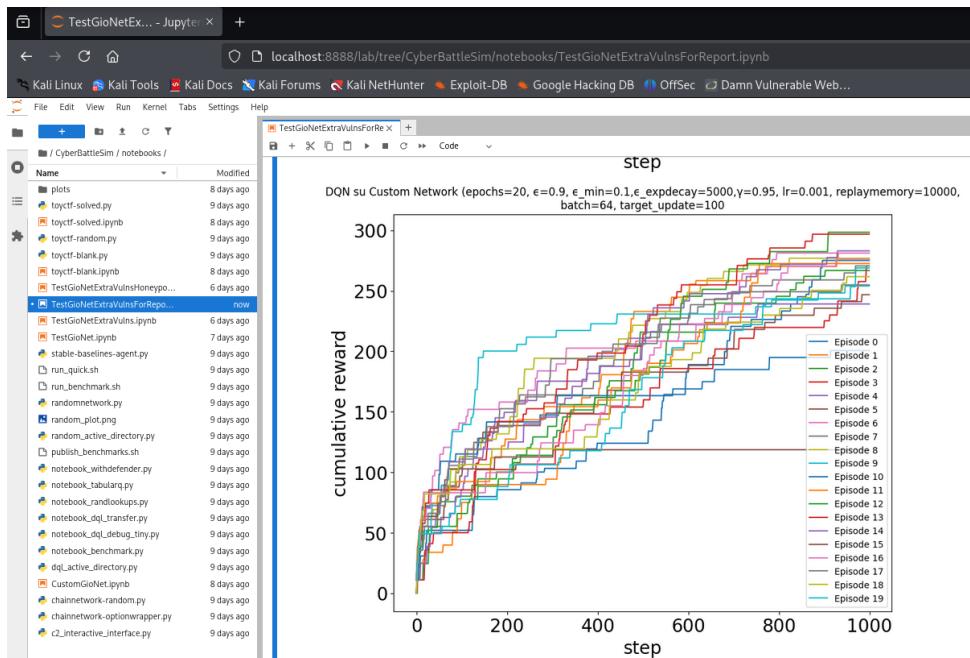


Figura 10: Reward per episodio di DQN nell’ambiente simulato con 20 episodi e 1000 iterazioni per ognuno e con agente difensivo

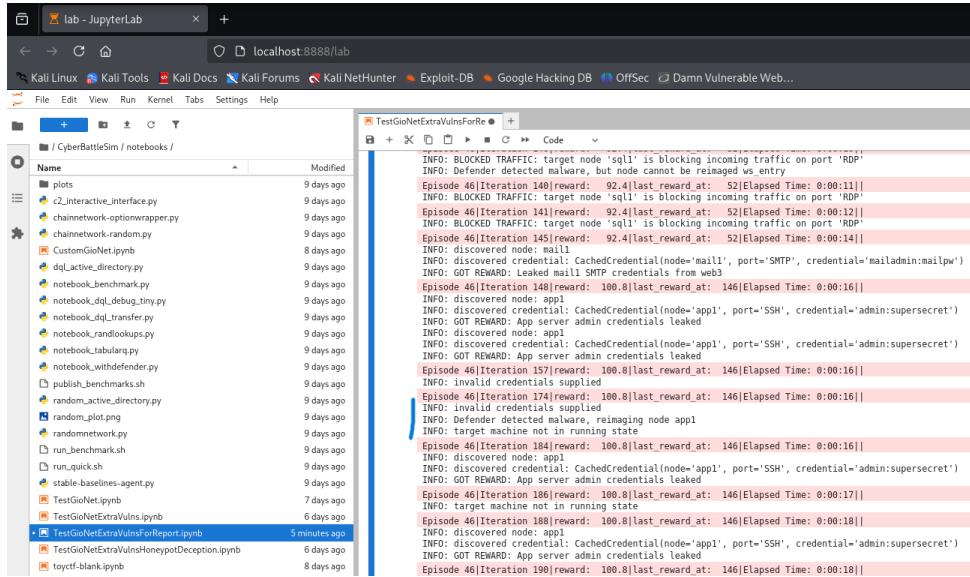


Figura 11: Esempio funzionamento agente difensivo che individua un nodo con malware e cerca di ripristinarlo

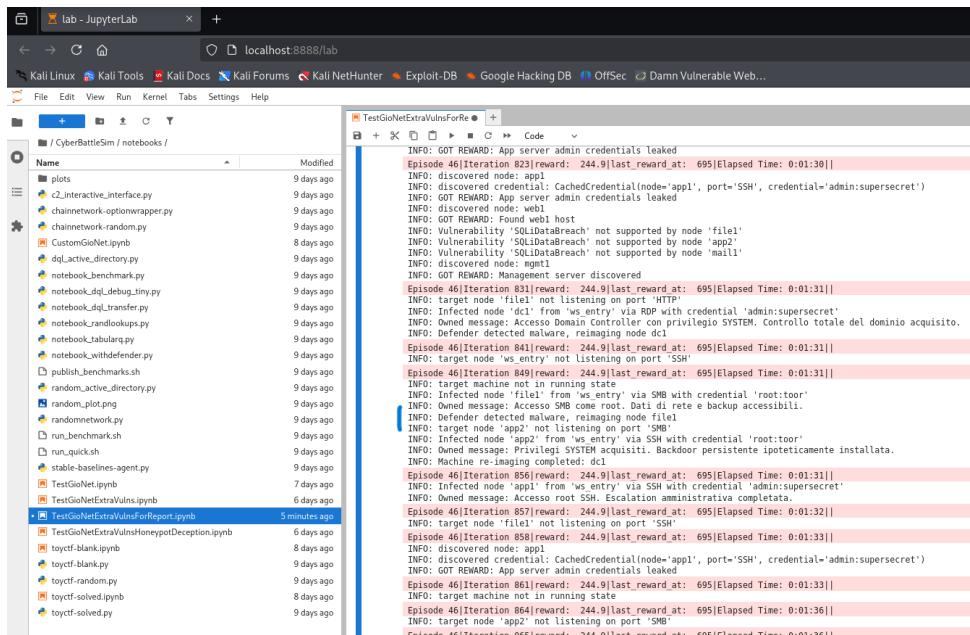


Figura 12: Esempio funzionamento agente difensivo che individua un nodo con malware e cerca di ripristinarlo

2 CybORG (CAGE Challenge)

2.1 Architettura ibrida: simulazione ed emulazione

CybORG (Cyber Operations Research Gym) è un ambiente sviluppato originariamente dal Defence Science and Technology Group in collaborazione con altri enti, nell’ambito delle sfide internazionali TTCP CAGE (Collaborative Autonomous Gaming Environment). Presentato inizialmente nel 2021, CybORG si distingue per un’architettura **ibrida** che combina una componente simutativa e una emulativa sotto una comune interfaccia. In

pratica, CybORG offre un modello di rete simulata analogo a quello degli altri tool (con host virtuali e azioni astratte), ma consente anche di collegare tali azioni a una rete reale composta da macchine virtuali nel cloud. Grazie a questa interfaccia unificata, è possibile addestrare un agente in modalità simulazione veloce e poi testarne il comportamento in uno scenario reale equivalente, ottenendo così un efficace trasferimento dal simulato al reale. La parte simulativa di CybORG modella reti con un livello di dettaglio maggiore rispetto a CyberBattleSim: include host con sistemi operativi specifici (Windows e Linux), servizi in esecuzione, utenti, credenziali, processi, e persino componenti di deception come honeypot o file esca. La presenza di questi elementi consente di simulare non solo l'attacco, ma anche attività di utenti legittimi e inganni nati per confondere l'attaccante. Le challenge sono caratterizzate da scenari dove tipicamente ci sono tre tipi di agenti: **Red** (attaccante), **Blue** (difensore) e agenti **Green** che rappresentano utenti o servizi leciti di background. L'agente Red in CybORG svolge un ruolo di pentester offensivo: può iniziare con o senza conoscenze pregresse a seconda delle impostazioni ma in genere è in modalità black box. Ad esempio, in *Challenge 4* di CAGE (ambientazione enterprise) l'attaccante inizia probabilmente con accesso solo alla rete esterna e deve infiltrarsi tramite tecniche di phishing o vulnerabilità esposte. Sia il blue che il red agent possiedono diverse azioni per portare a termine i loro obiettivi. Questa ricchezza di azioni e ruoli evidenzia un'architettura orientata a riprodurre fasi multiple di un vero incidente di sicurezza, dal primo ingresso fino alla risposta attiva del difensore e al gioco di inganni reciproci. In termini implementativi, CybORG è scritto in Python e sfrutta anch'esso l'interfaccia OpenAI Gym. Esistono inoltre scenari predefiniti come Enterprise Scenario di Challenge 4 che configurano automaticamente una certa topologia e set di host/servizi. In alternativa, l'utente avanzato può creare nuovi scenari definendo manualmente gli host, servizi e vulnerabilità: questo richiede però familiarità con il mondo delle infrastrutture di rete. Tuttavia uno scenario può essere creato anche usando un file .yaml.

2.2 Implementazione

In questo caso è stata scelta la challenge 4 ed usato lo scenario preesistente Enterprise Scenario. L'obiettivo della challenge è quello di creare un agente blue difensivo che riesce a contrastare il red agent senza però ostacolare il funzionamento del green agent: esistono infatti alcune azioni del blue agent che possono andare in conflitto col funzionamento del green agent (vedremo in seguito).

La CAGE Challenge 4 propone uno scenario di cyber defence in un ambiente aziendale in cui il network è suddiviso in più zone di sicurezza (come Contractor, HQ Admin, Office e reti operative), ciascuna protetta da un agente Blue dedicato. Gli agenti Red iniziano con accesso iniziale nella subnet Contractor e dispongono di capacità come phishing, scansioni stealth/aggressive, exploit remoti, movimenti laterali, degradazione di servizi e impatto sugli asset.

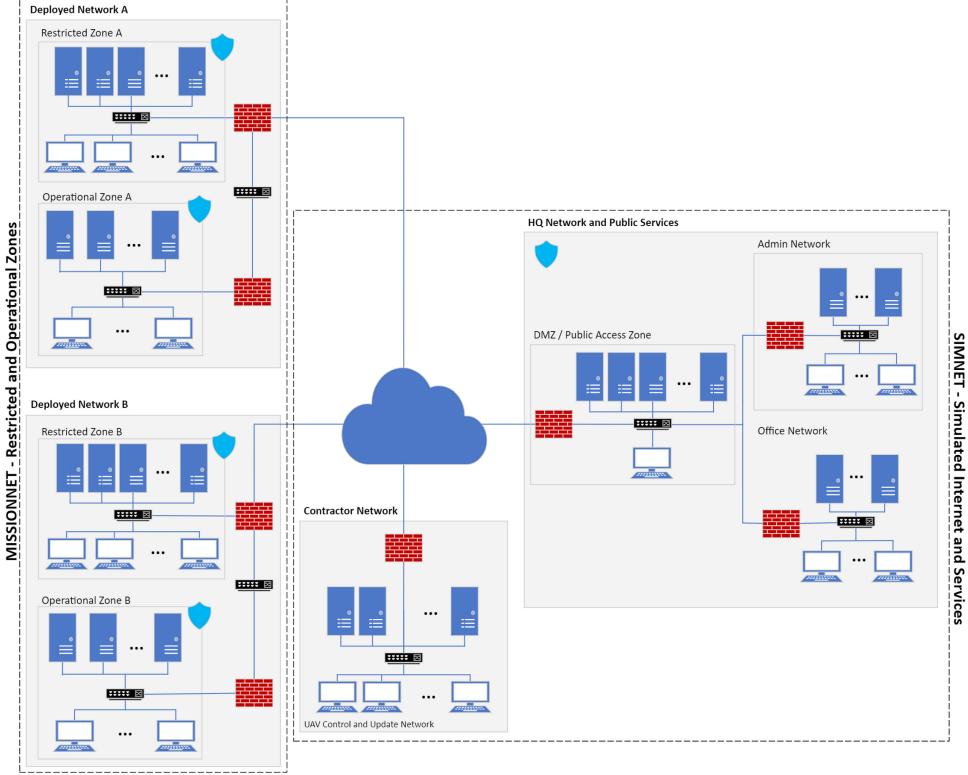


Figura 13: Infrastruttura di rete di Challenge 4

Le azioni possibili dell'agente blue sono:

- **Monitor Descrizione:** esegue una scansione di rete per raccogliere indicatori di attività sospette (come connessioni insolite o processi anomali). *Precondizione:* azione sempre disponibile come default ad ogni turno. *Output:* segnali di possibile compromissione rilevati.
- **Analyse Descrizione:** approfondisce l'analisi su un host sospetto per verificare presenza di malware o shell remote. *Precondizione:* è efficace se sono stati rilevati segnali sospetti tramite **Monitor**. *Output:* conferma di shell o malware, con livello di dettaglio superiore rispetto a **Monitor**.
- **DeployDecoy Descrizione:** installa un servizio d'esca (honeypot) su un host specifico e qualsiasi accesso non autorizzato genera subito allarme. *Precondizione:* l'host deve essere attivo e disponibile. *Output:* alert univoco se accesso malevolo a quello specifico servizio.
- **Remove Descrizione:** rimuove una shell utente (non root) installata da Red su un host compromesso. *Precondizione:* necessita di prove da **Analyse** che una shell utente sia presente. *Effetto:* rimuove l'accesso utente ma non è efficace contro shell root.
- **Restore Descrizione:** esegue un re-image completo dell'host, cancellando qualsiasi compromissione, compresa la shell root. *Precondizione:* host compromesso rilevato, anche a livello root. *Effetto:* ripristina completamente lo stato iniziale della macchina.

- **BlockTrafficZone / AllowTrafficZone** *Descrizione:* blocca o riattiva il traffico tra specifiche zone di rete per impedire movimenti laterali o ripristinarli. *Precondizione:* configurazioni di rete esistenti tra sottoreti. *Effetto:* modifica alle regole di firewall utili per isolare un attacco in corso o riaprire segmenti liberi. **Questa azione va usata con cautela perché può intralciare il funzionamento del Green Agent.**

Le azioni possibili dell’agente red sono:

- **DiscoverRemoteSystems** *Descrizione:* esegue uno scan di rete (ping) su una subnet per individuare host attivi. *Precondizione:* è necessaria la conoscenza della subnet e una sessione attiva. *Effetto:* restituisce la lista degli host attivi nella subnet specificata.
- **ServiceDiscovery (Aggressive / Stealth)** *Descrizione:* identifica i servizi attivi su un host. *Precondizione:* è richiesta una sessione su un nodo target. *Effetto:* modalità **aggressive** fornisce risultati rapidi ma con alta probabilità di allarme e **stealth** è lenta ma silenziosa.
- **ExploitRemoteService** *Descrizione:* tenta di ottenere una shell utente sfruttando vulnerabilità note in servizi remoti. *Precondizione:* il servizio vulnerabile deve essere stato precedentemente scoperto. *Effetto:* se ha successo, fornisce accesso utente al nodo target.
- **PrivilegeEscalate** *Descrizione:* esegue un exploit locale per ottenere una shell con privilegi root. *Precondizione:* presenza di una sessione utente attiva sul nodo target. *Effetto:* eleva i privilegi dell’agente Red al massimo livello.
- **DiscoverDeception** *Descrizione:* analizza un nodo per verificare la presenza di decoy/honeypot. *Precondizione:* il nodo e i servizi devono essere noti. *Effetto:* riduce il rischio di attacchi contro honeypot dispiegati dal Blue.
- **DegradeServices** *Descrizione:* compromette o degrada la funzionalità di servizi utilizzati da utenti legittimi (Green). *Precondizione:* è richiesta una sessione root attiva sul nodo. *Effetto:* diminuisce l’usabilità o disponibilità del servizio per gli utenti.
- **Impact** *Descrizione:* disabilita un servizio OT critico all’interno della rete operativa. *Precondizione:* accesso root su un nodo con un servizio OT. *Effetto:* genera ricompensa per Red, rappresentando un impatto critico sulla missione.
- **Withdraw** *Descrizione:* rimuove tutte le sessioni Red da un nodo compromesso. *Precondizione:* almeno una sessione Red (utente o root) deve essere attiva. *Effetto:* riduce le tracce dell’attacco, utile per evitare il rilevamento.

Inoltre, vengono forniti diversi agenti Red preconfezionati e quello scelto è stato FiniteStateRedAgent, un agente Red basato su una macchina a stati finiti che gestisce autonomamente l’avanzamento dell’attacco su host noti seguendo una transizione logica tra stati:

- *Meccanismo interno*: mantiene una variabile `host_states` con 9 stati possibili per ciascun host (K, KD, S, SD, U, UD, R, RD, F) che descrivono progressione da semplice rilevamento fino alla compromissione completa. Le transizioni tra stati avvengono in base a matrici predefinite che dipendono dal successo o fallimento dell'azione.
- *Selezione dell'azione*: ad ogni turno, l'agente osserva lo stato corrente, processa informazioni (come nuovi host, rimozione di sessioni) e utilizza le matrici per scegliere l'azione.

Nella CAGE Challenge 4, `FiniteStateRedAgent` può essere esteso definendo una sottoclassa nel nostro caso chiamata `Cage4CustomRed`, che consente di modificare la logica delle transizioni di stato. Gli elementi personalizzabili principali sono:

- **Metodo `_set_host_state_priority_list`** permette di influenzare la scelta dell'host target, definendo un dizionario con probabilità associate ai 9 stati noti dei nodi, la somma dei quali deve essere pari a 100.
- **Metodo `_state_transitions_probability`** consente di definire una nuova matrice di probabilità per la selezione dell'azione in base allo stato corrente. Ogni riga corrisponde a uno stato, ogni colonna a un'azione.

Ogni host conosciuto dall'agente può trovarsi in uno dei seguenti 9 stati, che rappresentano fasi progressivi dell'attacco:

- **K (Known)**: l'IP dell'host è stato scoperto.
- **KD (Known + Discovery)**: oltre all'IP, è stata scoperta la rete della subnet del nodo.
- **S (Services)**: sono stati identificati i servizi in esecuzione sull'host.
- **SD (Services + Discovery)**: servizi e subnet sono stati scoperti.
- **U (User)**: l'agente ha ottenuto una shell utente.
- **UD (User + Discovery)**: shell utente e subnet discovery completati.
- **R (Root)**: accesso root è stato ottenuto.
- **RD (Root + Discovery)**: shell root e discovery della rete completati.
- **F (Final)**: stato terminale, non ci sono ulteriori azioni da eseguire.

Inoltre, la matrice delle probabilità (definita da `_state_transitions_probability`) indica la scelta tra le possibili azioni per ciascun stato (assicurandosi che la somma per riga sia pari a 1). Un esempio per lo stato 'K' potrebbe essere:

'K': [0.5, 0.25, 0.25, None, ..., None]

che indica l'assegnazione di probabilità per le azioni che portano da K agli stati KD, KS, ecc.

```

** Turn 1 for red_agent_0 ***
Action: PrivilegeEscalate contractor.network_subnet_user_host_9
Action Success: IN_PROGRESS

Observation:
{'contractor_network_subnet_user_host_9': {'Interface': [{'Subnet': IPv4Network('10.0.109.0/24'), 'ip_address': IPv4Address('10.0.109.41')}], 'Sessions': [{Type: <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_0', 'session_id': 0, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_9'}}}

Host States:
{'10.0.109.41': {'hostname': 'contractor_network_subnet_user_host_9', 'state': 'U'}}

[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze restricted_zone_a_subnet_router'}
[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze restricted_zone_b_subnet_router'}
[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze public_access_zone_subnet_router'}
[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze restricted_zone_a_subnet_router'}
[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze restricted_zone_b_subnet_router'}
[BlueDefenderAgent] get_action called, observation type: <class 'dict'>
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': 'Analyze public_access_zone_subnet_router'}

** Turn 2 for red_agent_0 ***
Action: PrivilegeEscalate contractor.network_subnet_user_host_9
Action Success: TRUE

Observation:
{'contractor_network_subnet_user_host_9': {'Interface': [{'Subnet': IPv4Network('10.0.109.0/24'), 'ip_address': IPv4Address('10.0.109.41')}], 'Sessions': [{Type: <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_0', 'session_id': 0, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_9'}}}

Host States:
{'10.0.109.41': {'hostname': 'contractor_network_subnet_user_host_9', 'state': 'R'}}
```

Figura 14: Screenshot dell'esecuzione con successo di un'azione da parte del custom Blue Agent

```

** Turn 1 for red_agent_0 ***
Action: PrivilegeEscalate restricted_zone_b_subnet_user_host_1
Action Success: TRUE

Observation:
{'restricted_zone_b_subnet_user_host_1': {'Interface': [{'Subnet': IPv4Network('10.0.96.0/24'), 'ip_address': IPv4Address('10.0.96.138')}], 'Sessions': [{Type: <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_3', 'session_id': 0, 'username': 'root'}], 'System info': {'Hostname': 'restricted_zone_b_subnet_user_host_1'}}}

Host States:
{'10.0.96.138': {'hostname': 'restricted_zone_b_subnet_user_host_1', 'state': 'R'}}
```

Figura 15: Screenshot dell'esecuzione con successo di un'azione da parte del custom Blue Agent

```

** Turn 4 for red_agent_0 ***
Action: Impact contractor_network_subnet_user_host_9
Action Success: FALSE

Observation:
{'contractor_network_subnet_user_host_9': {'Interface': [{'Subnet': IPv4Network('10.0.109.0/24'), 'ip_address': IPv4Address('10.0.109.41')}], 'Sessions': [{Type: <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_0', 'session_id': 0, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_9'}}}

Host States:
{'10.0.109.41': {'hostname': 'contractor_network_subnet_user_host_9', 'state': 'R'}}
```



```

** Turn 2 for red_agent_3 ***
Action: PrivilegeEscalate restricted_zone_b_subnet_user_host_1
Action Success: TRUE

Observation:
{'restricted_zone_b_subnet_user_host_1': {'Interface': [{'Subnet': IPv4Network('10.0.96.0/24'), 'ip_address': IPv4Address('10.0.96.138')}], 'Sessions': [{Type: <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_3', 'session_id': 0, 'username': 'root'}], 'System info': {'Hostname': 'restricted_zone_b_subnet_user_host_1'}}}
```

Figura 16: Screenshot dell'esecuzione con successo di un'azione da parte del Red Agent

```

NewTests.ipynb
+ - X □ ▶ C Code

Host States:
{'10.0.109.41': {'hostname': 'contractor_network_subnet_user_host_9',
  'state': 'R'}}

** Turn 4 for red_agent_3 **
Action: DegradeServices restricted_zone_b_subnet_user_host_1
Action Success: TRUE

Observation:
{'restricted_zone_b_subnet_user_host_1': {'Interface': [{"Subnet": IPv4Network('10.0.96.0/24'),
   'ip_address': IPv4Address('10.0.96.138')}], 'Processes': [{"Connections": [{"Transport Protocol": <TransportProtocol.UNKNOWN: 1>, 'local_address': IPv4Address('0.0.0.0'), 'local_port': 22}, {"Known Path": <Path.UNKNOWN: 1>, 'Known Process': <ProcessName.SSHD: 7>, 'PID': 6607, 'Path': '/usr /sbin', 'process_name': <ProcessName.SSHD: 7>, 'process_type': <ProcessType.SSH: 2>, 'username': 'user'}, {"Connections": [{"Transport Protocol": <TransportProtocol.UNKNOWN: 1>, 'local_address': IPv4Address('0.0.0.0'), 'local_port': 3390}, {"Known Path": <Path.UNKNOWN: 1>, 'Known Process': <ProcessName.MYSQLD: 9>, 'PID': 4162, 'Path': '/usr /sbin', 'process_name': <ProcessName.MYSQLD: 9>, 'process_type': <ProcessType.MYSQL: 12>, 'username': 'user'}, {"Connections": [{"Transport Protocol": <TransportProtocol.UNKNOWN: 1>, 'local_address': IPv4Address('0.0.0.0'), 'local_port': 80}, {"Known Path": <Path.UNKNOWN: 1>, 'Known Process': <ProcessName.APACHE2: 13>, 'PID': 2808, 'Path': '/usr /sbin', 'process_name': <ProcessName.APACHE2: 13>, 'process_type': <ProcessType.WEBSERVER: 7>, 'username': 'user'}, {"Connections": [{"Transport Protocol": <TransportProtocol.UNKNOWN: 1>, 'local_address': IPv4Address('0.0.0.0'), 'local_port': 25}, {"Known Path": <Path.UNKNOWN: 1>, 'Known Process': <ProcessName.SMTP: 11>, 'PID': 1522, 'Path': '/usr /sbin', 'Process Version': <ProcessVersion.HARAKA_2_8_9: 8>, 'username': 'root'}]}]}]}

```

Figura 17: Screenshot dell'esecuzione con successo di un azione da parte del Red Agent

```

NewTests.ipynb
+ - X □ ▶ C Code
observation: {'success': <TernaryEnum.TRUE: 1>, 'action': Remove_public_access_zone_subnet_router}

** Turn 93 for red_agent_0 **
Action: DiscoverDeception contractor_network_subnet_user_host_2
Action Success: TRUE

Observation:
{'contractor_network_subnet_server_host_0': {'Interface': [{"Subnet": IPv4Network('10.0.109.0/24'),
   'ip_address': IPv4Address('10.0.109.254')}], 'Sessions': [{"Type": <SessionType.SSH: 2>, 'agent': 'red_agent_0', 'session_id': 2, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_server_host_0'}},
'contractor_network_subnet_user_host_1': {'Interface': [{"Subnet": IPv4Network('10.0.109.0/24'),
   'ip_address': IPv4Address('10.0.109.108')}], 'Sessions': [{"Type": <SessionType.SSH: 2>, 'agent': 'red_agent_0', 'session_id': 1, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_1'}},
'contractor_network_subnet_user_host_8': {'Interface': [{"Subnet": IPv4Network('10.0.109.0/24'),
   'ip_address': IPv4Address('10.0.109.182')}], 'Sessions': [{"Type": <SessionType.SSH: 2>, 'agent': 'red_agent_0', 'session_id': 8, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_8'}},
'contractor_network_subnet_user_host_9': {'Interface': [{"Subnet": IPv4Network('10.0.109.0/24'),
   'ip_address': IPv4Address('10.0.109.41')}], 'Sessions': [{"Type": <SessionType.RED_ABSTRACT_SESSION: 10>, 'agent': 'red_agent_0', 'session_id': 9, 'username': 'root'}], 'System info': {'Hostname': 'contractor_network_subnet_user_host_9'}}}

Host States:
{'10.0.109.108': {'hostname': 'contractor_network_subnet_user_host_1', 'state': 'RD'},
 '10.0.109.134': {'hostname': None, 'state': 'SD'}}

```

Figura 18: Screenshot dell'esecuzione con successo di un azione da parte del Red Agent

.3 Network Attack Simulator (NASim)

.3.1 Funzionamento e architettura

Network Attack Simulator (NASim) è un simulatore leggero e modulare sviluppato in ambito accademico (iniziato da Jonathon Schwartz nel 2019) per modellare attacchi di rete in scenari di penetration testing classici. NASim è scritto in Python e fornisce un ambiente di simulazione ad alto livello che punta a riprodurre gli aspetti salienti di un

pentest senza scendere nei dettagli di exploit reali. L’architettura di NASim ruota attorno al concetto di **scenario**: un file di configurazione (in formato YAML) o generato via codice che descrive sia la rete bersaglio sia le capacità dell’attaccante. Lo scenario definisce:

- la **configurazione di rete**: numero di sottoreti e loro interconnessioni, elenco degli host con i relativi sistemi operativi, servizi attivi, processi vulnerabili, indirizzi IP , e regole di firewall tra sottoreti;
- il **profilo del pentester**: l’insieme di exploit noti all’attaccante, con specifica del servizio/OS su cui ognuno funziona, costo (in termini di rumore o tempo), probabilità di successo e livello di accesso ottenuto (utente o amministratore). Oltre agli exploit remoti, si definiscono le azioni di *privilege escalation* locali disponibili (anche se con precondizioni su processi in esecuzione e probabilità di successo). Infine, si configurano i costi delle scansioni e quali host sono considerati obiettivi sensibili con un certo valore in punti.

L’ambiente è *parzialmente osservabile*: l’agente inizia conoscendo solo informazioni di base e deve usare azioni di scan per rivelare host e servizi. Quando riesce a compromettere un host, ottiene visibilità sui dettagli di quel nodo e su eventuali nuove connessioni raggiungibili da esso. Questo riflette fedelmente un approccio **black box** di pentest: l’attaccante non ha conoscenza interna finché non la guadagna sul campo attraverso la ricognizione attiva. **Non ci sono meccanismi di difesa esplicativi** o randomizzazioni dovute a un difensore: NASim simula un ambiente “statico” con configurazione fissa, dove solo l’attaccante agisce (si assume che il difensore non interferisca, oppure che operi solo implicitamente attraverso i fallimenti di exploit e i firewall). Tuttavia, il simulatore prevede esiti probabilistici: ogni exploit ha una probabilità di successo, quindi l’agente potrebbe dover ripetere tentativi o scegliere percorsi alternativi se un attacco fallisce. Un exploit fallito semplicemente non fornisce accesso ma l’agente può riprovare. L’episodio termina quando l’agente ha conquistato l’intera rete oppure il numero di step() limite stabilito è terminato. In termini di architettura software, NASim implementa un environment simile Gym oppure usando direttamente Gym. Si può inizializzare uno scenario predefinito o caricare da YAML un ambiente personalizzato. L’osservazione dell’agente può essere configurata per essere parziale oppure completa per esperimenti a conoscenza piena.

3.2 Implementazione

Secondo la documentazione ufficiale, un file YAML contiene le seguenti sezioni principali per creare uno scenario:

- **subnets**: lista delle subnet, dove ogni sottorete ha una dimensione specificata (`subnets: [1, 2, 3]`).
- **topology**: matrice di adiacenza simmetrica che indica le connessioni tra le subnet, inclusa una subnet “internet” riservata all’indice 0.
- **os, services, processes**: liste di sistemi operativi, servizi e processi disponibili, usati poi per validare la configurazione degli host.
- **hosts**: con definizione di OS, servizi, processi, firewall host-level e valore positivo per indicare la reward ed eventualmente negativo per rappresentare honeypot.

- **firewall**: specifica delle regole di blocco tra subnet;
- **privilege_escalation**: elenco di exploit locali mappati a processi, costi e livelli di privilegio da ottenere.

Dopo il caricamento (tramite `nasim.load(...)`), NASim costruisce un grafo diretto che rappresenta la rete.

Questo approccio rende NASim uno strumento modulare e riproducibile dove le caratteristiche dello scenario sono completamente parametriche e adatte a studi controllati sul comportamento di agenti mediante varie combinazioni di struttura, valori e vulnerabilità di rete.

Lo scenario utilizzato è uno di quelli già forniti (**medium.yaml**) a cui però è stato manualmente aggiunto un nuovo host che ha value uguale a -100 e quindi rappresenta un honeypot (il nuovo file è **medium_with_honeypout.yaml**). Per testing nello scenario è stato impostato come limite di `step()` 100.000. Inoltre è stato impostato un seed a tutti i diversi elementi del codice (numpy, random, torch e così via) per rendere i risultati uguali per ogni riproduzione.

Gli agenti testati usano visibilità completa (il test in questo caso è white box per permettere il corretto fine-tuning e testing immediato degli agenti) e sono: DQN, Random Agent e QL with Replay Memory. **Per DQN è necessario eseguire un ulteriore script per ottenere un file policy (.pt) il quale è il risultato del training dell'agente e viene poi usato per il suo funzionamento e decision making.**

I test hanno evidenziato che l'agente DQN fatica a vincere ed ha bisogno di un training adeguato ed estensivo per poter vincere lo scenario seppur con un elevato numero di `step()`. Il Random Agent è migliore di gran lunga riuscendo a vincere con meno di 5000 passi. Questo testimonia come l'agente DQN abbia bisogno di un fine-tuning dei parametri: è stato quindi creato uno script che testa tutte le combinazioni dei diversi parametri forniti; non è stato possibile tuttavia completare l'esecuzione dello script considerando l'alto numero di combinazioni e quindi l'alto tempo richiesto. Il QL with Replay avendo la Q-table impiega molto tempo a finire il training quindi è stato usato un basso training step a causa di insufficienti capacità computazionali e tempo. Questo ha portato QL a non essere al pari di DQN o Random Agent.

Risultati DQN:

```
Steps: 98994
Total cost: -99274.0
Goal reached: True
```

Risultati QL with Replay:

```
Steps: 100000
Total cost: -108765.0
Goal Reached: False
```

Risultati Random Agent:

```
Steps: 877
Total cost: -1226.0
Goal reached: True
```

Ricordando che **total cost** non indica la reward ma il costo totale di tutte le azione effettuate per cercare di raggiungere l'obiettivo.

```

FinetuningDQN.ipynb  NetworkSimTestMediumHorizon.ipynb  ObtainPolicyFileDQN.ipynb
+ % □ ▶ C ➤ Code
t = 0
step_limit = 100_000

while not done and not env_step_limit_reached and t < step_limit:
    a = env.action_space.sample()
    # FORZA a essere un int puro!
    if not isinstance(a, int):
        a = int(a)
    _, r, done, env_step_limit_reached, _ = env.step(a)
    total_reward += r
    t += 1

print("Steps:", t)
print("Total reward:", total_reward)
print("Goal reached:", env.goal_reached())

Steps: 877
Total reward: -1226.0
Goal reached: True

FinetuningDQN.ipynb  NetworkSimTestMediumHorizon.ipynb  ObtainPolicyFileDQN.ipynb
+ % □ ▶ C ➤ Code
random.seed(seed)

np.random.seed(seed)

# Torch
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

env = nasim.load('/home/kali/NetworkAttackSimulator/Experiments/medium_with_honeypot.yaml',
dqn_agent = DQNAgent(env,
seed=100,
hidden_sizes=[256, 256],
verbose=False)

dqn_agent.load('/home/kali/NetworkAttackSimulator/Experiments/dqn_medium_with_honeypot.pt')
ret, steps, goal = dqn_agent.run_eval_episode(env, False, 3.01, 'human')
print(goal)
print(steps)
print(ret)
True
98994
-99274.0

```

```

FinetuningDQN.ipynb  NetworkSimTestMediumHorizon.ipynb  ObtainPolicyFileDQN.ipynb
+ % □ ▶ C ➤ Code
    training_steps=2000,
    batch_size=64,
    replay_size=10000,
    final_epsilon=0.01,
    exploration_steps=10000,
    gamma=0.99,
    verbose=False)

# Addestra il DQN per qualche migliaio di passi (esempio rapido, puoi
ql.agent.train()

# Salva il modello
episode_return, steps, done = ql.agent.run_eval_episode(env,
render=False,
eval_epsilon=0.01,
render_mode="readable")

print(episode_return)
print(steps)
print(done)

-108765.0
100000
False

```

Figura 19: Screenshot dei tre test effettuati con i tre agenti su Network Attack Simulator

4 Infection Monkey

4.1 Architettura

Infection Monkey è uno strumento open-source per la simulazione di attacchi informatici (breach and attack simulation) concepito per valutare la resilienza di reti e data center di fronte a violazioni perimetrali e infezioni interne. Nato nel 2016 per iniziativa di Guardicore (oggi parte di Akamai), esso emula il comportamento di un attaccante reale all’interno della rete, consentendo di identificare falliche di sicurezza prima che possano essere sfruttate da attori malevoli. I principali casi d’uso di Infection Monkey includono:

- **Penetration testing automatizzato:** esegue test in modo automatico e continuativo, aiutando a convalidare l’efficacia dei controlli di sicurezza esistenti.
- **Simulazione di attacchi laterali:** valuta fino a che punto un malware o attaccante, una volta ottenuto un primo accesso, potrebbe muoversi lateralmente nella rete compromettendo altri sistemi.
- **Identificazione di vulnerabilità di rete:** fornisce una panoramica dal punto di vista dell’attaccante, evidenziando servizi esposti, credenziali deboli e configurazioni errate sfruttabili per violare la sicurezza.

Architetturalmente, Infection Monkey è composto da due componenti chiave: il Monkey Agent e il Monkey Island. Il Monkey Agent, spesso chiamato semplicemente “Monkey”, è un programma binario, sicuro e che si autopropaga ed esegue la scansione della rete locale, cerca vulnerabilità, si propaga su altri host e simula tecniche di attacco proprio come farebbe un worm malware. Tutto questo senza danneggiare i sistemi: l’unica cosa che fa è lasciare un file che se aperto contiene una scimmia che recita ”non preoccuparti, va tutto bene”. Il Monkey Island è invece il server di comando e controllo con interfaccia web: esso funge da centro di coordinamento, permettendo all’utente di configurare e

monitorare gli agenti e raccogliendo tutte le informazioni che i Monkey Agent scoprono durante la simulazione. Una volta completato un test, il Monkey Island aggrega i risultati e li presenta sotto forma di mappe e report. Vale la pena notare che Infection Monkey è indipendente dall'ambiente in cui opera: può essere utilizzato in reti tradizionali, in container, nonché in cloud pubblici o privati. Questa versatilità ne consente l'impiego in una varietà di scenari per mettere alla prova la sicurezza di sistemi eterogenei.

4.2 Funzionamento Pratico di Infection Monkey

In un utilizzo tipico, l'utente installa **Monkey Island** su una macchina (ad esempio una VM Linux o Windows dedicata) e vi inietta almeno un **Monkey Agent**, che fungerà da “attaccante simulato”. L'agente può essere eseguito in due modalità operative:

- **From Island:** il Monkey Agent viene avviato sulla stessa macchina del Monkey Island. Si simula un attaccante che ha compromesso direttamente il nodo centrale.
- **Manual:** l'agente viene lanciato manualmente su un altro host della rete. Si suppone che l'attaccante abbia già violato un sistema interno e voglia espandersi.

In entrambi i casi, gli agenti comunicano con il server Monkey Island per inviare i dati raccolti e ricevere istruzioni.

Una volta avviato il server Monkey Island (ad esempio lanciando l'AppImage su Linux), l'interfaccia web di Infection Monkey è accessibile tramite browser all'indirizzo <https://localhost:5000>. Al primo accesso, l'utente deve creare credenziali di login per proteggere l'accesso al tool.

L'interfaccia è organizzata in sezioni chiare:

- **Dashboard**
- **Configurazione**
- **Esecuzione (Run Monkey)**
- **Mappa di Infezione**
- **Report di Sicurezza**

Prima di avviare un attacco simulato, è possibile (e consigliato) personalizzare diversi parametri:

- **Network:** definisce l'ambito IP della simulazione e consente di abilitare l'*Agent Scan* per la scansione attiva della rete. È anche possibile escludere specifici IP/host.
- **Credentials:** permette di fornire un elenco di username/password note o di default, per simulare la compromissione tramite credenziali deboli o trapelate (SSH, RDP, SMB, ecc.).
- **Plugins:** Infection Monkey usa plugin modulari per le funzionalità avanzate. Nessun plugin è preinstallato di default. Con l'opzione “Download All Safe Plugins” si possono scaricare quelli sicuri per l'ambiente in uso.

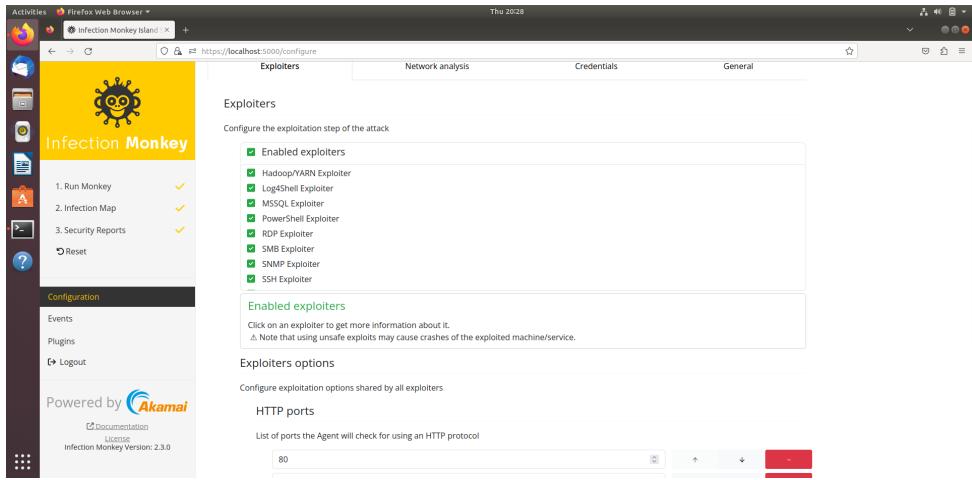


Figura 20: Screenshot dei plugin scaricati

Durante la simulazione, Infection Monkey può impiegare diverse tecniche realistiche:

- **Esecuzione Remota:** su servizi SMB (*SMBexec*), via WMI su host Windows (*WMIexec*), su servizi MSSQL, o vulnerabilità Hadoop.
- **Estrazione Credenziali:** in versioni recenti può recuperare password da browser (Chrome, Edge) per ampliare l'attacco.
- **Payload Simulati:**
 - *Ransomware Simulation*: simula la cifratura di file per testare i sistemi di difesa, senza modificare realmente i dati.
 - *Cryptojacking Module*: esegue calcoli intensivi per emulare un malware che mina criptovalute.

Durante l'esecuzione, ogni host “visto” o compromesso è visualizzato nella **Mappa di Infezione**, con:

- indicazione dei servizi trovati,
- esito dei tentativi di attacco (successo o fallimento).

Al termine della simulazione, viene generato un **Security Report dettagliato** che:

- riassume le vulnerabilità sfruttate,
- categorizza i problemi rilevati (porte aperte, password deboli, patch mancanti, errori di configurazione, ecc.),
- fornisce indicazioni su come mitigare ciascun rischio.

Per i diversi test effettuati abbiamo usato una rete NAT in cui erano presenti diverse macchine ”domestiche” con Windows e due VM vulnerabili per natura: **Stapler1** e **Metasploitable2**. Sono stati scaricati solo i safe plugin e sono state fornite credenziali user e per Metasploitable2 anche credenziali root ed amministratore. Il **Network Scanning** è stato eccelso individuando tutte le macchine effettivamente presenti sulla LAN:

stranamente è uscito anche dalla NAT Network che è stata correttamente configurata.

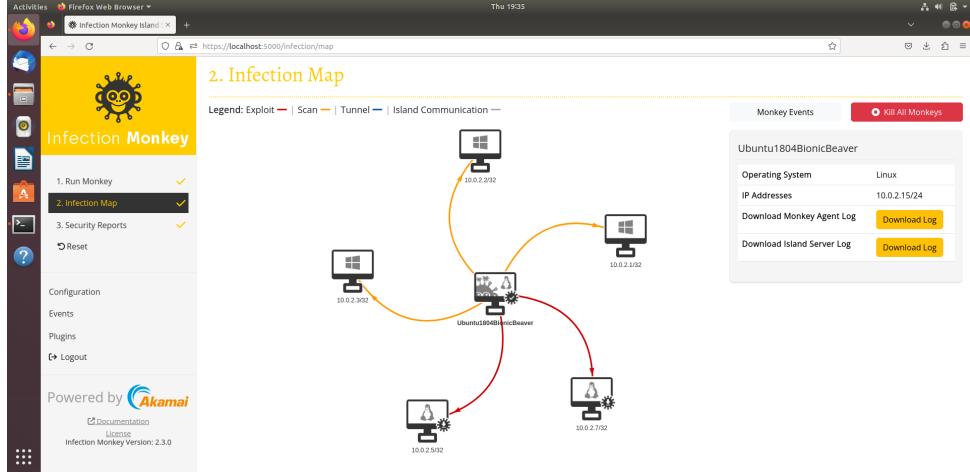


Figura 21: Screenshot della rete LAN scannerizzata da Infection Monkey

Infection Monkey è riuscita ad infettare le due macchine senza però fare molto: ha individuato la possibilità di sfruttare il **protocollo SSH** per eseguire exploit ma si è fermato lì senza individuare alcuna altra vulnerabilità.

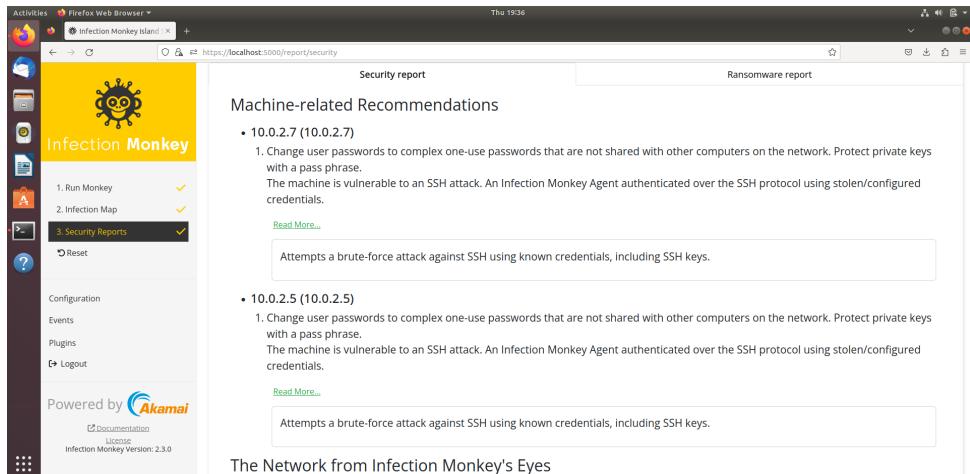


Figura 22: Screenshot delle vulnerabilità individuate

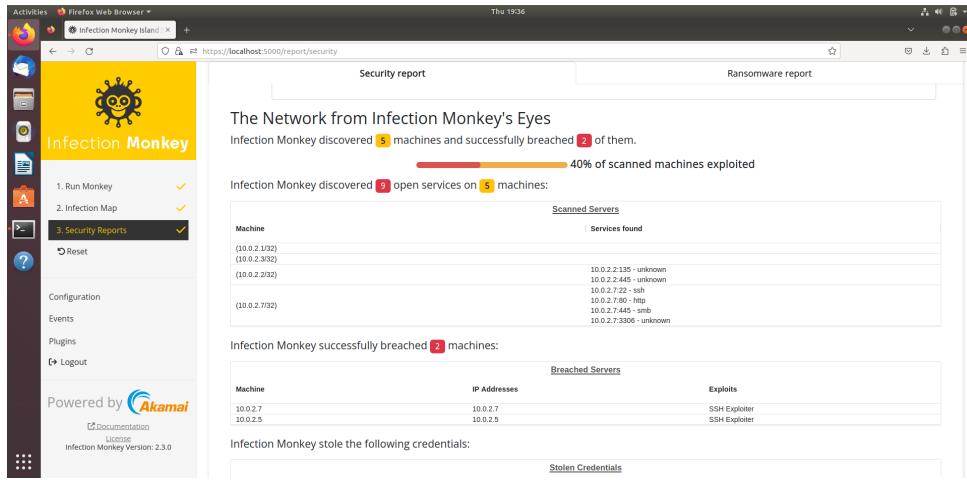


Figura 23: Screenshot dei risultati di attacco

.5 Tentativi e Processo di Apprendimento dei Tool

In questa sezione ci si concentra sulle difficoltà riscontrate durante l'utilizzo dei diversi tool e durante la sperimentazione.

.5.1 CyberBattleSim

Nel corso delle attività di simulazione delle Cyber Battle, ho utilizzato anche CyberBattleSim, un ambiente sviluppato da Microsoft Research che consente la modellazione di reti aziendali e la simulazione di attacchi e difese tramite agenti RL. L'obiettivo era quello di comprendere le dinamiche di attacco, le contromisure difensive e valutare la flessibilità dello strumento rispetto ad altri.

CyberBattleSim è stato usato per costruire una rete complessa simulata, composta da numerosi host con differenti livelli di vulnerabilità, privilegi, servizi attivi e possibilità di escalation. Le sperimentazioni hanno incluso:

- Creazione di una topologia realistica con asset critici, punti di accesso e obiettivi di esfiltrazione.
- Definizione di agenti Red personalizzati basati su Reinforcement Learning (DQN) e con azioni definite a priori nel codice sorgente.

Il generatore di rete di CyberBattleSim ha permesso di creare topologie articolate e modulari, dove è stato possibile modellare vulnerabilità locali, remote, password e accessi privilegiati. La costruzione della rete ha avuto diverse problematiche derivanti dalla mancanza di una documentazione completa che aiutasse a comprendere come strutturare una rete all'interno del tool. Si è resa necessaria la lettura del codice sorgente,

È stato simulato un attacco pressochè realistico: accesso iniziale da nodo vulnerabile, movimento laterale e tentativi di privilege escalation verso asset critici.

L'agente di attacco basato su Deep Q-Learning ha mostrato capacità di esplorazione progressiva della rete, migliorando col tempo il numero di host compromessi e riducendo i tentativi inefficaci. Tuttavia si notano forti diminishing returns: incrementando gli episodi, l'agente non migliora ed anzi o peggiora o raggiunge un limite testimoniano che il training risulta complesso e necessita di reti costruite ad-hoc per poter costruire la

conoscenza dell’agente in maniera incrementale. La rete presentata infatti risulta assai complessa come primissima rete per un agente completamente nuovo e privo di conoscenza. Si è cercato infatti di non esagerare: la rete all’inizio comprendeva Firewall con politiche bloccante ad ogni nodo. Successivamente sono state eliminate e lasciate solo nei nodi più critici e quindi con valore più alto.

L’uso di reward e penalità per azioni ridondanti ha permesso di indirizzare l’agente verso strategie più ottimizzate, come evitare host già compromessi o preferire exploit ad alta probabilità di successo.

.5.2 CybORG

Nel corso dell’analisi e sperimentazione condotta con il framework **CybORG (Cyber Operations Research Gym)**, sono stati esplorati diversi approcci per la simulazione di attacchi e difese in ambienti di rete controllati, con l’obiettivo di valutare l’efficacia e l’interoperabilità degli agenti Red (offensivi) e Blue (difensivi) su scenari complessi.

Il focus è stato in particolare sul **Challenge 4**, uno scenario simulato avanzato dove la rete è popolata da molteplici host, processi e servizi. Gli esperimenti sono stati condotti attraverso:

- Definizione di agenti nuovi per il Blue Team (come BlueDefenderAgent) e utilizzo di agenti già presenti ma customizzati (FiniteStateRedAgent).
- Simulazione passo-passo e monitoraggio dell’interazione tra agenti tramite l’interfaccia CybORG.step().
- Analisi delle risposte del simulatore, in particolare delle azioni classificate come InvalidAction.

L’agente CAGE4CustomRed ha mostrato comportamenti deterministici e ben calibrati e le azioni offensive (come ExploitRemoteService, PrivilegeEscalate) risultavano quasi sempre valide e spesso efficaci, indicando una buona integrazione con la struttura della rete simulata.

L’agente è stato costruito a partire da uno già presente e cioè il FiniteStateAgent che però è stato customizzato nel decision-making modificando le tabelle di probabilità di transizione precedentemente descritte.

Diversamente, l’agente Blue (in particolare BlueDefenderAgent) ha presentato criticità nell’interazione con l’ambiente.

All’inizio la maggior parte delle azioni restituivano un’osservazione contenente InvalidAction, segno che l’agente spesso tentava azioni non ammesse dallo stato attuale della rete o non valide per il contesto.

Dopo diversi tentativi di correzione strutturale (fallback, validazione dei tipi di osservazione, sanitizzazione degli host e così via), il Blue Agent è riuscito a reagire in modo più o meno efficace agli attacchi del Red Agent con azioni di monitoraggio e rimozione del malware piantato dall’agente o restore dell’intero host. **L’obiettivo della challenge di costruire un Blue Agent nuovo e funzionante è stato quindi raggiunto.**

È emerso tuttavia che, mentre il simulatore CybORG è potente e flessibile, richiede una modellazione esplicita e precisa degli agenti, soprattutto per quanto riguarda gli agenti difensivi.

Il confronto tra agenti suggerisce che gli agenti Red partono con un vantaggio intrinseco nella simulazione, mentre gli agenti Blue devono essere progettati con una logica altamente resiliente per non cadere in errori di sincronizzazione o InvalidAction.

Nel caso tuttavia si voglia testare solo il RedAgent è possibile usare uno SleepAgent (che esegue semplicemente l'azione Sleep) come Blue Agent.

.5.3 NetworkAttackSimulator

Nel corso dell’analisi e sperimentazione condotta con il framework NetworkAttackSimulator non ci sono state grandi problematiche: il tool è semplice ma potente e ben documentato. Lo sforzo principale è stato relativo al rendere l’agente DQN più competitivo rispetto al Random Agent ma ci si è resi conto che si necessita di un training e fine-tuning più accurati.

.6 Confronto Trai Tool

Questa sezione confronta i simulatori di attacco informatico **CyberBattleSim**, **NASim** e **CybORG** rispetto a facilità d’uso, realismo, personalizzazione, aderenza agli scenari reali e utilità per un pentester. Si discuterà poi anche di **Infection Monkey**.

Facilità d’uso

NASim è il più semplice da installare e utilizzare, grazie al supporto PyPI e all’interfaccia Gym. **CyberBattleSim** richiede la lettura del codice sorgente e configurazioni minime, ma resta accessibile. **CybORG**, invece, è il più complesso: include componenti avanzate (Docker, Ray, multi-agent) e, se usato in modalità emulativa, richiede risorse cloud e competenze DevOps.

Realismo e astrazione

CybORG è il più realistico, simula scenari attacco-difesa e supporta l’emulazione su infrastruttura reale. **NASim** offre un buon compromesso, includendo reti segmentate, firewall e scansioni, ma con astrazione semplificata. **CyberBattleSim** è il più astratto: non simula utenti, pacchetti o exploit concreti, ma si concentra su movimenti laterali in reti modellate come grafi.

Personalizzazione

NASim eccelle per rapidità: gli scenari sono modificabili in YAML e supportano la generazione automatica. **CyberBattleSim** richiede modifiche in Python, senza supporto a file di scenario o strumenti di generazione. **CybORG** è altamente estendibile ma complesso: la creazione di nuovi scenari richiede la scrittura di classi Python o ricchi file yaml e comprensione approfondita dell’architettura interna.

Utilità per un pentester

Per un **pentester accademico**, **CyberBattleSim** è ideale per prototipazione rapida e studio di algoritmi, ma limitato in realismo. **NASim** è un buon equilibrio per esperimenti realistici ma gestibili. **CybORG** offre il massimo in termini di complessità e completezza ma richiede tempo e competenze per essere sfruttato appieno.

Infection Monkey

Il tool **Infection Monkey** lavora su ambienti reali e permette parametrizzazione ma è molto limitato. L'agente è molto semplice e può fare pochi exploit reali ma la scansione risulta ottima ed accurata. Nonostante esso sia improntato al mondo reale, la sua diretta controparte CybORG, nella sua accezione realistica e non simulata, risulta comunque più ricco e personalizzabile. Questo quindi risulta essere un tool utile se un pentester vuole effettuare una scansione di massa in tempi relativamente brevi con qualche exploit nel mezzo ma poco utile in ambienti davvero professionali.

.7 Installazione tool

.7.1 CyberBattleSim

Per far funzionare correttamente `CyberBattleSim` su una macchina Linux come Kali, si possono seguire i seguenti passaggi:

1. Aggiornamento del sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Installare **Miniconda**, se non già presente. È consigliabile scaricarlo dal sito ufficiale.

3. Clonare il repository ufficiale:

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

4. Creare un nuovo ambiente conda per il simulatore:

```
conda create -n cybersim python=3.9  
conda activate cyberbattlesim
```

5. Installare le dipendenze:

```
pip install --upgrade pip  
pip install -r requirements.txt  
pip install jupyter ipykernel  
pip install asciichartpy pre-commit  
pip install --upgrade pyright  
pip install torch --index-url https://download.pytorch.org/wheel/cpu
```

6. Eseguire lo script di inizializzazione:

```
bash init.sh
```

7. Registrare il kernel per Jupyter: `python -m ipykernel install --user --name cyberbattlesim --display-name "CyberBattleSim"`
8. Installare il pacchetto in modalità `editable` per rendere visibili le sue dipendenze a Jupyter:
`pip install -e .`
9. sostituire dopo aver scaricato la repo learner.py in /CyberBattleSim/cyberbattle/agents/baseline

7.2 CybORG Challenge 4

Per configurare l'ambiente relativo alla challenge CAGE 4, è possibile procedere come segue:

1. Creare ed attivare l'ambiente:

```
conda create -n cage4 python=3.10 -y
conda activate cage4
```

2. Clonare il repository ufficiale:

```
git clone https://github.com/cage-challenge/cage-challenge-4.git
cd cage-challenge-4
```

3. Installare manualmente PyTorch:

```
pip install torch==2.2.0 --index-url https://download.pytorch.org/whl/cpu
```

4. Rimuovere la voce `torch==2.2.0` da `Requirements.txt`, poiché già installata separatamente.

5. Installare le dipendenze:

```
pip install -r Requirements.txt
pip install jupyterlab ipykernel
```

6. Registrare il kernel per Jupyter:

```
python -m ipykernel install --user
--name cage4 --display-name "Python (cage4)"
```

7. Installare il pacchetto in modalità `editable`:

```
pip install -e .
```

8. sostituire dopo aver scaricato la repo FiniteStateRedAgent.py in /CybORG/Agents/SimpleAgents

.7.3 NetworkAttackSimulation (Nasim)

Per far funzionare correttamente **NASim**, si può procedere come segue:

1. Aggiornare il sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Installare **Miniconda**, se non già presente.

3. Creare ed attivare un nuovo ambiente conda:

```
conda create -n nasim python=3.11 -y  
conda activate nasim
```

4. Clonare il repository ufficiale dalla branch **master**:

```
git clone -b master https://github.com/Jjschwartz/NetworkAttackSimulator.git  
cd NetworkAttackSimulator
```

5. Entrare nella cartella **docs** e installare i requisiti della documentazione:

```
cd docs  
pip install -r requirements.txt
```

6. Installare PyTorch per CPU:

```
pip install torch --index-url https://download.pytorch.org/whl/cpu
```

7. Tornare alla cartella principale:

```
cd ..
```

8. Installare le estensioni opzionali di **NASim**:

```
pip install nasim[dqn]  
pip install nasim[test]  
pip install nasim[docs]
```

9. Installare le dipendenze:

```
pip install jupyterlab ipykernel
```

10. Registrare il kernel per Jupyter:

```
python -m ipykernel install --user  
--name nasim --display-name "NetworkAttackSimulator"
```

11. Installare il pacchetto localmente in modalità **editable**:

```
pip install -e .
```

12. sostituire dopo aver scaricato la repo ql_replay_agent.py in /nasim/agents

.7.4 Infection Monkey

Per configurare correttamente Infection Monkey su una macchina virtuale con **Ubuntu 18.04 Desktop (Bionic Beaver)**, è possibile seguire la seguente procedura:

1. Scaricare Ubuntu 18.04 Desktop Bionic Beaver.

2. In caso di problemi con il terminale che non si avvia, seguire il workaround:

Settings → Language & Settings → Cambiare English (United States) in English (Canada) e riavviare.

3. Entrare in modalità TTY con Ctrl + Alt + F3 ed elevare l'utente attuale a sudoer:

```
sudo usermod -aG sudo ubuntu
```

4. Riavviare il sistema.

5. Aggiornare i pacchetti:

```
sudo apt update && sudo apt upgrade -y
```

6. Rimuovere eventuali installazioni precedenti di MongoDB:

```
sudo systemctl stop mongod
sudo apt purge mongodb mongodb-server mongodb-server-core mongodb-clients
sudo rm -rf /var/lib/mongodb
sudo rm -rf /etc/mongodb.conf
```

7. Aggiungere il repository di MongoDB 4.2:

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
sudo apt update
```

8. Installare MongoDB e strumenti associati:

```
sudo apt install -y mongodb-org
sudo dpkg --purge mongo-tools
sudo rm -f /usr/bin/bsodump
sudo apt-get clean
sudo apt-get update
sudo apt-get install -y mongodb-org-tools mongodb-org
sudo dpkg --configure -a
sudo apt-get install -f
sudo systemctl restart mongod
```

9. Scaricare la versione AppImage di Infection Monkey:

```
https://github.com/guardicore/monkey/releases/tag/v2.3.0
```

10. Posizionare il file .AppImage nella ~ /home ed eseguire:

```
chmod u+x InfectionMonkey-v2.3.0.AppImage
./InfectionMonkey-v2.3.0.AppImage
```

Nota: prima di ogni esecuzione è consigliato riavviare il servizio MongoDB:

```
sudo systemctl restart mongod
```

11. Accedere all'interfaccia web:

```
https://localhost:5000 (da browser Firefox)
```

12. In **Configurations** → **Network**:

- Abilitare la casella **Agent Scan**
- Escludere eventuali IP specifici dallo scan

13. Nella sezione **Configurations** → **Plugins**, scaricare tutti i plugin contrassegnati come **SAFE**.

14. Inserire manualmente credenziali comuni da provare, ad esempio:

```
"user,user"    "root,toor"    "SHayslett,SHayslett"
```

15. Assicurarsi che tutte le VM coinvolte siano collegate alla stessa rete **NAT Network**.

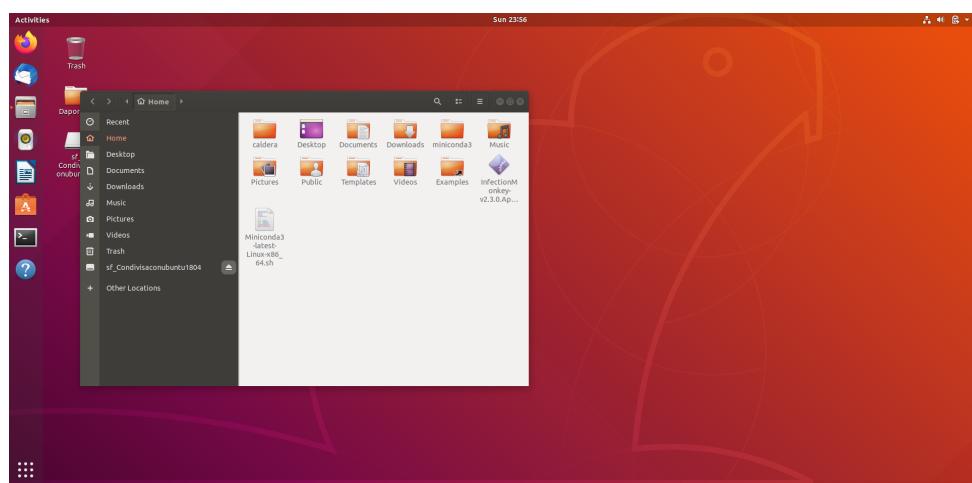


Figura 24: Screenshot del sistema operativo Ubuntu emulato tramite Oracle VirtualBox

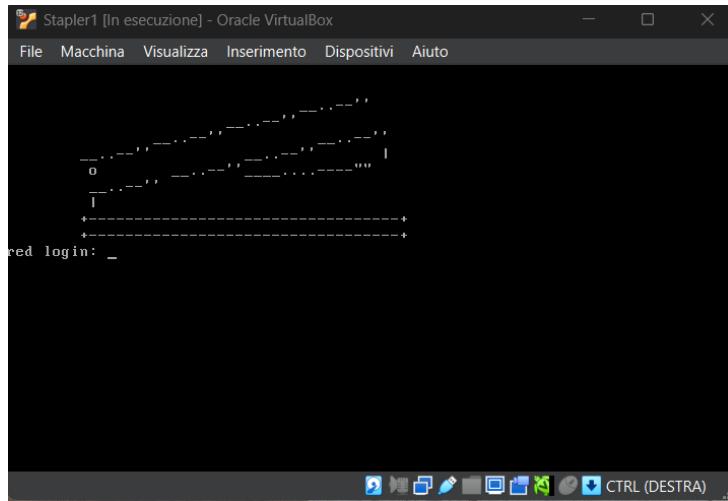


Figura 25: Screenshot della macchina vulnerabile Stapler1 emulata tramite Oracle VirtualBox

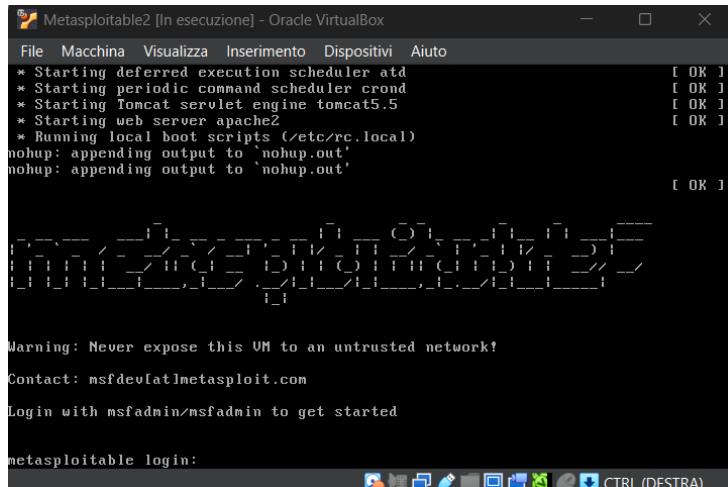


Figura 26: Screenshot della macchina vulnerabile Stapler1 emulata tramite Oracle VirtualBox

.8 Script Forniti

- **TestGioNetExtraVulnsForReport.ipynb** contiene l'esecuzione dell'esperimento per CyberBattleSim;
- **CybORGTests.ipynb** contiene l'esecuzione dell'esperimento per CybORG Challenge 4;
- **Infection Monkey Island Server.csv** è il log della scansione di Infection Monkey;
- **FinetuningDQN.ipynb** è il file per fare il fine-tuning dei parametri di DQN per NetworkAttackSimulator;
- **NetworkSimTestMediumHoneypot.ipynb** contiene l'esecuzione dell'esperimento per NetworkAttackSimulator il quale però necessita del file policy .pt per DQN;

- **ObtainPolicyFileDQN.ipynb** contiene il codice per ottenere la policy .pt per DQN per NetworkAttackSimulator;
- **TestQLReplay.ipynb** contiene l'esecuzione dell'esperimento per NetworkAttackSimulator usando QL with Replay Memory che necessita del file scenario medium_with_honeypot.yaml;
- **medium_with_honeypot.yaml** è il file scenario per NetworkAttackSimulator;
- **learner.py** che è una versione modificata di un file del codice sorgente di CyberBattleSim da aggiungere dopo aver scaricato la repo in /CyberBattleSim/cyberbattle/agents/baseline
- **FiniteStateRedAgent.py** che è una versione modificata di un file del codice sorgente di CybORG Challenge 4 da aggiungere dopo aver scaricato la repo in /CybORG/Agents/SimpleAgents
- **ql_replay_agent.py** che è una versione modificata di un file del codice sorgente di NetworkAttackSimulation da aggiungere dopo aver scaricato la repo in /nasim/agents

Si è resa necessaria la modifica di tre file del codice sorgente per miglioramenti relativi a stampa dei risultati (learner.py) oppure per alcune piccole correzioni per evitare errori (FiniteStateRedAgent.py e ql_replay_agent.py). I file vengono forniti e dovrebbero essere inseriti dove indicato nelle cartelle delle relative repo per il corretto funzionamento. Potrebbe essere necessario modificare i path di file in alcuni script soprattutto quelli di NetworkAttackSimulator.