Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп’ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Тестування безпеки комп'ютерних систем з використанням навчання з підкріпленням
(тема)

Виконав: студент 2 курсу, групи СКСм-20-2

_____Сімакін В.А._____
(прізвище, ініціали)

Спеціальність *123 _Комп’ютерна інженерія*

Тип програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

*Спеціалізовані комп’ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи _доцент Адамов О.С._
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ___(підпис)___ Чумаченко С.В.
(прізвище, ініціали)

2022 р

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123_ Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«__25__» ___03____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сімакіну Вячеславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Тестування безпеки комп'ютерних систем з використанням навчання з підкріпленням

Penetration testing with Reinforcement learning

затверджена наказом по університету від «_24_» __03__2022 р. № _408 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20.05.2022

3. Вихідні дані до роботи (проекту)

Теоретичні відомості про тестування безпеки

Теоретичні відомості про навчання з підкріпленням

Мова програмування Python

Середовище розробки Visual Studio Code

4. Перелік питань, що потрібно опрацювати у роботі

Аналіз предметної області.

Симулятор мережі

Тестування на проникнення за допомогою навчання з підкріпленням

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _22 слайдів_ _____

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 20.01.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Отримання завдання	20.01. 2022-21.01.2022	
2	Огляд літератури за темою роботи	22.01. 2022-01.02.2022	
3	Розробка архітектури симулятора мережі та агентів штучного інтелекту	02.02.2022-14.02.2022	
4	Розробка симулятора мережі	15.02. 2022-03.03.2022	
5	Розробка моделей агнетів тетсування безпеки	04.03. 2022-23.03.2022	
6	Тестування програмної реалізації машинних алгоритмів	24.03. 2021-10.04.2022	
7	Висновки	11.04. 2022-12.04.2022	
8	Оформлення пояснювальної записки	13.04. 2022-08.05.2022	

Студент _____
(підпис)

Керівник роботи (проекту) _____ доц. каф. АПОТ Адамов О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 83 с., 26 рис., 5 табл., 33 джерел.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ, БЕЗПЕКА, PYTHON, PENETRATION TESTING, REINFORCEMENT LEARNING, МЕРЕЖА, СИМУЛЯТОР

Метою даної роботи є розробка програмного забезпечення для автоматичного тестування на проникнення, та симулятора мережі для практики тестування, що дозволяє прискорити виконання тестування на проникнення та навчання цьому процесу програмам, що основані на машинному навчанні.

У процесі виконання даної роботи було розроблено програму, яка дозволяє, використовуючи спеціальний алгоритм, виконувати тестування на проникнення, до розробленого симулятора для оцінки швидкості алгоритмів.

Вирішено задачу розробки симулятора для машинного навчання програм для тестування на проникнення, що засновані на машинному навчанні. Розроблено програму для автоматичного тестування мереж.

ABSTRACT

Explanatory note contains: 83 pages, 26 figures, 5 tables, 33 sources according to the list of links.

SOFTWARE, TESTING, SECURITY, PYTHON, PENETRATION TESTING, REINFORCEMENT LEARNING, NETWORK, SIMULATOR.

The purpose of this work is to develop software for autonomous penetration testing, and a network simulator for testing practice, which allows you to accelerate the implementation of penetration testing and learning this process programs based on machine learning.

In the process of performing this work, a program was developed that allows, using a special algorithm, to perform penetration testing, both to real machines and to simulators to assess the speed of algorithms.

The problem of developing a simulator for machine learning programs for penetration testing based on machine learning has been solved. A program for automatic network testing has been developed.

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ 11
1.1 Тестування на проникнення 11
1.2 Інструменти для тестування 122
1.3 Machine Learning 122
1.4 Автоматизоване тестування на проникнення: за допомогою MDP 125
1.5 Автоматизоване тестування на проникнення: за допомогою POMDP 126
1.6 Автоматизоване тестування на проникнення: коли модель світу невідома . 127
2 ОПИС ВИКОРИСТОВУВАНИХ ПРОГРАМНИХ ЗАСОБІВ 199
2.1 Мова програмування Python 199
2.1.1 Використані бібліотеки ..... **Error! Bookmark not defined.**20
2.2 Середовище розробки Visual Studio 20
3 СИМУЛЯТОР МЕРЕЖІ 2222
3.1 Вплив оточення на штучний інтелект 2222
3.2 Структура симулятора мережевих атак ..... **Error! Bookmark not defined.**23
3.2.1 Підмережі ..... **Error! Bookmark not defined.**24
3.2.2 Топологія ..... **Error! Bookmark not defined.**25
3.2.3 Машини ..... **Error! Bookmark not defined.**25
3.2.4 Сервіси ..... **Error! Bookmark not defined.**26
3.2.5 Брандмауери ..... **Error! Bookmark not defined.**28
3.3 Оточення агенту ..... **Error! Bookmark not defined.**29
3.4 Реалізація ..... **Error! Bookmark not defined.**33
3.5 Продуктивність мережі ..... **Error! Bookmark not defined.**38
4 ТЕСТУВАННЯ НА ПРОНИКНЕННЯ ЗА ДОПОМОГОЮ НАВЧАННЯ З ПІДКРІПЛЕННЯМ 2242
4.1 Навчання з підкріпленням 2242
4.2 Алгоритм навчання з підкріпленням 2244
4.3 Тестування на проникнення на основі навчання з підкріпленням 2247
4.4 Тестування моделей 2247

ВИСНОВКИ 65
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 66
ДОДАТОК А 669
ДОДАТОК Б 80

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система;

ШІ – штучний інтеллект;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних;

ВМД – верхнімі межі довіри;

ML – Machine Learning(машинне навчання);

RL – Reinforcement learning(навчання з підкріпленням);

MDP – Markov decision process(Марківські процеси прийняття рішень);

POMDP – Partially observable Markov decision process(часткого обізрювані Марківські процеси прийняття рішень);

DMZ – Demilitarized Zone(демілітаризована зона);

DQL – Deep Q-Learning(глибоке Q-навчання);

GUI – Graphical User Interface(графічний інтерфейс користувача);

ВСТУП

Тестування на проникнення передбачає виконання контрольованої атаки на комп’ютерну систему з метою оцінки, що вона є захищеною. В даний час це один з ключових методів, який використовують організації для посилення свого захисту від кіберзагроз.

Однак тестування на проникнення в мережу вимагає значних обсягів знань для отримання гарних результатів через що й зростає нестача кваліфікованих фахівців з кібербезпеки. Один із шляхів вирішення цієї проблеми – це застосовування методів штучного інтелекту (ШІ) до домену кібербезпеки для автоматизації процесу тестування на проникнення. Сучасні підходи до автоматизованого тестування на проникнення спираються на методи, які вимагають моделі які засновані на результатах вразливостей, однак ландшафт кібербезпеки швидко змінюється, так швидко як розробляється нове програмне забезпечення та нові вектори атак, що робить створення та підтримку сучасних моделей - проблемною.

Щоб спробувати вирішити проблему нинішніх методів для виконання цього проекту буде досліджено застосування навчання з підкріпленням (RL) у автоматизованому тестуванні на проникнення. Навчання з підкріпленням - це метод оптимізації ШІ, який має ключову перевагу у тому, що для вироблення атаки йому не потрібна модель середовища, та натомість обирає найкращу політику атаки завдяки взаємодії з навколишнім середовищем. На першому етапі цього дослідженні буде розроблений та побудуваний швидкий та легкий вихідний симулятор мережевих атак ,що може використовуватися для підготовки та тестування автономних агентів для тестування на проникнення.

Це буде зроблено за допомогою кадрування тестування на проникнення як процес прийняття Марківських рішень (MDP) з відомою конфігурацією

мережі як стану, доступними скануваннями та експлуатаціями як дії, винагорода визначається вартістю машин у мережі та використанням недетермінованих дій для моделювання результатів сканування та проникнення до машин.

На другому етапі проекту буде використано мережеву атаку на симулятор для дослідження застосування RL для тестування на проникнення. Перевірено стандартний Q-Learning алгоритм навчання з підкріпленням, що використовує як табличний, так і на основі нейронних мереж підхід. Буде виявлено, чи зможуть як табличні, так і нейромережеві алгоритми RL знайти оптимальні шляхи атак для цілого ряду різних топологій та розмірів мережі в рамках змодельованого середовища. Ця знахідка надасть певне обґрунтування для використання RL у тестуванні на проникнення.

Універсальність RL для розв’язання задач, де модель невідома, надає перевагу у мінливому характеру кібербезпеки і може запропонувати собою цінний інструмент зменшення навантаження на фахівців з кібербезпеки.

1 АНАЛІЗ ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

1.1 Тестування на проникнення

Penetration testing або тестування на проникнення існує вже більше чотирьох десятиліть і є критичним процесом у розробці безпечних кібер-систем. Тестування на проникнення передбачає виконання контрольованої атаки на частину програмного забезпечення або мережі для оцінки безпеки. Процес тестування, як правило, коли застосовується до мережі, поділяється на послідовність етапів для методичної оцінки даної системи. Конкретні кроки можуть варіюватися від атаки до атаки, але загалом тестування на проникнення спочатку передбачає збір інформації, метою якої є пошук вразливості (вада в системі) в мережі, наступною є атака та проникнення туди, де виявлено вразливість. Потім, нещодавно отриманий доступ використовується для повторення усіх кроків до досягнення бажаної цілі. Етап збору інформації, як правило, передбачає використання таких інструментів, як моніторинг трафіку, сканування портів та виявлення операційної системи (ОС) для того, щоб зібрати відповідну інформацію, за якою можна визначити, чи містить система вразливість, що можна використати. Фаза атаки та проникнення передбачає експлуатацією програми або певних даних, щоб скористатися виявленою вразливістю та викликати небажану поведінку в системі з кінцевою метою скомпрометувати ціль і отримання привілейованого доступу до неї. Після успішної атаки певна послідовність дій та атак може бути повідомлена та використана системними адміністраторами та розробники для виправлення вразливості.

Хоча системи та мережі, які оцінюються за допомогою такого тестування, можуть різнитися надзвичайно, у кожному випадку при проведенні атак дотримуються однакових загальних кроків. Це дозволило розробкам ряду інструментів, зробити тестування на проникнення більш ефективним.

1.2 Інструменти для тестування

Індустрія кібербезпеки - це дуже велика та активна спільнота, де існує безліч інструментів які існують для сприяння тестуванню на проникнення, тут ми в основному зупинимось на найбільш часто використовуваних інструментах для мережевого тестування.

На етапі збору інформації метою є пошук корисної інформації, що стосується мереж. Це зазвичай робиться за допомогою мережевого сканера. Найвідомішим мережевим сканером є Nmap. Nmap надає користувачеві таку інформацію як ОС, відкриті порти та служби, які зараз працюють у системі. Ця інформація може бути використана для перевірки на наявність вразливостей у системі за допомогою сканерів вразливостей, таких як Nessus та, безкоштовної альтернативи, OpenVAS. Ці сканери можна використовувати для визначення, чи існує вразливість, яку слід використовувати.

Як тільки інформація про вразливості в системі буде зібрана, в хід ідуть фреймворки пентестування, такі як Metasploit. Метасплойт - це сукупність інструментів та відомих вразливостей, які можна використовувати в одному середовищі .Metasploit - це проект з відкритим кодом, який був розпочатий у 2003 році та придбаний Rapid7 у 2007 році. І регулярно оновлюється новими вразливостями та інструментами. Користувач може використовувати зібрану інформацію на етапі збору інформації для пошуку та використання вразливостей від Metasploit, що дозволяє користувачеві зосередитися на тактичній операції (яку вразливість використовувати) ніж робити зусилля на технічному рівні (як знайти та розробити вразливість). Ця автоматизація на технічному рівні дозволяє значно збільшити ефективність під час пентестування.

1.3 Machine Learning

Машинне навчання (Machine Learning, ML) – це галузь штучного

інтелекту (ШІ) та комп’ютерних наук, яка зосереджена на використанні даних та алгоритмах для імітації способу навчання людей, поступово покращуючи його точність.

Першу програму на основі алгоритмів, здатних самонавчатися, розробив Артур Самуель (Arthur Samuel) в 1952 році, призначена вона була для гри в шашки. Самуель дав і перше визначення терміну «машинне навчання»: це «область досліджень розробки машин, які не є заздалегідь запрограмованими». Більш точне визначення терміну «навчання» дав набагато пізніше Т. М. Мітчелл: кажуть, що комп'ютерна програма навчається на основі досвіду E по відношенню до деякого класу задач T і міри якості P, якщо якість вирішення завдань з T, виміряний на основі P, поліпшується з набуттям досвіду E.

Вже в 1957 році була запропонована перша модель нейронної мережі, що реалізує алгоритми машинного навчання, схожі на сучасні. В даний час ведеться розробка найрізноманітніших систем машинного навчання, призначених для використання в таких технологіях майбутнього, як Інтернет Речей, Промисловий Інтернет Речей, в концепції «розумне» місто, при створенні безпілотного транспорту і в багатьох інших.

Розрізняють два типи машинного навчання. Навчання по прецедентах, або індуктивне навчання, засноване на виявленні загальних закономірностей по приватним емпіричним даним. Дедуктивне навчання передбачає формалізацію знань експертів і їх перенесення в комп'ютер у вигляді бази знань. Дедуктивне навчання прийнято відносити до області експертних систем, тому терміни машинне навчання і навчання по прецедентах можна вважати синонімами. Цей метод навчання зараз є доволі популярним, а ось експертні системи переживають кризу. Бази знань, що лежать в їх основі, важко узгоджувати з реляційною моделлю даних, тому промислові СУБД неможливо ефективно використовувати для наповнення баз знань експертних систем.

Навчання по прецедентах, в свою чергу, поділяють на три основних типи:

– контрольоване навчання, або навчання з учителем (supervised learning);

– неконтрольоване навчання (unsupervised learning), або навчання без

учителя;

– навчання з підкріпленням (reinforcement learning).

Крім названих, розробляються і інші методи навчання: активне, многозадачне, різноманітне, трансферне і т.д. Особливо успішно розвивається в останні роки «глибоке навчання», при використанні якого можуть успішно поєднуватися алгоритми навчання з вчителем і без вчителя.

Машинне навчання знаходиться на стику математичної статистики, методів оптимізації та класичних математичних дисциплін, але має також і власну специфіку, пов'язану з проблемами обчислювальної ефективності та перенавчання. Багато методів індуктивного навчання розроблялися як альтернатива класичним статистичним підходам. Велика кількість методів тісно пов'язана з витягуванням інформації та інтелектуальним аналізом даних.

Машинне навчання - не тільки математична, а й практична, інженерна дисципліна. Чиста теорія, як правило, не приводить відразу до методів і алгоритмів, які можуть застосовуватися на практиці. Щоб змусити їх добре працювати, доводиться винаходити додаткові евристики, що компенсують невідповідність зроблених в теорії припущень умовам реальних завдань. Практично жодне дослідження в машинному навчанні не обходиться без експерименту на модельних або реальних даних, що підтверджує практичну працездатність методу.

Систему навчання алгоритму машинного навчання розбивають на три основні частини:

– Процес прийняття рішень: загалом алгоритми машинного навчання використовуються для прогнозування або класифікації. На основі деяких вхідних даних, які можуть бути позначені або не позначені, ваш алгоритм створить оцінку щодо шаблону в даних.

– Функція помилки: функція помилки служить для оцінки передбачення моделі. Якщо є відомі приклади, функція помилки може провести порівняння для оцінки точності моделі.

– Процес оптимізації моделі: якщо модель може краще відповідати

точкам даних навчального набору, то ваги коригуються, щоб зменшити невідповідність між відомим прикладом та оцінкою моделі. Алгоритм буде повторювати цей процес оцінки та оптимізації, оновлюючи ваги автономно, доки не буде досягнуто поріг точності.

1.4 Автоматизоване тестування на проникнення: за допомогою MDP

Одним з підходів до моделювання та планування атак на систему є використання Марківських процессів прийняття рішень (MDP) для моделювання середовища. MDP — це загальна структура для моделювання дискретних задач прийняття рішень в умовах невизначеностях. Визначается MDP зазвичай як кортеж {$\boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{T}}$}, де $\boldsymbol{\mathcal{S}}$ – простір станів, $\boldsymbol{\mathcal{A}}$ – простір дій, $\boldsymbol{\mathcal{T}}$ – модель переходів $\boldsymbol{\mathcal{T}}$ (s, a, s') = P (s'| s,a), а $\boldsymbol{\mathcal{R}}$ — функція винагороди $\boldsymbol{\mathcal{R}}$ (s, a). У кожен момент часу t система буде в деякому стані, s ∈ $\boldsymbol{\mathcal{S}}$, і агент виконає дію, а ∈ $\boldsymbol{\mathcal{A}}$,що призводить до двох речей: (1) перехід до нового стану s', де новий стан визначається функцією переходу $\boldsymbol{\mathcal{T}}$, (2) винагорода, яка визначається функцією винагороди, $\boldsymbol{\mathcal{R}}$. Метою агента що намагається розв'язати MDP – є знаходження оптимального відношення з $\boldsymbol{\mathcal{S}}$ до $\boldsymbol{\mathcal{A}}$, щоб максимізувати загальну накопичену винагороду. Це відношення відоме як політика прийняття рішень π.

При застосуванні до пентестування простір станів MDP стає можливими конфігураціями цільових машин або мережі, дії є можливими експлойтами або доступними скануваннями а винагорода буде залежати від вартості дії та вартості винагороди, отриманої, після успішно скомпрометованої системи.

Поки що були лише обмежені застосування MDP для автоматизованого пентестування. Один з таких підходів - повне ігнорування конфігурації цільової системи і натомість атакуючий покладається на формулювання невизначеності у вигляді можливих результатів дій. Атаки потім плануються на основі ймовірності успіху атаки, де кожна дія отримує ймовірність успіху на основі попереднього затосування даної дії. Такий підхід додає деякий недетермінізм.

Основною перевагою цього підходу є те, що він дозволяє моделювати невизначеності атакуючого, та є можливим для обчислення. Однак цей підхід не враховує відомі знання про конфігурацію системи, що є ключовим етапом ефективного тестування на проникнення, а натомість розглядає всі машини як ідентичні. Крім того, це також вимагає попереднього знання ймовірності результату атаки (модель переходу) перед тим, як його можна буде використовувати, і ці ймовірності можуть сильно відрізнятися залежно від систем, проти яких він використовується (наприклад, Windows або Linux) і буде змінюватися з часом по мірі появи нового програмного забезпечення та розроблених вразливостей.

1.5 Автоматизоване тестування на проникнення: за допомогою Марківського процесу прийняття рішень, що частково спостерігається

Інший підхід до автоматизації пентестування спрямований на вирішення припущення про повне знання мережі, що необхідні для графів атак, та враховуючи при цьому невизначеності атакуючого для моделюванні проблеми пентестування як POMDP.

POMDP, або частково спостережуваний процес прийняття рішень Маркова, є MDP, у якому є невизначеність щодо точного стану, в якому перебуває система, і тому вона моделює поточний стан як розподіл ймовірностей за всіма можливими станами. POMDP зазвичай визначаються кортежами{$\boldsymbol{S}$, $\boldsymbol{A}$, Ω, $\boldsymbol{T}$,$\boldsymbol{O}$, $\boldsymbol{R}$, bo} де $\boldsymbol{S}$, $\boldsymbol{A}$, $\boldsymbol{T}$ і $\boldsymbol{R}$ такі самі, як у MDP, а Ω — простір спостереження, $\boldsymbol{O}$ — функція спостереження $\boldsymbol{O}$ (s’, a, o) = P (o|s’,a) і є початковим розподілом ймовірностей за станами. Подібно до MDP, на кожному кроці часу буде перехід в новий стан s’ і винагорода, але, крім того, кожен крок також призведе до спостереження, o ∈ Ω, що визначається функцією спостереження $\boldsymbol{O}$. Мета є те сама, що і для MDP, яка полягає в пошуку оптимальної політики прийняття рішень π*.

При застосуванні до пентестування простір станів POMDP стає

можливими конфігураціями цільової машини або мережі, дії є можливими експлойтами або доступними скануваннями, простір спостереження — це можлива інформація, яка отримується під час експлойту або сканування (наприклад, відкриті порти, невдача/успіх використання), тоді як винагорода буде залежати від вартості дії та вартості винагороди, отриманої, після успішно скомпрометованої системи. Використання підходу POMDP дозволяє моделювати багато відповідних властивостей злому в реальному світі, оскільки під час реального нападу зловмисник мав би неповне знання про те, яка насправді конфігурація системи, а також те, чи будуть успішні певні експлойти або сканування. Крім того, це також означає, що цей самий підхід автоматичного пентестування можна використовувати для перевірки системи, навіть якщо система змінюється, оскільки знання про точну конфігурацію системи не припускається. Саме це відрізняється від графів атак, які потрібно оновлювати щоразу, коли щось змінюється в системі.

Підхід POMDP є багатообіцяючим, оскільки він більш точно моделює реальний світ. Однак у зловмисника є одна критична проблема: вирішення на основі POMDP не дуже добре масштабується і швидко стає нездійсненними з точки зору обчислень по мірі зростання розміру простору станів. В підхідах деяких дослідників їм довелося підійти до тесту на проникнення в мережу розкладаючи мережі на окремі POMDP для окремих цільових машин. Так, хоча підхід був більш реалістичним з точки зору зловмисників, зараз він не використовується через занадто дорогу обчислювальну вартість.

1.6 Автоматизоване тестування на проникнення: коли модель світу невідома

І MDP, і POMDP забезпечують загальну структуру для моделювання притаманної невизначеності при виконанні тестування на проникнення та мають ряд доступних методів для їх вирішення. MDP — це простіша версія POMDP, ключовою відмінністю якої є відсутність невизначеності поточного

стану довкілля (вона повністю спостерігається). Перевага MDP полягає в тому, що вони набагато більш піддатні для обчислень у порівнянні з POMDP, і існує багато ефективних алгоритмів їх вирішення. Таке підвищення ефективності дозволяє їм бути кориснішими на практиці. Однак ця ефективність коштує деякими здатностями моделювати невизначеності. Основною формою невизначеності, яка залишається для MDP, є невизначеність, пов'язана з недетермінованими діями, продиктована функцією переходу або моделлю світу.

Одним з методів, який можна використовувати для пошуку оптимальної політики для MDP, є навчання з підкріпленням (RL). RL використовує зразки, згенеровані в результаті взаємодії з середовищем щоб оптимізувати свою продуктивність. Основними перевагами RL перед класичними підходами до планування є його здатність обробляти великі середовища та ситуації коли модель середовища невідома або вирішення з використанням моделі недоступне через його компютерну неітерпритованість.

Для тестування на проникнення, через складну природу комп’ютерних систем є велика проблема створити та підтримувати точну модель того, як експлойти вплинуть на будь-яку систему. Це пов’язано з постійною еволюцією атак і самих систем. Ця властивість робить пентестування хорошим кандидатом для використання RL, оскільки можна визначити тестування на проникнення як MDP, але з використанням RL нам не потрібна модель переходів. Головна проблема, з якою RL стикається, полягає в тому, що для навчання потрібно багато взаємодій із середовищем для напрацювання оптимальної політики. Ця особливість привела до багатьох успішних застосувань RL в імітованих або ігрових середовищах, де агенти RL можуть швидко взаємодіяти з оточенням.

2 ОПИС ВИКОРИСТОВУВАНИХ ПРОГРАМНИХ ЗАСОБІВ

2.1 Мова програмування Python

Python – динамічна інтерпретована об’єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Python – це універсальна мова, що широко використовується в усьому світі для самих різних цілей – бази даних і оброблення текстів, вбудовування інтерпретатора в ігри, програмування GUI, швидке створення прототипів (RAD) програмування Internet і Web додатків – серверних (CGI), клієнтських (роботи), Webсерверів і серверів додатків.

Серед переваг мови Python можна виділити переносимість написаних програм, на комп’ютери різної архітектури та з різними операційними системами, лаконічність запису алгоритмів, можливість отримати ефективний код програм за швидкістю виконання. Зручність мови Python основане на тому, що вона є мовою високого рівня, має набір конструкцій структурного програмування та підтримує модульність.

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи

з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

– відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування.

Один з можливих недоліків Python - швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп’ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше. Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

2.1.1 Використані бібліотеки

В цій роботі було використано декілька сучасних python бібліотек, а саме:

– numPy для швидких обчислень на основі масивів;
– Matplotlib і NetworkX для візуалізацій;
– Keras Neural Network library, що працює на основі TensorFlow що дає можливість легко створити нейронну мережу, що є частиною одного з використаних алгоритмів.

2.2 Середовище розробки Visual Studio

Для написання програми використаємо середовище розробки Visual Studio. Microsoft Visual Studio – лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, Android, IOS, .NET Compact Framework і Silverlight. Підтримує наступні мови: Visual Basic, C ++, C #, F #.

Для написання програми було обрано саме Visual Studio через його основні переваги, такі як CodeAnilizer, що допомагає знайти помилки в коді, технологія автодоповнення IntelliSense, яка дописує назву функції при вводі початкових літер.

Хоч visual studio не підтримує мову програмування python у своїй початковій збірці, але завдяки можливості встановлювати плагіни є можливість створити потужний дует для розробки завдяки можливостям мови python та зручності visual studio.

Отже, за допомогою середовища розробки visual studio та мови програмування Python можна легко створювати, запускати, тестувати та відстежувати власні багато-функціональні додатки.

3 СИМУЛЯТОР МЕРЕЖІ

3.1 Вплив оточення на штучний інтелект

Останнім вдосконаленням методів ШІ значною мірою посприяло створення добре відомих, оточень для навчання. Ці оточення можуть мати різну форму залежно від домену програми, наприклад, існує середовище аркадного навчання для тестування узагальнених алгоритмів навчання з підкріпленням та набір даних ImageNet для комп'ютерного зору. Ці оточення стають тестовим майданчиком, що дозволяє дослідникам порівнювати результативність алгоритмів між собою та за часом. В даний час для мережі тестування на проникнення не існує легкого, вільно доступного еталону, для якого можна використовувати розробклені автоматизовані алгоритміи тестування на проникнення. Для мережевого тестування, щоб воно було реалістичним, еталон повинен мати форму симулятора мережі, яка дозволить агентам впливати на мережу шляхом сканування та виконання атак.

В даний час існує безліч широко використовуваних та вільно доступних симуляторів мережевого трафіку, таких як NS3 та mininet. Ці тренажери мають невелику вагу, здатні ефективно працювати на одній машині, і здатні імітувати топологію мережі та трафіку з високою точністю за допомогою мінімальної віртуалізації фактичної ОС. Однак вони не дозволяють моделювати атаку на мережу через запуск експлойтів та отримання доступу до машин і тому не придатні для оцінки ефективності тестування на проникнення.

Інший варіант - використання мережі віртуальних машин (ВМ). Такий підхід має високе східство з реальним світом, а також є гнучким. Основним недоліком використання віртуальних машин є відносно висока обчислювальна вартість запуску кожної ВМ, що може уповільнити навчання певних типів алгоритмів ШІ, таких як навчання з підкріпленням, а також значна обчислювальна потужність, необхідна для роботи більших мереж. Найкращою на даний момент доступною опцією є система Core Insight. Ця система моделює

мережу на рівні ОС, але підтримує лише підмножину системних викликів, подібну до симуляторів мережевого трафіку, таким чином система може масштабуватися до мереж сотень машин на єдиний комп’ютер. Однак це програмне забезпечення не є відкритим або безкоштовним для громадськості і коштує десяткитисяч доларів за ліцензію, про що для багатьох дослідників не може бути й мови.

У цій роботі було розроблено та побудовано новий симулятор мережевих атак. Симулятор спроектований так, щоб його було легко встановити з мінімальними залежностями та з можливістю швидко працювати на одному комп'ютері. Він також призначений для моделювання мережевого тестування на вищому рівні абстракції, для швидкого створення прототипів та тестування алгоритмів.

3.2 Структура симулятора мережевих атак

Мережева модель визначає організацію, підключення та конфігурацію машин на мережі і визначається кортежем {підмережі, топологія, машини, служби, брандмауери}. Наприклад мережі, що складається з п'яти підмереж, 11 машин і брандмауерів між кожною підмережею показана на рисунку 3.1. Ця мережа може запускати будь-яку кількість служб, так як це визначається на рівні машини. Ця модель має на меті абстрагувати деякі деталі реальної мережі, які не є такими необхідими при розробці автономних агентів, таких як конкретні типи зв'язків між машинами та розташування комутаторів та маршрутизаторів у мережі. Причина цієї абстракції полягає в тому, щоб намагатись тримати симулятор якомога простішим і на тому рівні абстракції, на якому очікується для агента.

Конкретні деталі виконання кожної дії, наприклад який порт для комунікації - це деталі, які можуть бути оброблені в залежності від програми-реалізації при переході до систем вищої точності. Тестування на проникнення вже рухається у цьому напрямку за допомогою таких фреймворків, як Metasploit,

які абстрагують як точно виконується експлойт і просто надається спосіб дізнатись, чи експлойт можно застосувати до сценарію та запустити його, подбавши про всі деталі експлойту нижчого рівня. Ця простіша модель мережі також використовується для того, щоб зберегти її якомога більш загальною та легко масштабованою.

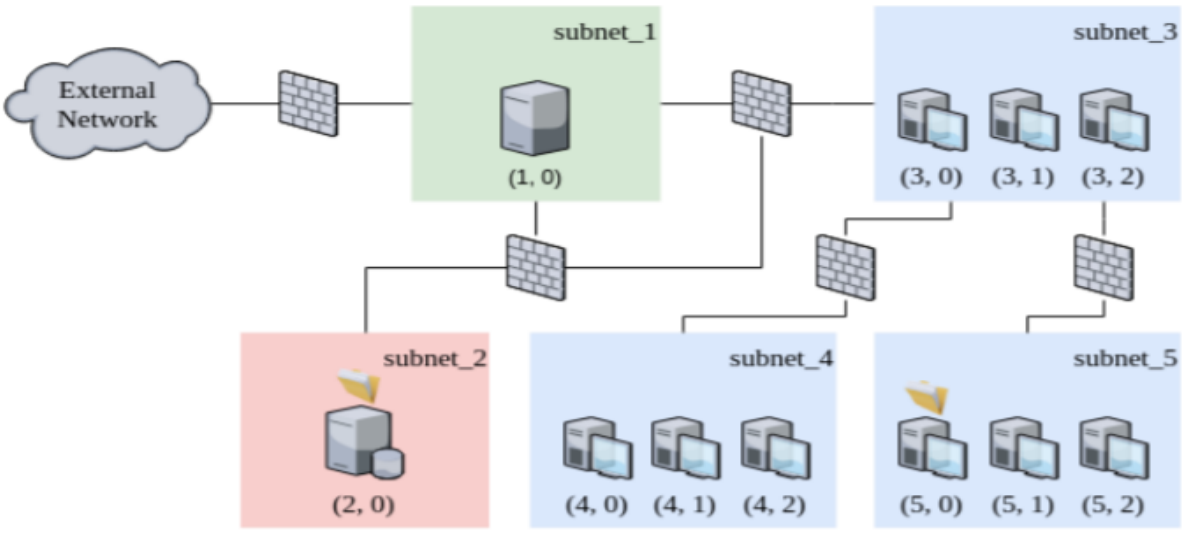


Рисунок 3.1 – Приклад мережі з пятьма підмережами

3.2.1 Підмережі

Кожна мережа складається з декількох підмереж. Підмережа - це менша мережа всередині більшої мережі, яка складається з групи однієї або декількох машин, які всі здатні повноцінно спілкуватися один з одним. Кожна підмережа має свою адресу підмережі, яка вказана як перше число в будь-якій машинній адресі (наприклад, 4 у адресі (4, 0)). Це спрощення IP-адрес, які використовують 32-розрядний рядок та окрему 32-розрядну маску підмережі для визначення мережі, підмережі та адреси машини. Для цілей симулятора мережі має сенс використовти простішу систему, оскільки ми маємо справу лише з однією мережею, на відміну від IP-адрес які мають справу з мільйонами машин у тисячах мереж через Інтернет. Хоча всі машини в межах підмережі

можуть повноцінно спілкуватися, зв'язок між машинами на різних підмережі обмежено. Зв'язок між підмережами контролюється топологією мережі та налаштуванням брандмауера.

3.2.2 Топологія

Топологія мережі визначає, як підключені різні підмережі, та керує як підмережі можуть взаємодіяти безпосередньо між собою та із зовнішньою мережею. Як приклад, в мережі на рис. 3.1 підмережа 1 є єдиною мережею, яка підключена до зовнішнього світу і всіх підмереж 1, 2 і 3 пов’язані між собою, тоді як це можливо лише для зв’язку з машинами в підмережах 4 і 5 через машину в підмережі 3. Таким чином, зловмисникові, можливо, доведеться переміщайтеся по машинах у різних підмережах, щоб мати змогу досягти цільових машин. Ми можемо розглядати топологію мережі як неорієнтований граф з підмережами як вершинами та з'єднання як ребра відповідно. Так, ми можемо представити його за допомогою матриці суміжності з рядками та стовпцями, що представляють різні підмережі, приклад матриці, як показано на рис.3.2 для прикладу мережі на рис. 3.1.

subnet	0	1	2	3	4	5
0	1	1	0	0	0	0
1	1	1	1	1	0	0
2	0	1	1	1	0	0
3	0	1	1	1	1	1
4	0	0	0	1	1	0
5	0	0	0	1	0	1

Рисунок 3.2 – Приклад топології мережі з пятьма підмережами

3.2.3 Машини

Найпримітивнішим будівельним блоком мережевої моделі є машина.

Машина в симуляторі мережі являє собою будь-який пристрій, який може бути підключений до мережі і, отже, мати з ним зв'язок і теоретичний контроль. Кожна машина визначається своєю адресою у вигляді кортежу (ідентифікатор підмережі, ідентифікатор машини), та конфігурацією. Приклад визначення машини можна побачити на рис 3.3. Значення машини визначається користувачем із вищими привілегіями, що надаються машинам, до яких зловмисник хоче отримати доступ або які власник хоче захистити. Кожна машина запускає сервіси, з якими можна зв’язатись з інших машин, що знаходяться в цій же підмережі або в сусідніх підмережах, якщо такий зв'язок дозволено брандмауером. Сервіси, доступні на кожному апараті визначаються його конфігурацією, і кожна машина в підмережі не обов'язково матиме однакові конфігурації. Ця опція додана, оскільки не кожна машина в мережі буде однаковою. У деяких випадках можна очікувати використання для різних цілей, наприклад Веб-сервери, сховище файлів, користувацькі машини. Служби, присутні на машині, також визначають її точки вразливості, оскільки вони є тими, що зловмисник прагне використати.

```
Machine: {
        address: (1, 2),
        value: 0,
        configuration: {
                ftp: true,
                ssh: true,
                http: true,
}
```

Рисунок 3.3 – Приклад конфігурації машини

3.2.4 Сервіси

Сервіси або служби використовуються для представлення будь-якого програмного забезпечення, що працює на машині, або спілкується з мережею.

Вони аналогічні програмному забезпеченню, яке слухає на відкритому порту комп’ютера або підключеного пристрою. Внутрішні служби симулятора мережі вважаються вразливими точками на будь-якій з машин, і можуть розглядатися як служба, яка має відомий експлойт, про який зловмисник знає. У реальному сценарії це було б те саме, що відстежувати лише послуги, на які зловмисник має відомий експлойт, ігноруючи будь-які інші незахищені служби. Виходячи з цього ми припускаємо, що кожна машина може бути скомпрометована хоча б однією дією, тому робота агентів полягає в тому, щоб знайти, яка служба працює на машині, і вибрати правильний експлойт проти неї. Кожен сервіс визначається унікальним ідентифікатором, а також вірогідністю вдалого використання експлойта. На рис. 3.3 показано приклад машини в мережевому сценарії де зловмисник може використати експлойти для служб ftp, ssh та http, тоді як на рис. 3.4 показано набір послуг, що використовуються, і пов'язана з цим ймовірність успіху та вартість їхніх дій. Ідентифікатор кожної послуги може мати будь-яку унікальну цінність і не обов’язково повинна бути назвою, пов’язаною з сервісу реального світу. Таким чином, симулятору мережі легко генерувати тестові сценарії з будь-якою кількістю машини та сервісів для допомоги у тестуванні ефективності масштабування агентів шляхом простого генерування ідентифікаторів сервісів за потреб. При досліджені програми на більш реальні налаштування, ідентифікатор буде замінено на конкретну назву служби та версію, щоб можна було відстежувати вразливості та знати, які служби вимагають виправлення (наприклад, версія Samba 3.5.0).

```
Exploitable_services: {
        ftp: {
                probability: 0.8,
                cost: 3
        },
        ssh: {
                probability: 0.5,
                cost = 2
        },
        http: {
                probability: 0.2,
                cost: 1
        }
}
```

Рисунок 3.4 – Приклад налаштувань сервісів та їх значень

3.2.5 Брандмауери

Кінцевим компонентом мережевої моделі є брандмауери, які існують уздовж з'єднань між будь-якими підмережами, а також між мережею та зовнішнім світом. Діють брандмауери як контролери, які контролюють яким службам можна спілкуватися на машинах у даній підмережі та з іншими точками підключення поза підмережею. Вони функціонують, щоб дозволити користуватися певними послугами і доступами до них із машин у підмережі з правильними дозволами, одночасно блокуючи доступ до цієї послуги з небажаних точок входу. Кожен брандмауер визначається набором правил, який визначає, який службовий трафік дозволений для кожного напрямку вздовж з'єднання між будь-якими двома підмережами або із зовнішньої мережі. На рисунку 3.5 показаний приклад брандмауера, який знаходиться між підмережами 1 і 3, що дозволяє отримати доступ до служби ssh на машинах підмережі 3 з машини на підмережі 1 та доступ до служб ftp та http на машинах у підмережі 1 з машини підмережі 3. У реальному світі правила налаштування брандмауера зазвичай встановлюються шляхом визначення того, який порт може бути доступний або налаштувань ос, однак для простоти і оскільки в більшості випадків одні й ті ж служби працюють на одних і тих же номерах портів, було вирішено замість цього визначити правила за послугою, а не за портом.

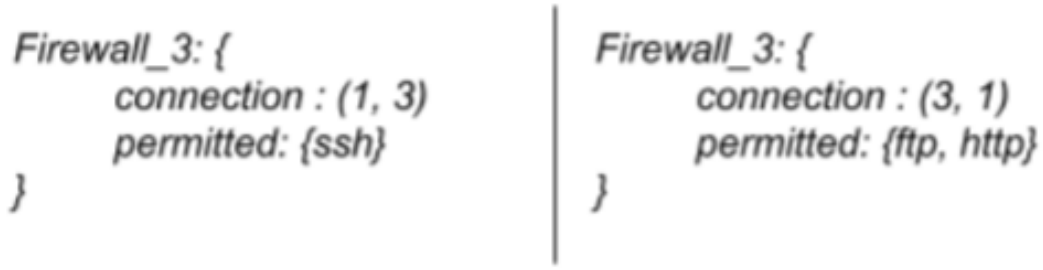


Рисунок 3.5 – Приклад брендмауерів у мережі.

3.3 Оточення агенту

Компонент оточення побудований на основі моделі мережі і діє як інтерфейс між зловмисником і мережею. Він відповідає за моделювання поточних знань нападників та положення під час атаки на мережу. Наприклад, він відстежує інформацію про те, які машини зловмисник успішно зламав, які машини вони можуть досягти та які знання вони мають про сервіси, наявні на кожній машині. Його головна функція полягає в тому, щоб контролювати запуск атаки з самого початку, де зловмисник ще не зробив взаємодію з мережею і не має інформації про жодну машину в мережі, та до кінця епізоду, коли нападник або здається, або успішно скомпрометує цільові машини в мережі. Ми моделюємо компонент середовища як MDP, оскількиця структура дуже універсальна і є будівельним блоком, який використовується багатьма алгоритмами ШІ.

Компонент середовища моделює проблему пентестування мережі як MDP і визначається кортежем $\{\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{T}}\}$. Стан визначається як поточні знання та положення зловмисника в мережі. Дії це доступні сканування та експлойти, які зловмисник може виконати для кожної машини в мережі. Функція винагороди — це просто вартість будь-яких експлуатованих машин мінус вартість виконаних дій. Функція переходу контролює результат будь-якої даної дії та враховує тип дії, підключення, брандмауери та ймовірнісний характер експлойтів.

Стан $s \in \boldsymbol{\mathcal{S}}$ визначається як сукупність усієї відомої інформації для кожної машини мережі. Тобто стан включає в себе кожну машину, якщо машина скомпрометована чи ні, доступна чи ні, і кожноу послугу, незалежно від того, чи є ця послуга, відсутня чи невідомий, існує вона чи ні. Машина вважається скомпрометованою, якщо експлойт було успішно використано проти неї. У той час як машина вважається доступною, якщо вона знаходиться в підмережі, яка є загальнодоступною (підключений безпосередньо до зовнішньої мережі) у тій самій підмережі, що й скомпрометована машина або в підмережі,

безпосередньо підключеній до підмережі, яка містить зламану машину. Таким чином, простір станів — це всі можливі комбінації скомпрометованих, доступних і сервісних знанньь для кожного сервісу та для кожної машини. Отже, простір станів зростає в геометричній прогресії з кількістю машин і сервісів у мережі. $|\boldsymbol{S}| \in O(3^{|E||M|})$ показує розмір простору станів, $|\boldsymbol{S}|$, де |E| - кількість послуг, які можна використовувати, і |M| єкількість машин в мережі. Основа для експоненції дорівнює 3, оскільки для кожного вразливого сервісу, знання агентів може мати одне з трьох значень: наявне, відсутнє або невідоме.

Розглянемо приклад мережі та ассоційованого з цією мережею станом.

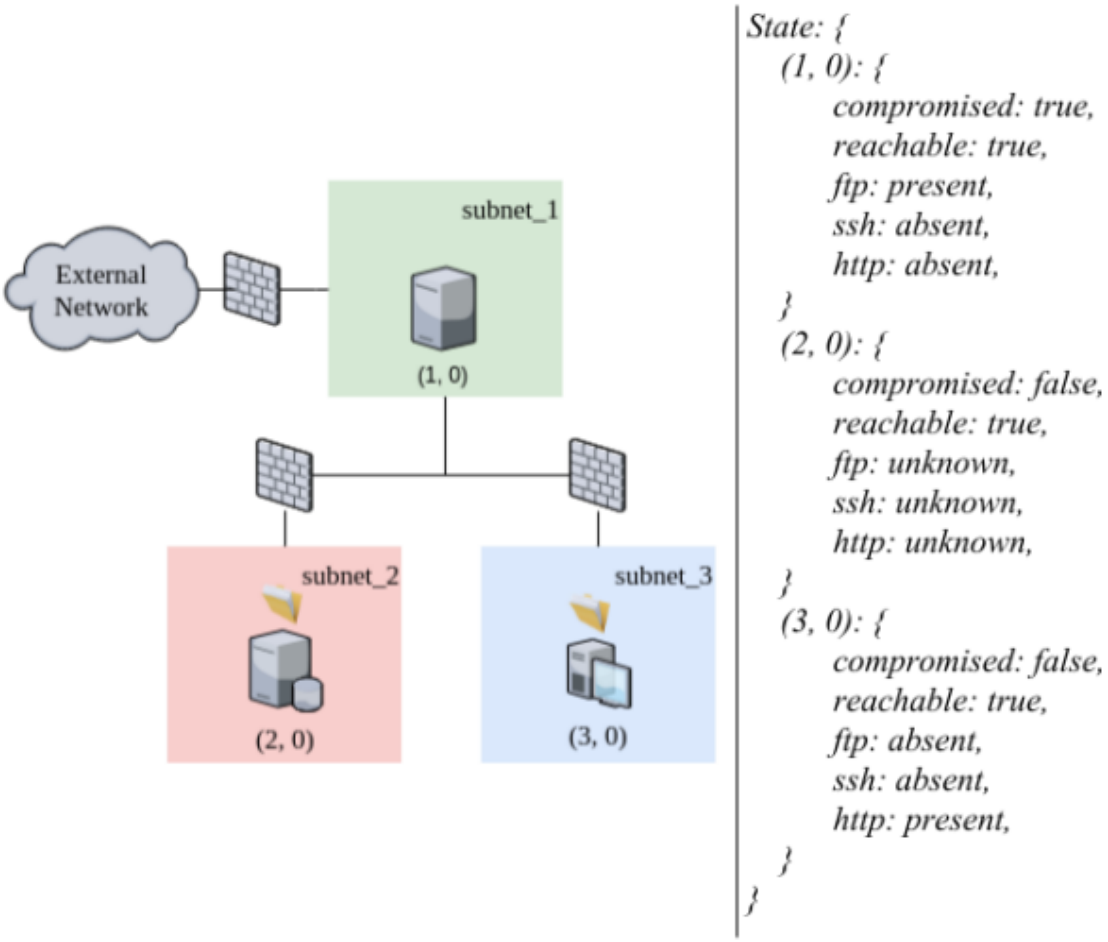


Рисунок 3.6 – Приклад мережі та стану.

У цьому стані зловмисник успішно зламав комп’ютер за адресою (1, 0) і може зараз взаємодіяти з машинами в підмережах 2 і 3, що вказує на те, що для «досяжності» встановлено значення «true».для кожної машини в цих

підмережах. Додатково конфігурація машини на (3, 0) є відомою, що було б отримано за допомогою дій сканування, тоді як конфігурація для машини (2, 0) невідома. Зауважте, що стан не містить жодної інформації про налаштування брандмауера, оскільки для визначення цього потрібен привілейований доступ. Для цього симулятора ми припустимо, що зловмисник не може отримати цю інформацію, і замість цього він повинен її дізнатися опосередковано через успіх і невдачу своїх дій.

Простір дій, $\mathcal{A}$, — це набір доступних дій у симуляторі і включає окремі сканування та експлойти для кожної служби та кожної машини в мережі. Дію сканування розроблено для імітації повного сканування Nmap, яке повертає інформацію про те, які сервіси працюють на кожному порті даної машини, а також версії кожного сервісу. Насправді більш цільове сканування може знадобитися для виявлення повної інформації про конкретні сервіси, але у багатьох випадках сканування Nmap повертає інформацію, необхідну для визначення того, який сервіс працює. Дії сканування вважаються детермінованими, завжди повертаючи інформацію про наявність або відсутність сервісу.

Для кожної можливої служби в мережі існує відповідна дія експлойту. Кожна дія злому може бути детермінованою або недетермінованою залежно від конфігурації середовища, вибраної користувачем. Успішна дія злому призведе до того, що цільова машина стає скомпрометованою. Успіх будь-якого експлойту визначається тим, чи є цільова машина доступна, цільова служба присутня, чи ця служба заблокована брандмауером чи ні, а також ймовірністю успіху дії.

Кожна дія також має відповідну вартість. Цю вартість можна використовувати для представлення будь-яких показників, таких як час, навички, грошові витрати або шум, створений від певної дії, залежно від того, який показник ефективності намагається отримати оптимізація.

```
Action_1: {
   target: (1, 0),
   type: scan,
   cost: 1
}
```

```
Action_2: {
   target: (1, 0)
   type: exploit
   service: ssh
   cost: 3
   probability: 0.8
}
```

Рисунок 3.7 – Приклад сканування та дії злому.

Показано приклади визначення дій сканування та злому. Обидві дії націлені на одну і ту саму машину за адресою (1, 0). «Action_1» — це сканування з вартістю 1, а «Action_2» — це експлойт для служби ssh і має вартість 3 і ймовірність успіху 0,8.

Функція винагороди використовується для визначення цілей автономного агента і того, що намагається оптимузовати агент. Винагорода визначається за перехід $\boldsymbol{\mathcal{R}}$ (s, a, s'), тож починаючи з одного стану s агент виконує дію a і переходить в результуючий стан s'. Нагорода за будь-який перехід дорівнює вартості будь-якої знову скомпрометованої машини в наступному стані s' мінус вартість дії a. Тож якщо жодна машина не була зламана, то нагородою буде просто вартість виконаної дії. За допомогою цієї функції винагороди метою зловмисника стає спроба скомпрометувати всі машини з додатним значенням нагороди у мережі при мінімізації кількості або вартості дій що були використані. Це імітує дії зловмисника в реальному світі, чиєю метою, як ми припускаємо, є отримання привілейованої інформації або отримати привілейований доступ до системи.

Функція переходу, $\boldsymbol{\mathcal{T}}$, визначає, як середовище розвивається з часом після виконанх дій. Для симулятора наступний стан залежить від того, була дія успішною чи ні, що в свою чергу залежить від ряду факторів. Зокрема, чи є ціль дії досяжною, незалежно від того, чи є ця дія скануванням чи експлойтом. На рис. 3.8 показано приклад переходу, де зловмисник успішно використовує службу http на машині за адресою (3, 0). Новий стан представляє цей успіх через машину (3, 0), яка має для скомпрометованої змінної значення true.

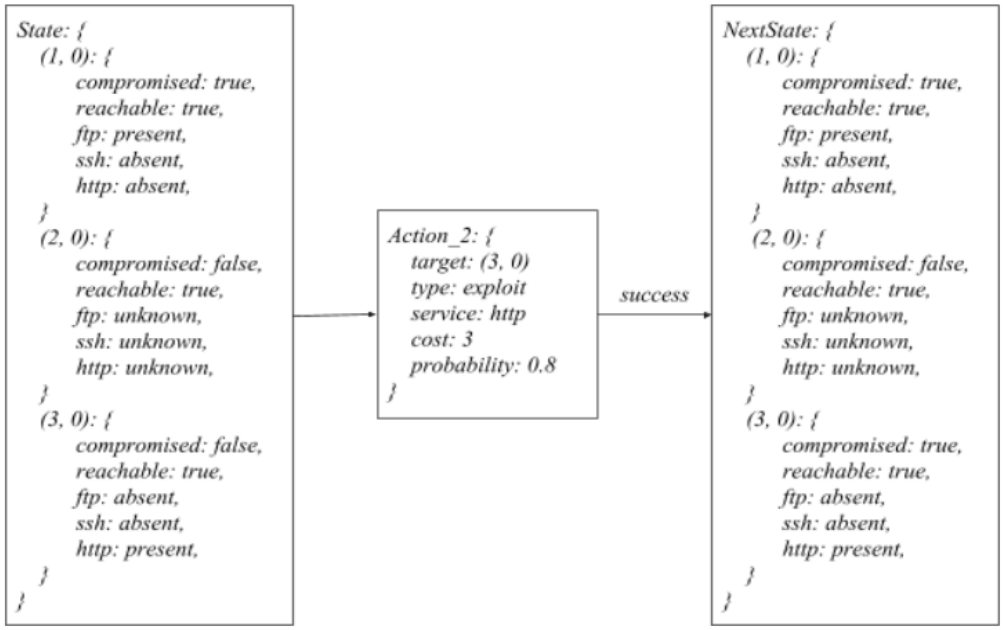


Рисунок 3.8 – Приклад переходу стану після успішно виконаної дії злому до http сервісу.

3.4 Реалізація

Цілями симулятора є швидкість, простота в установці та можливість використання для швидкого створення прототипів агентів ШІ. Щоб допомогти досягти цих цілей, програма була написана повністю з використанням мови програмування Python і популярниими, добре підтримуваними відкритими бібліотеками. Зокрема, використовувалися бібліотеки numPy для швидких обчислень на основі масивів і Matplotlib і NetworkX для візуалізацій. Можна було б побудувати більш швидкий симулятор, який використовує мову рівня нижчого за Python, таку як C++, однак ще одна причина використовувати Python була це популярність для досліджень машинного навчання та його добре підтримуваних бібліотек глибокого навчання, таких як як Tensorflow і Keras, які зазвичай використовуються для розробки агентів.

Схема архітектури симулятора показана на рис. 3.9. Існує ряд різних модулей, які обробляють мережеву модель, MDP та інші функції зі спільним

головним компонентом, який є модулем середовища. Модуль середовища є основною точкою взаємодії для агента і має чотири основні функції: завантаження, очищення, крок і рендер.

Функція завантаження завантажує новий сценарій середовища шляхом створення мережі за стандартними зразками або завантаження мережі з конфігураційного файлу.

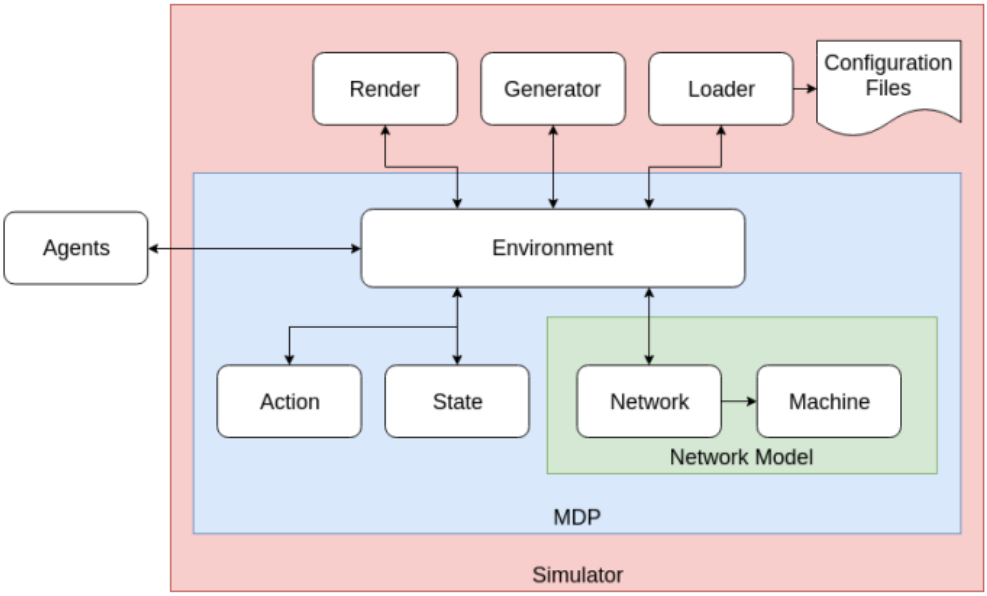


Рисунок 3.9 – Архітектура програми симулятора мережі

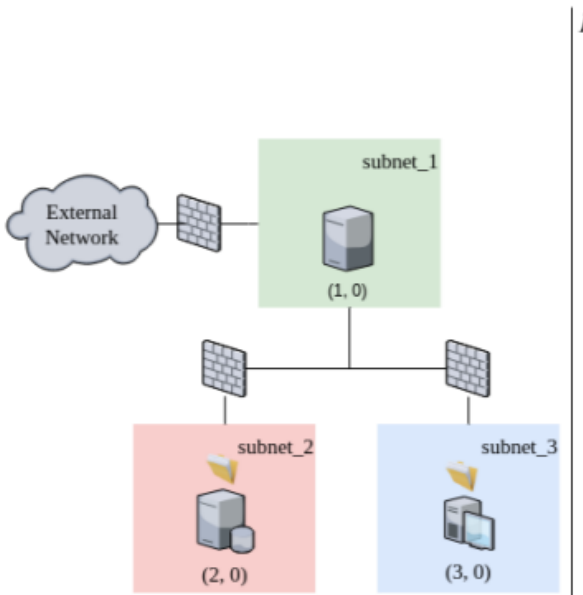
Функція кроку є основною точкою взаємодії агента з середовищем. Береться дія і виконується перехід з урахуванням поточного стану симулятора і повертається наступний стан, винагорода за виконання дії та чи досягнута загальна мета. Типовий цикл навчання або тестування автономного агента включає:

– скидання симулятора у початковий стан;
– використання стану для вибору дії;
– виконання дії проти середовища, використовуючи функцію кроку для отримання наступного стану та винагороди
– повтор кроків вибору дії та її виконання, доки мета не буде досягнута або спливає ліміт часу(кроків).

Цей повний цикл становить один епізод і агент може потім скинути

середовище і повторити процесс протягом необхідної кількості епізодів.

Функція очищення встановлює середовище в початковий стан і повертає початковий стан агента і є синонімом початку нової атаки на чисту мережу. Початковий стан, це той стан, де агент не зламав жодної машини в мережі, і лише підключені до мережі зовнішньої мережі девайси доступні. Також немає інформації про сервіси, які працюють на жодній машині в мережі.




```
Initial_State: {
  (1, 0): {
      compromised: false,
      reachable: true,
      ftp: unknown,
      ssh: unknown,
      http: unknown,
  }
  (2, 0): {
      compromised: false,
      reachable: false,
      ftp: unknown,
      ssh: unknown,
      http: unknown,
  }
  (3, 0): {
      compromised: false,
      reachable: false,
      ftp: unknown,
      ssh: unknown,
      http: unknown,
  }
}
```

Рисунок 3.10 – Приклад початкового стану мережі

Функція візуалізації надає кілька способів візуалізації середовища та мережі, а також епізод атаки в мережі. Перший варіант - просто відобразити мережу як граф, де кожна вершина - це машина в мережі, а машини в одній підмережі згруповані разом, а ребра - це зв’язки між кожною машиною в підмережі та між підмережами. Ця опція дозволяє користувачеві візуалізувати топологію мережі. Другий варіант візуалізації дозволяє користувачеві побачити, як змінюється стан мережі впродовж епізоду нападу. Цей режим показує стан мережі разом із дією та винагороду, отримані за кожен крок, і можуть бути використані для візуалізації політики атаки агента та ідентифікації зламаних служб на шляху атаки.

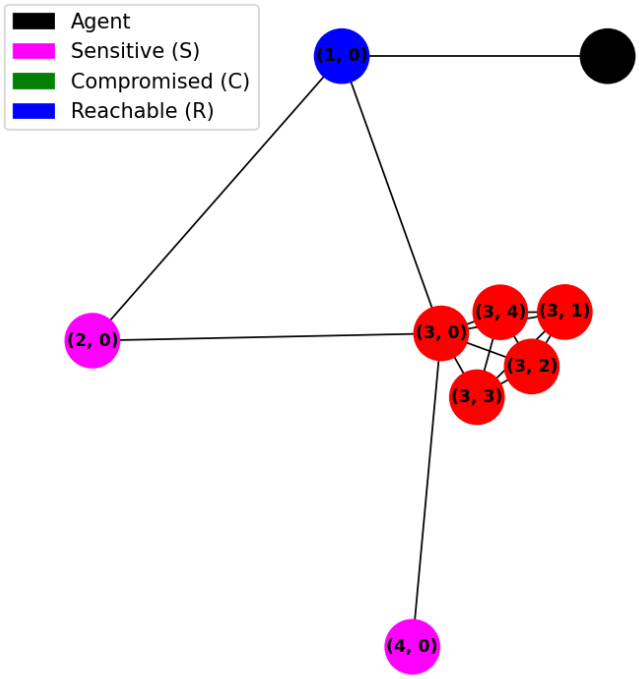


Рисунок 3.11 – Приклад візуалізації мережі

Кожна вершина представляє собою машину в мережі, яка згрупована за підмережею. Кожне ребро між вузлами з однаковими ідентифікаторами підмережі представляє підключення всередині підмережі, а ребра між вузлами з різними ідентифікаторами підмережі представляють зв'язок між підмережами. Рожеві вузли — це цільові машини, які містять «конфіденційні документи», червоні і рожеві вузли представляють машини, які недоступні агенту, сині вузли представляють машини, доступні для агента, і чорний вузол представляє позицію агента.

Симулятор здатний запускати будь-яку довільну мережеву структуру, визначену користувачем в файлі конфігурації. Спеціальна мережа визначається кортежем: {підмережі, топологія, чутливі машини, служби, службові експлойти, конфігурації машин, брандмауери}.

Щоб забезпечити швидке тестування агентів на мережах різного розміру та включено стандартну топології мережі, яка може бути використанаа як еталон для дослідників для порівняння продуктивності агентів, було включено можливість автоматичного створення мережі для заданої кількості машини, M і

сервісів, E.

Мережа розділена на три основні підмережі (рис 3.12):

– демілітаризована зона (DMZ),
– чутлива підмережа
– підмережі користувачів

M Машин поділяються на кожну підмережу, як показано нижче, по одній машині в кожній із DMZ(демілітаризована зона) та чутливих підмереж, а решта M - 2 машини в підмережах користувачів, які з'єднані в структуру бінарного дерева з максимум п’ятьма машинами на підмережу та з’єднані лише між батьківським і дочірнім вузлами дерева. Підмережі DMZ, чутливих і рут усі з’єднані, а DMZ підключене до зовнішньої мережі.

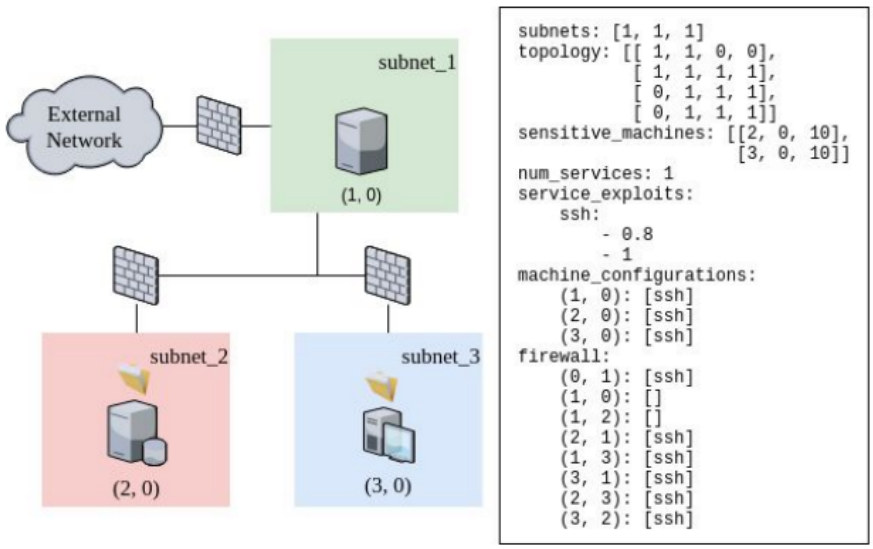


Рисунок 3.12 – Приклад конфігураційного файлу та топології згенерованої мережі.

Імовірність успіху кожного експлойту можна вибрати, однак за замовчуванням береться значення ймовірностей із розподілу на основі розподілу оцінок складності 10 найпоширеніших експлойтів, які використовувалися у останніх роках. Оцінка складності атаки – це показник, створений за допомогою загальної системи оцінки вразливостей (CVSS) і використовується для відображення, наскільки важко знайти машину, яку можна скомпрометуати даною атакою, а також ймовірність успіху та вміння, необхідне для

використання даної атаки. Зокрема, ймовірності були обрані на основі оцінок складності атаки для CVSSv2, яка оцінює експлойти як «низької», «середьої» або «високої» складності атаки. Буі використаний той самий підхід, коли встановлені ймовірності успіху становлять 0,8, 0,5 і 0,2 для «низької», «середньої» або «високої» складності атаки відповідно. Використовуется цей підхід аби наблизитися до реальних даних, знайдених в реальному світі.

3.5 Продуктивність мережі

Для перевірки продуктивності та властивостей масштабування були використані показники дій в секунду та час завантаження, виміри яких були проведені залежно від кількості машин і служб у мережі. Всі експерименти були проведені за допомогою одного ядра на персональному ноутбуці, під керуванням операційної системи windows 10. На тестовій машині працював процесор Intel Core i5 8-го покоління на частоті 2,7 ГГц і мав 16 ГБ оперативної пам'яті.

Час завантаження симулятора вимірювався для діапазону машин від 10 до 1000 і діапазону сервісів від 10 до 1000. На рис 3.13 показано середній час завантаження для симулятора порівняно з кількістю машин і послуг. Для середнього часу завантаження мінімальний час становив 0,007 ± 0,0007сек (середнє ± стандартне відхилення), знайдено в найменшому розмірі симулятора, перевіреному на 10 машинах і 10 сервісах і максимальний час становив 3,657 ± 0,09 с і був для найбільшого розміру симулятора, 1000 машин і 1000 сервісів.

Для порівняння також було виміряно час завантаження окремої віртуальної машини. Було використане програмне забезпечення віртуалізації Oracle VM VirtualBox, що запускав віртуальні машини на базі Linux Metasploitable 2, як тест ВМ. Середній час завантаження для однієї машини без графічного інтерфейсу (режим без голови) становив 0,249 ± 0,023 с, усереднена за 10 запусків. Цей час завантаження більше ніж у 300 разів за завантаження 10

машин і служб у симуляторі.

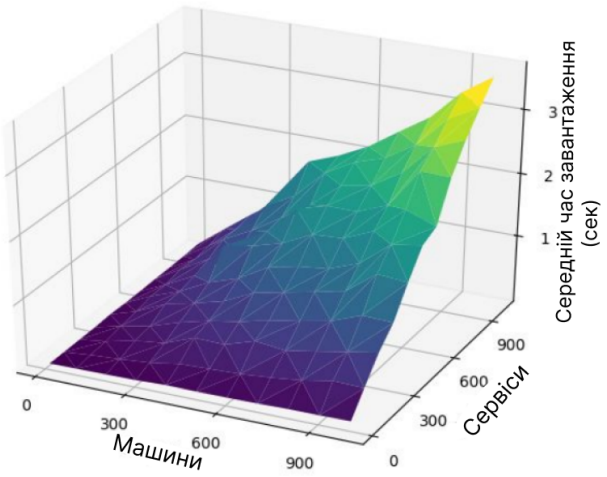


Рисунок 3.13 – Час завантаження симулятора мережевих атак залежно від кількості машин і служб.

Щоб перевірити властивості масштабування симулятора під час регулярного використання, було виміряно число дій за секунду в залежності від розміру симулятора. Проведені експерименти на симуляторах, використовуючи ряд машини (від 10 до 480) і служб (від 10 до 480) і вимірювались дії за секунду та усереднювось це протягом кількох запусків. Для середнього значення дій за секунду перевірених налаштувань мінімум становив 16329 ± 1907 дій в секунду для тренажера з 480 машинами і 10 сервісами, при цьому максимальна була 126383 ± 3843 дії в секунду для симулятора з 10 машинами та 30 сервісами.

Результати усереднені за 10 запусків з 10 000 дій на кожен запуск, використовуючи тільки детерміновані дії. Виконані дії вибирались з простору дій послідовно, для спроби збереження постійності у тесстах. На лівому графіку наведена продуктивність у залежності від кількості машин для мережі працює 10, 240 і 480 сервісів. На правому графіку наведена продуктивність у залежності від кількості сервісів для мережі з 10, 240 і 480 машинами. На обох графіках червоний рядок «Усі» усереднений за всіма значеннями сервісів (ліворуч) або машини (праворуч), що були протестовані.

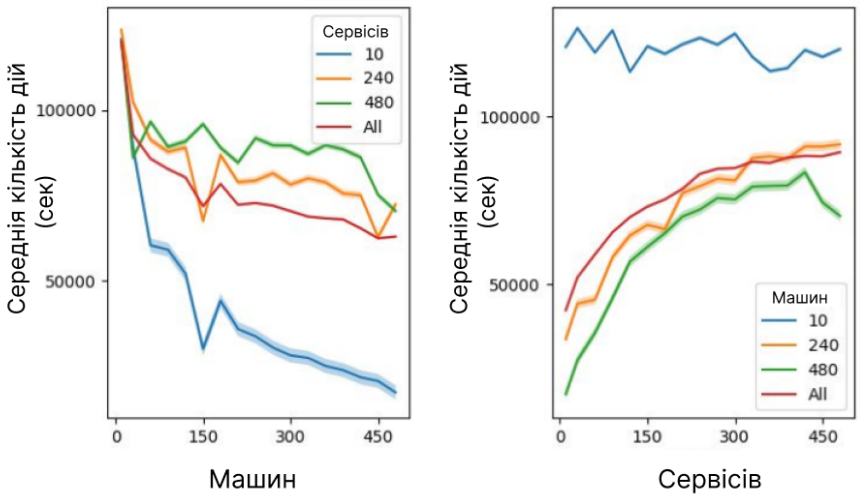


Рисунок 3.14 – Масштабування продуктивності симулятора

Для порівняння був виміряний середній час, необхідний для виконання сканування та експлойту між двома віртуальними машинами. Було використано одну віртуальну машину зловмисника під керуванням Kali Linux (https://www.kali.org/) і одну вразливу віртуальну машину під керуванням ОС Metasploitable. Також використано стандартне сканування Nmap одного порту, зокрема порту 21, на якому працювала служба ftp, тоді як експлойт використовувався для бекдору ftp-сервера версії 2.3.4 VSFTPD (https://www.exploit-db.com/exploits/17491/), що був запущений за допомогою платформи Metasploit. Значення часу, необхідного для сканування Nmap одного порту, становив 0,253 ± 0,03 с, усереднений за 10 виконань. Це є еквівалентно приблизно 4 діям за секунду. Час на виконання експлойту становив від 3 до 5 секунд або приблизно 0,2-0,3 дії в секунду.

Тож, було виміряно продуктивність симулятора за двома показниками: час завантаження та час, необхідний для виконання дії (тобто дій за секунду). З точки зору часу завантаження, середній час лінійно збільшується з кількістю машин і числа сервісів, присутніх в мережі. З точки зору практичного використання, в гіршому випадку, коли було 1000 машин і 1000 сервісів час завантаження становив приблизно 3,5 секунди, що на багато порядків менше

ніж час, який агент витратив би на навчання в навколишньому середовищі. Крім того, у порівнянні з часом завантаження окремої віртуальної машини, що займало приблизно 0,25 секунд для запуску віртуальної машини і додаткового часу для завантаження ОС, швидкість симулятора є значно вищою. Найгірший варіант виконання дій тестованого симулятора становив близько 17 000 дій в секунду для мережі з 480 машинами та 10 службами, що на порядок швидше, ніж використання мережі ВМ, яка може виконати приблизно 4 дії сканування Nmap і 0,3 дії експлойту в секунду для простої мережі з двох віртуальних машин.

Для алгоритмів ШІ, які покладаються на взаємодію з середовищем для вивчення шляху атаки, такого як RL, швидкість, з якою відбуваються ці взаємодії має значний вплив на те, наскільки швидко вони здатні створити оптимальну політику атаки.

Загалом, представлений симулятор є простим у розгортанні, швидким і гнучким тестовим стендом для тестування на проникнення автоматизованими агентами. Головною слабкістю цього підходу – є кореляція з реальним середовищем, яке містить набагато більшу складність з точки зору експлойтів, сервісів, конфігурації машини та динаміки мережі, однак симулятор пропонує абстрактне середовище, яке має велику кількість застосувань в сфері ші.

4 ТЕСТУВАННЯ НА ПРОНИКНЕННЯ ЗА ДОПОМОГОЮ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

4.1 Навчання з підкріпленням

Навчання з підкріпленням - це галузь дослідження, а також клас методів рішення для навчання переходів зі станів на дії з метою максимізації винагороди. У навчанні з підкріпленням є середовище та агент, який взаємодіє з навколишнім середовищем з метою вивченням оптимальних дій, які слід вжити при кожному стані обєкта. Існує чотири основні компоненти системи навчання з підкріпленням окрім середовища та агенту, це політика π, функція винагороди, $\boldsymbol{\mathcal{R}}$ (s ’, a, s), функція ваги(значення), V (s), і не обов’язкова модель навколишнього середовища, модель переходів $\boldsymbol{\mathcal{T}}$. Політика - це залежність від простору станів, $\boldsymbol{\mathcal{S}}$, до простору дій, $\boldsymbol{\mathcal{A}}$. Ми хочемо, щоб агент знайшов оптимальну політику π * що вибирає дію, $\boldsymbol{\mathcal{A}}$ із будь-якого стану, що максимізує загальну очікувану винагороду зі знижкою вартісті дій. Винагорода визначає негайну винагороду за поточний стан і надається оточенням у кожен момент часу. Функція value визначає значення стану в довгостроковій перспективі, значення стану, s, (тобто V (s)) - це загальна накопичена винагорода, яку агент може розраховувати отримати довгостроково, починаючи з цього стану. Модель середовища - це те, що говорить агенту щось про те, як поведеться середовище, і дозволяє агенту робити висновки. Модель переходу, $\boldsymbol{\mathcal{T}}$ (s ’, s, a) є прикладом моделі, оскільки вона дає агенту інформацію про очікуваний майбутній стан з урахуванням поточного стану та обраної дії. Навчання з підкріпленням ділится на стан коли модель присутня, вона відома як навчання з підкріпленням на основі моделі, тоді як якщо жодної моделі немає, то вона є без модельним навчанням з підкріпленням. Проблеми, засновані на моделях, як правило, вирішуються за допомогою планування, а вирішення проблем без використання моделей повинні спиратися на спроби і помилки, щоб знайти оптимальну політику щодо навколишнього середовища.

Однією з головних переваг використання підходу RL є те, що він дозволяє нам підійти до проблеми без передбачуваних попередніх знань або моделі щодо ймовірності результату дії для будь-якого заданого стану і замість цього дозволяє агенту дізнатися їх. Це забезпечує рішення однієї з проблем автоматизованого тестування на проникнення, а саме підтримку актуальності та точності моделі у результатах роботи. Природа програмного забезпечення та вразливостей швидко розвивается тож для того, щоб створити точну модель, необхідно було б протестувати будь-яку експлойт на широкому діапазоні систем і повторювати цей процес з часом. Використання RL з іншого боку вимагає лише визначення стану, набору дій, які може здійснити агент, і функцію винагороди. Потім агент явно вивчає модель середовища через взаємодію. Це означає, що в міру розвитку простору кібербезпеки було б необхідним лише оновити дії, які може виконати агент, і залишити моделювання агенту.

Протягом багатьох років було розроблено багато різних алгоритмів навчання з підкріпленням. Для цього дослідженнями будемо використовувати Q-алгоритм, який є алгоритмом RL для вивчення оптимальної політики в без модельних задачах. Він покладається на використання досвіду для вивчення функції Q-значення, Q (s, a), що повідомляє агенту очікувану винагороду, якщо вони виконують дію a із стану s. Було доведено, що за умови достатнього часу та дослідження навколишнього середовища цей алгоритм буде зближуватися щодо оптимальних значень Q для кожного стану та пари дій. Як тільки функція Q-значення збігається, воно потім може бути використано для визначення оптимальної дії для стану, просто вибравши дію за допомогою найвищого значення Q і, отже, використовують його для пошуку оптимальної політики для даного середовища. RL - це потужний та універсальний підхід для вирішення марківських процесів прийняття рішень, однак незважаючи на переваги існують складності в реалізації підходу та його продуктивність може бути не стабільною. Ця складність може бути однією з ключових причин, через що навчання з підкріпленням ще не використовується в автоматизованому режимі

тестування на проникнення.

4.2 Алгоритм навчання з підкріпленням

Алгоритми RL вивчають оптимальні дії завдяки взаємодії з оточенням. Все починается з якоїсь початкової, як правило, випадкової політики дій, потім ітеративно вивчаючи цінність певних дій для даного стану, Q (s, a), вибравши дію на основі поточної політики, застосовуючи цю дію до навколишнього середовища, а потім оновлюючи значення дії стану, Q (s, a), на основі отриманого досвіду (рис. 4.1). Конкретні алгоритми RL відрізняються залежно від того, як вони вибирають дії, оновлюють їх оціночні значення для функції вартості та форми функції вартості.

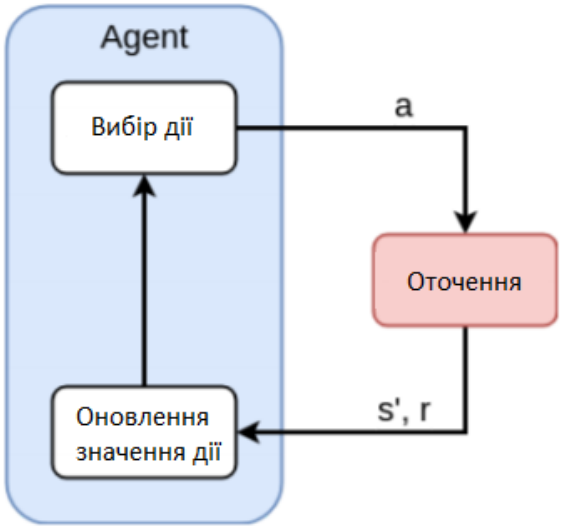


Рисунок 4.1 – Цикл навчання з підкріпленням.

Існує низка різних стратегій вибору дій, але дві з них є загальновживаними та ті, що використовуються для цього дослідження, є ε-жадібна та стратегія верхніміх меж довіри(ВМД). Обидві ці стратегії спрямовані на збалансування навчання(обираются не найкращі за оцінкою дії) та використання(обираются найкращі за оцінкою дії). Стратегія вибору ε-жадібної дії робить це, вибираючи випадкову дію(навчання) з ε-ймовірностю і

обираючи найкращу поточну дію(використання) з ймовірністю 1-ε.

$$a_t = \begin{cases} \underset{a \in A}{argmax}\, Q(a) & with\ p(1-\varepsilon) \\ random\ a \in A & with\ p(\varepsilon) \end{cases}$$

Рисунок 4.2 Формула ε-жадібної стратегії вибору дій.

Це змушує агента навмання досліджувати дії, які, на його думку, не є цінними з огляду на поточний досвід. Загально прийнятим є впровадження ε-жадібного підходу поряд з ε-спадом, що зменшує значення ε з часом щоб по мірі покращення оцінок дій агентом, він обирав випадкові дії рідше. З іншого боку, підбір дій ВМД використовує додатковий вираз для дослідження, виконуючи вибір дій.

$$a_t = \underset{a \in A}{argmax} \left[Q(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Рисунок 4.3 Формула стратегії вибору дій верхніх меж довіри

Цей додатковий вираз збільшує цінність дій, які були прийняті рідше і діє для вимірювання визначеності оцінки вартості дії. Для цього дослідження ми реалізуємо алгоритми, які використовують обидва ці методи вибору дій, щоб мати змогу порівняти та дослідити, як вибір дії може вплинути на використання RL в автоматизованому режимі тестування.

Ми використовуємо Q-навчання у наших реалізаціях RL для стратегії оновлення вартості. Q-навчання - це поза політичний алгоритм для вивчення значень стан-дія та визначається рекурсивною функцією оновлення у рівнянні (рис 4.4) . Де α - розмір кроку, який контролює на скільки рухати поточну оцінку до нової оцінки і γ - коефіцієнт дисконтування, який контролює, скільки

важить безпосередня винагорода порівняно з майбутніми винагородами, котрі необхідно брати у розрахунок для уникнення ситуацій, не вибору дії, коли дія зараз дає погану винагороду, а в майбутньому найкращу. Доведено, що Q-навчання сходиться до оптимальних значень стану-дії, коли кількість відвідувань пари стану-дії наближається до ∞.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Рисунок 4.4 Формула Q-навчання

Останній ключовий аспект алгоритмів RL, який відрізняється між реалізаціями, - це яку форму приймає значення Q (s, a). Є два основних варіанти це табличне та функціональне наближення. Табличні методи використовують таблицю на зразок структури даних для зберігання значення дії-стану для кожної пари дії-стану, які оновлюються по мірі отримання агентом досвіду. Методи апроксимації функцій, навпаки, використовують функцію для генерації значення дії-стану та оновлення параметрів функції для покращення цих оцінок по мірі збільшення агентського досвіду. У цьому дослідженні буде застосовано як табличну, так і функціональну апроксимацію, буде досліджено наскільки ефективні обидва ці методи при застосуванні до мережевого тестування.

Табличні методи мають перевагу в тому, що їх легко реалізувати і часто можна знайти точні оптимальні рішення. Їх використання, однак, обмежується проблемами з відносно невеликими розмірами станів, оскільки кожна пара стан-дія повинна зберігатися. Вони також розглядають кожен стан окремо і тому не може використовувати знання, отримані про різні, але схожі стани, для узагальнення невідомих станів.

З іншого боку, методи апроксимації функцій можна використовувати наскільки завгодно великими просторами станів і здатні генерувати оцінки до невідомих станів. Однак вони більш складні у реалізації, та вимагають використання додаткової функції для апроксимації функції значення, та на

продуктивність сильно впливає використовуване представлення функції. Є багато варіантів як представити функцію, однак метод, який останнім часом привертає найбільшу увагу, і для якого були найбільші покращення RL, які були помічені за останні роки, було використання глибокої нейронної мережі. Поєднання використання Q-навчання та нейронних мереж для функції апроксимації - відома як глибоке Q-навчання і використовується для отримання найсучасніших результатів у ряді середовищ, включаючи гру Go та багатьох різних відеоігор.

4.3 Тестування на проникнення на основі навчання з підкріпленням

Для того, щоб використовувати безмодельний варіант навчання з підкріпленням для автоматизованого тестування, нам потрібно визначити проблему, як Марківські процесси приняття рішень у невідомому оточенні. Як обговорювалося раніше, MDP визначається кортежем $\{\boldsymbol{S},\boldsymbol{A}, \boldsymbol{R}, \boldsymbol{T}\}$. Ми представляємо стани, дії та винагороду, як представлено у Розділі 3, причому стани це статус та знання конфігурації агента для кожної машини в мережі, дії це доступні скани та атаки для кожної машини та винагорода, надана вартістю нещодавно скомпрометованих машин мінус вартість дії. Модель переходів, $\boldsymbol{T}$, звичайно невідома.

Таблиця 4.1 Визначення Марківських процессів прийняття рішень, де |M| це кількість машин у мережі, |E| це кількість вразливих сервісів

Компонент	Визначення
S	|M| x (скомпрометовані) x (досяжні) x |E| x (знання сервісів машини) Де (скомпрометовані) ∈ (true, false) (досяжні) ∈ (true, false) (знання сервісів машини) ∈ (немає, є, невідомо)
A	|M| x (скан, атака) x |E|
R (s', a, s)	Значення (s', s) – вартість(a)

T (s’, a, s)	Невідомо

Оскільки це основне дослідження використання RL для пентестування, було використано одне ключове припущення: агент має повне знання топології мережі. Це означає, що агентзнає адресу кожної машини в мережі та її досяжність. Це припущення зроблено в більшості підходів до планування атак і базується на доступності даних топології від клієнта, який запитує проводження тестування на проникнення. Це припущення можна було б послабити, але це означало б, що вектор представлення стану зміниться з часом, оскільки ми не знали би заздалегідь, скільки машин є в мережі і таким чином розмір станів буде зростати, оскільки буде відкрито все більше машин. Застосування RL до проблем, де представлення станів динамічне - набагато складніше, і переважна більшість роботи з RL включає в себе стаціонарні представлення станів. З цієї причини вважається, що краще мати це припущення на ранньому етапі дослідження, а потім, якщо RL виявиться багатообіцяючим, можливо, спробувати його застосувати без знань топології мережі і точно імітувати точку зору реального зловмисника.

Ми використовуємо три різні алгоритми Q-навчання: табличне Q-навчання з використанням ε-жадібного вибору дії (табличне ε-жадібне), табличне Q-навчання з використанням вибору дії ВМД (табличне ВМД) і глибоке Q-навчання з використанням одношарової нейронної мережі та ε-жадібний метод вибору дій (DQL). Ці алгоритми були обрані через те, що вони забезпечують табличну та функціональну апроксимацію RL реалізації, використовуючи табличні ВМД також надає можливість дослідити, як стратегія вибору дії впливає на продуктивність пентестування мережі.

Лістинг 4.1 – Алгоритм Q-навчання з ε-жадібною політикою вибору дії

```
Ініціалізуємо Q(s,a), для усіх s∈S, a∈A(s)
for епізод = 1,V:
      s_1=початковий стан
      for крок=1,T:
            З вірогідністю ε обираємо випадкову дію a_t
            з вірогідністю 1-ε обираємо a_t = argmax_a Q(s_t, a)
```

```
            Виконуємо дію a_t та отримуємо винагороду r_t та стан s_{t+1}
            Вираховуємо оцінку парі стан-дія
            Q(s_t, a_t) = Q(s_t, a_t) + α[r_t + γmax_a Q(s_{t+1}, a) − Q(s_t, a_t)]
            if s_{t+1} кінцевий стан:
                  закінчуємо епізод
            end if
            s_t = s_{t+1}
      end for
end for
```

Лістинг 4.2 – Алгоритм Q-навчання з ВМД політикою вибору дії

```
Ініціалізуємо Q(s,a), для усіх s∈S, a∈A(s)
for епізод = 1,V:
      s_1=початковий стан
      for крок=1,T:
            a_t = argmax_a[Q(s_t, a) + c√(log(n)/N(s_t a))], де log(n) це логарифм загальної кількості
            випробуваних нами дій n
            Виконуємо дію a_t та отримуємо винагороду r_t та стан s_{t+1}
            Q(s_t, a_t) = Q(s_t, a_t) + α[r_t + γmax_a Q(s_{t+1}, a) − Q(s_t, a_t)]
            N(s_t, a_t) = N(s_t, a_t) + 1
            if s_{t+1} кінцевий стан:
                  закінчуємо епізод
            end if
            s_t = s_{t+1}
      end for
end for
```

Було реалізовано DQL, із використанням відтворення досвіду та окрему цільову нейронну мережу. Повний алгоритм представлений в лістингу 4.3. Цей алгоритм був обраний на заміну оригінальному підходу DQL, який використовував лише одну нейронну мережу, оскільки він мав покращену продуктивність з точки зору швидкості навчання та стабільності.

Лістинг 4.3 – Алгоритм DQL-навчання з ε-жадібною політикою вибору дії

```
Ініціалізуємо функцію оцінки дії Q випадковими вагами параметру θ
Ініціалізуємо пам'ять D від об'єму N
Ініціалізуємо цільову функцію оцінки дії Q'

for епізод = 1,V:
      s_1=початковий стан
      for крок=1,T:
            З вірогідністю ε обираємо випадкову дію a_t
            з вірогідністю 1-ε обираємо a_t = argmax_a Q(s_t, a)
            Виконуємо дію a_t та отримуємо винагороду r_t та стан s_{t+1}
```

Записуємо перехід у пам'ять D (s_t, a_t, r_t, s_{t+1})

Витягуємо випадкову множину переходів (s_t, a_t, r_t, s_{t+1}) з пам'яті D

Оцінюємо поточну Q за допомогою y_j

$$y_j = \begin{cases} r_j, \text{ якщо } s_j \text{ кінцеве} \\ r_j + \gamma max_{a'} Q'(s_{j+1}, a'; \theta') \end{cases}$$

Оновлюємо значення параметрів θ, виконуючи градієнтний зпуск по $(y_j - Q(s_j, a_j; \theta))^2$

if s_{t+1} кінцевий стан:

закінчуємо епізод

end if

$s_t = s_{t+1}$

end for

end for

Використано повністю з'єднану одношарову нейронну мережу як для основної, так і для цільової нейронної мережі, яка приймає вектор стану як вхідні дані та виводить прогнозоване значення для кожної дії. Вектор стану являє собою впорядкований масив інформації для кожної машини в мережі, у той час як на виході є впорядкований масив значень для кожного сканування та експлойту для кожної машини мережі. Також використано ε-жадібну політику вибору дії для алгоритму DQL, щоб для заданого стану наступна дія обиралась шляхом вибору дії з найбільшим прогнозованим значенням з ймовірністю 1 -ε і рівномірно випадкова випадкова дія з ймовірністю ε.

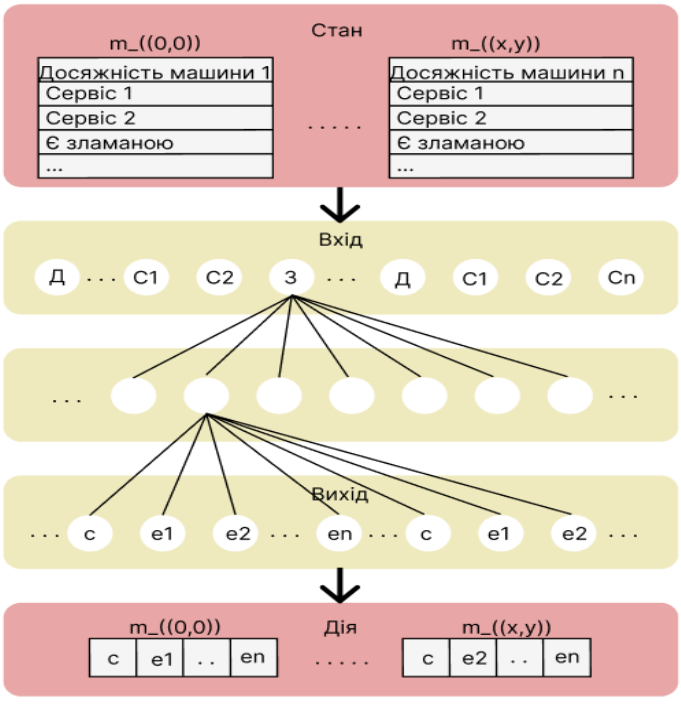


Рисунок 4.5 Схематична ілюстрація Глибокої Q-мережі.

На рис. 4.5 зображено ілюстрацію DQN, на якій вхідний шар — це стан, представлений вектором інформації для кожної машини в мережі. Єдиний прихований шар повністю пов’язаний з обома вхідним та вихідним шарами. Вихідний рівень — це значення дії для кожного сканування та експлойту для всіх комп’ютерів в мережі.

4.4 Тестування моделей

У ході цієї роботи експериментально досліджувалось застосування RL до автоматизованого тестування безпеки. Було досліджено продуктивність кожного алгоритму RL, вимірюючи продуктивність на діапазоні сценаріїв. Загальна експериментальна процедура полягала у виборі сценарію (тобто розмір мережі, топологія, кількість сервісів тощо), навчання агента протягом встановленого періоду часу, а потім оцінки політики, створеної агентом.

Зокрема, було використано три різні топології комп’ютерної мережі:

- стандартна мережа, описана в розділі 3
- мережа з однією підмережею
- глобальна мережа.

На жаль, знайти конкретні приклади мережевих проектів для використання у сценаріях, крім стандартного дизайну мережі, який базувався на практичному комерційнійеому досвіді не вдалося. Тому були розроблені два інші сценарії, щоб представити просту мережа з одним місцем розташування та мережа для кількох локацій однакового розміру.

Стандартна архітектура мережі представляє собою сценарій реального світу. Крім того, симулятор підтримує генерацію випадкових сценаріїв за допомогою проектування на основі кількості машин і кількості експлойтів. Цю функцію було використано для дослідження властивостей масштабування різних алгоритмів RL.

Всі три топології мережеві містять 16 машин, п'ять придатних для зламу

типів сервісів і дві чутливих машини. Таку кылькысть було обрано для дослідити впливу різних мережевих топологій на продуктивність алгоритму RL. Аналогічно, для всіх сценаріїв потрібно зламати мінімум три машини, щоб отримати доступ до всіх конфіденційних документів і завершити сценарій.

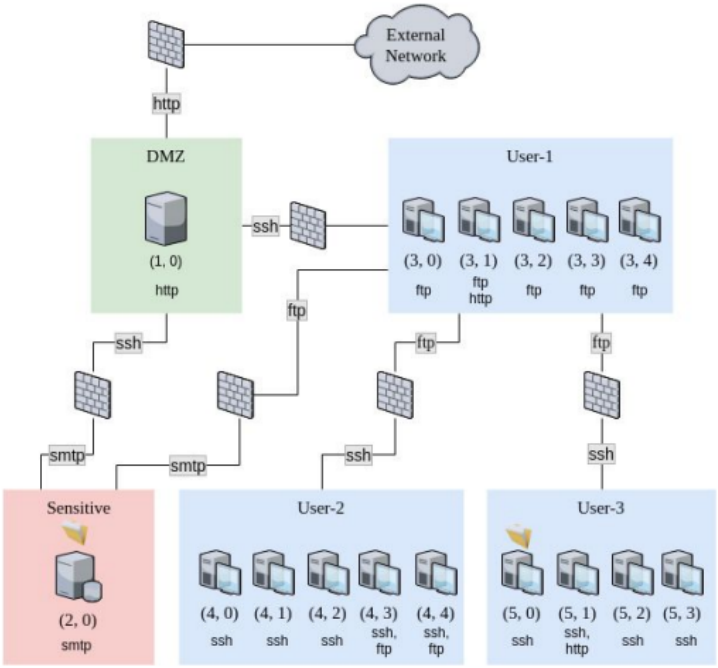


Рисунок 4.6 Стандартна топологія мережі.

Мітки під кожною машиною вказують, на служби, які можна зламати, запущені на цій машині. Мітки вздовж межі до брандмауера від підмережі вказують на можливість експлуатації трафіку, дозволеного через брандмауер. Документи над машиною вказують на цінну машину.

Для кожного сценарію використовувалось якомога більше спільних налаштувань, щоб краще з'ясувати вплив змінних, що цікавить. У таблиці 4.2 наведено докладну інформацію про різні використані значення параметрів. Вибране максимального значення кроків(дій), як правило, давало хорошу продуктивність в діапазоні розмірів використовуваних сценаріїв під час попереднього тестування. Аналогічно для значень чутливих машин та вартості дій.

Таблиця 4.2 Параметри сценарію експерименту та їх значення

Параметр	Значення	Опис

Винагорода за чутливу машину	10	Винагорода, яку агент отримує при успішному злому чутливої машини

Продовження таблиці 4.2

Вартість сканування	1	Вартість кожної дії сканування
Вартість експлойту	1	Вартість кожної дії злому
Макс. кількість дій	500	Максимальна кількість кроків в епізоді перед скиданням середовища до початкового стану

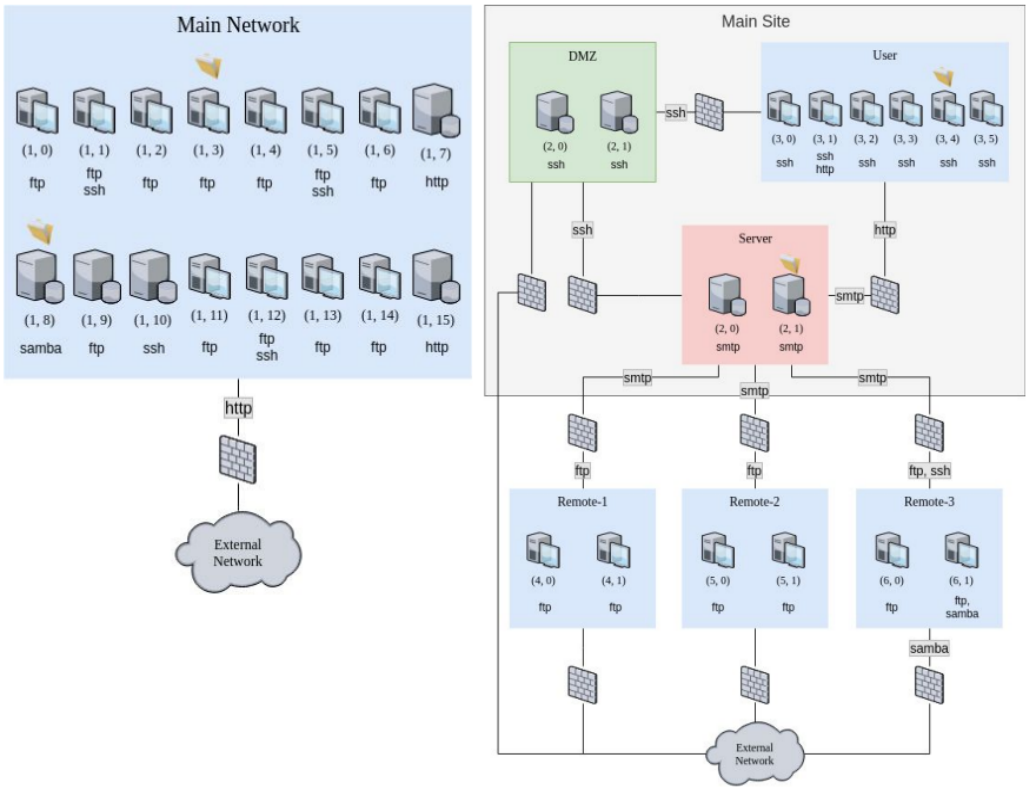


Рисунок 4.7 Топологія мережі з однією підмережею та топологія мережі з сценарієм глобальної мережі.

Для кожного сценарію кожного агента навчали протягом двох хвилин, перш ніж оцінити вивчену політику. Було вирішено навчати агентів протягом встановленого періоду часу через те, що час на дію табличного агента значно швидший, за алгоритми DQL, і оскільки, на практиці, у разі застосування RL , у реальному світі, нас буде цікавити те, скільки часу знадобиться агенту, щоб знайти шлях атаки за часом у порівнянні з часом тренування. Обмеження часу становило дві хвилини і було обрано з практичних міркувань, оскільки дозволяло протестувати досить велику різноманітність сценаріїв.

Кожен алгоритм вимагав виділення ряду значень гіперпараметрів. Вибір гіперпараметрів проводився за допомогою суміші неформального пошуку у стандартних мережах різного розміру, вибираючи варіанти за замовчуванням з використаних бібліотек машинного навчання та значення, які можна побачити в літературі для DQL. Для табличного $\boldsymbol{\varepsilon}$-жадібного і DQL ми також використовували $\boldsymbol{\varepsilon}$-спад, який впливає на зменшення $\boldsymbol{\varepsilon}$, а отже, ймовірність вибору випадкової дії, з часом від початкового значення, $\boldsymbol{\varepsilon}$ макс, до кінцевого значення, $\boldsymbol{\varepsilon}$ мінімальне (1,0 і 0,05 відповідно для обох табличних алгоритмів $\boldsymbol{\varepsilon}$-жадібних і DQL).

Таблиця 4.3 Список гіперпараметрів, що використовуються для кожного алгоритму, та їх значення

Гіперпараметр	$\boldsymbol{\varepsilon}$-жадібний	ВМД	глибоке Q-навчання
Розмір кроку константи α	0.1	0.1	-
Знижка γ	0.99	0.99	0.99
Початкове значення ε_{max}	1.0	-	1.0
Кінцеве значення ε_{min}	0.05	-	0.05
Темп спаду ε	0.0001	-	0.0001
Розмір вибірки множини переходів	-	-	32
Розмір скритого слою	-	-	256
Розмір пам’яті	-	-	10000
Темп навчання середньоквадратичного поширення (RMSProp)	-	-	0.00025
Імпульс темпу навчання	-	-	0.9

Після навчання результативність агентів оцінювалася за допомогою його навченої політики проти мережевого сценарія в симуляторі 10 або 30 разів залежно від експерименту. Політики було перевірено з використанням $\boldsymbol{\varepsilon}$-жадібного вибору дії з $\boldsymbol{\varepsilon}$ = 0,05, щоб уникнути ймовірності агента застрягти в якомусь стані назавжди і уникнути перепристосування політики до сценарію.

Для порівняння, де можливо, також був використаний випадковий агент, який вибирав випадкові дії в кожному стані.

Для сценаріїв, які використовувалися (стандартний, одна підмережа і загальна мережа), агента було навчено один раз для кожного сценарію, а потім проведено 30 епізодів проходження навченим агентом. Для експериментів, де генерувались сценарії мереж за допомогою симулятора, виконано також 10 окремих запусків агенту для кожного сценарію, використовуючи різні випадкові початкові дані для кожного запуску, і оцінили кожен із запусків 10 разів.

Для кожного з різних створених мережевих сценаріїв, було виміряно епізодичну винагороду під час навчання, а також остаточну продуктивність навченаої політики кожного алгоритму RL. На рис. 4.8 показано середню епізодичну винагороду за тренувальними епізодами. Винагороду за останні 100 епізодів усереднено, щоб отримати більш плавний графік винагороди. Усі три алгоритми сходилися до наближеного оптимального рішення протягом ліміту часу навчання, що показано на графіках зближенням середньої епізодичної винагороди до теоретичного максимума (червона пунктирна лінія).

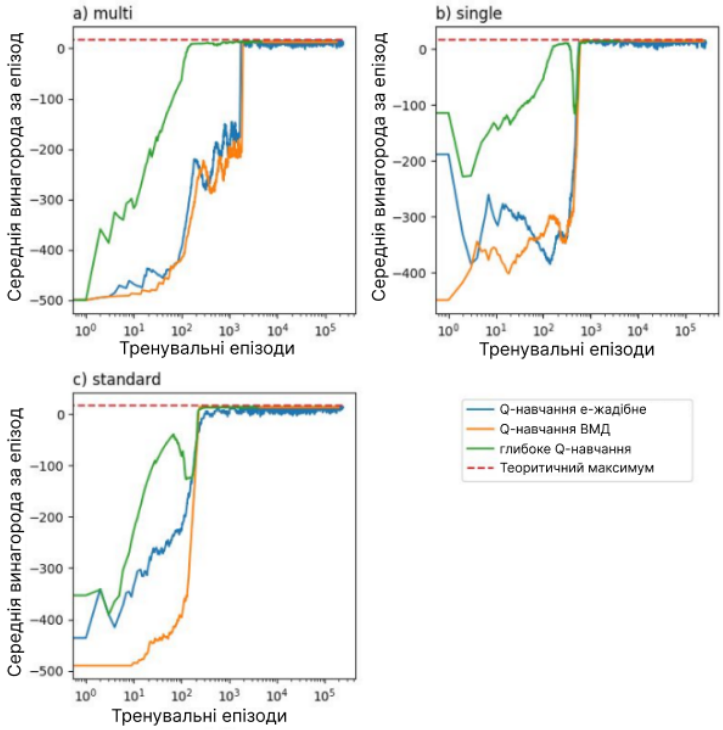


Рисунок 4.8 Середня епізодична винагорода на епізодах навчання

Теоретичний максимум базується на детермінованих діях і є значенням загальної винагороди чутливих машин мінус загальна вартість мінімальної кількості експлойтів, необхідних для їх компрометації з початкового стану. Для одномережевих і стандартно мережевих сценаріїв збіжність відбулася після однакової кількості епізодів для всіх трьох алгоритмів (~1000 епізодів для одно мережевої топології і ~ 150 епізодів для стандартної мережі). Для багатомережевого сценарію, алгоритм DQL збігався значно швидше, після ~100 епізодів порівняно з >1000 епізодами для двох табличних алгоритмів.

Також порівняно середню епізодичну винагороду протягом часу для різних алгоритмів RL (рис. 4.9). На відміну від винагороди по епізодам, де DQL, як правило, навчався швидше, обидва табличні алгоритми сходилися до наближеного оптимального рішення значно швидше, ніж DQL з точки зору часу. Для сценаріїв з однією підмережею і стандартної мережі, табличні методи зійшлися за <10 секунд, тоді як DQL зайняв ~50 секунд для стандартної мережі та ~75 секунд для мережі з однією підмережею. Для сценарію глобальної мережі час збіжності був більше схожий з усіма алгоритмами, які збігаються за <25 секунд, проте DQL все ще був повільнішим на ~5 секунд.

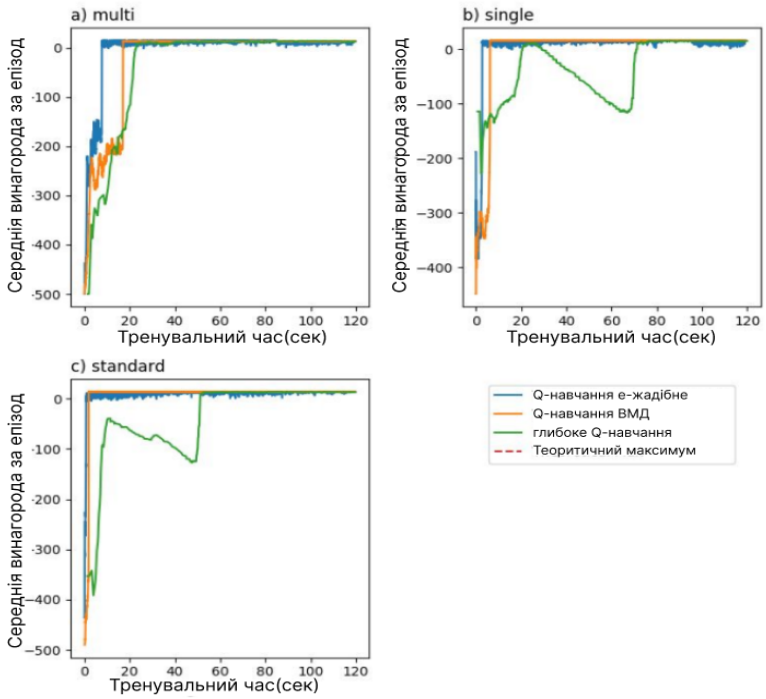


Рисунок 4.9 Середня епізодична винагорода з часом навчання

Ця різниця в часі збіжності, ймовірно, пов’язана з часом, за який кожен алгоритм завершує епізод. Щоб виміряти це, було записано кількість навчальних епізодів, завершених кожним алгоритмом протягом двох хвилинного періоду навчання (таблиця 4.4). Табличні методи були значно швидші, ніж алгоритм DQL, при цьому табличний $\boldsymbol{\varepsilon}$-жадібний алгоритм зробив в 50 разів більше епізодів, ніж DQL у гіршому випадку, а табличний ВМД алгоритм зробив в 37 разів більше епізодів, ніж DQL в гіршому випадку. Ця різниця була очікувана через додаткові обчислення, необхідні для обчислення нейронної мережі DQL.

Таблиця 4.4 Кількість навчальних епізодів для кожного алгоритму та сценарію навчання з підкріпленням, протягом двох хвилин тренування

Сценарій	$\boldsymbol{\varepsilon}$-жадібний	ВМД	глибоке Q-навчання
Стандартна мережа	241 101	193 829	3989
Мережа з однією підмережею	260 526	243 599	4020
Глобальна мережа	235 142	169 448	4712

Після двохвилинного періоду навчання для кожного алгоритму RL для кожного сценарію було виміряно продуктивність остаточно навчених політик, використовуючи ε-жадібний вибір дії з ε =0,05. Було записали частку 30 оціночних епізодів, які були вирішені агентом, де епізод вважався вирішеним, коли агент успішно зламував чутливі машини в мережі в межах обмеження кількості дій (500 кроків). Також знайдено максимальну винагороду досягнуту агентом для кожного сценарію разом із відхиленням винагороди за 30 епізодів. Для порівняння також вимірювалася ефективність агенту з випадковою політикою.

Результати по 30 епізодах навченого агента показані на малюнку 4.10 і записані в таблицю 4.5. Усі три алгоритми та агент з випадковою політикою

змогли вирішити кожен сценарій на принаймні деяких епізодах, при цьому три агенти RL працюють значно краще, ніж агент з випадковою політикою на глобальній та стандартній мережевих сценаріях. Для сценарію мережі з однією підмережею табличний ε-жадібний агент насправді працював гірше, ніж випадковий агент (0,9 проти 0,97 відповідно), тоді як табличні алгоритми UCB і DQL працювали однаково добре або краще. Тим гірше, продуктивність табличного ε-жадібного, ймовірно, пов'язана з вибором дії ε-жадібного в політиці оцінки дії, що призводить до того, що агент застряг у стані, який зазвичай не зустрічається у звичайних умовах шляху до вирішення. Крім того, продуктивність випадкового агента дуже висока для цього сценарію через його простоту порівняно з іншими сценаріями. Єдиний алгоритм, здатний вирішити 100% всіх епізодів був алгоритм DQL.

Усі алгоритми змогли досягти приблизно оптимальної максимальної винагороди для кожного сценарію, як показано на графіку максимальної винагороди на малюнку 4.10. Випадковий агент зміг розгадати кожен сценарій, однак, щоразу потрібно було значно більше дій, що показано негативною максимальною винагородою. Продуктивність алгоритму DQL була найбільш стабільною в усіх сценаріях, ймовірно, це пов’язане з кращим узагальненням його політики щодо невидимих ще не бачених станів та станів що знаходятся не по оптимальному шляху атаки.

Таблиця 4.5 Пропорція вирішення та максимальної винагороди, досягнутої для кожного сценарію та алгоритму навчання з підкріпленням після навчання

Сценарій	Випадкові дії	$\boldsymbol{\varepsilon}$-жадібний	ВМД	глибоке Q-навчання
Стандартна мережа	0.53 -177(± 18.94)	1.0 15(± 0.32)	0.97 15(± 16.33)	1.0 16(± 0.3)
Мережа з однією підмережею	0.97 -48(± 24.91)	0.9 16(± 28.53)	0.97 17(± 17.12)	1.0 17(± 0.23)

Глобальна мережа	0.4 -99(± 26.91)	1.0 16(± 0.31)	0.97 16(± 16.82)	1.0 16(± 0.3)

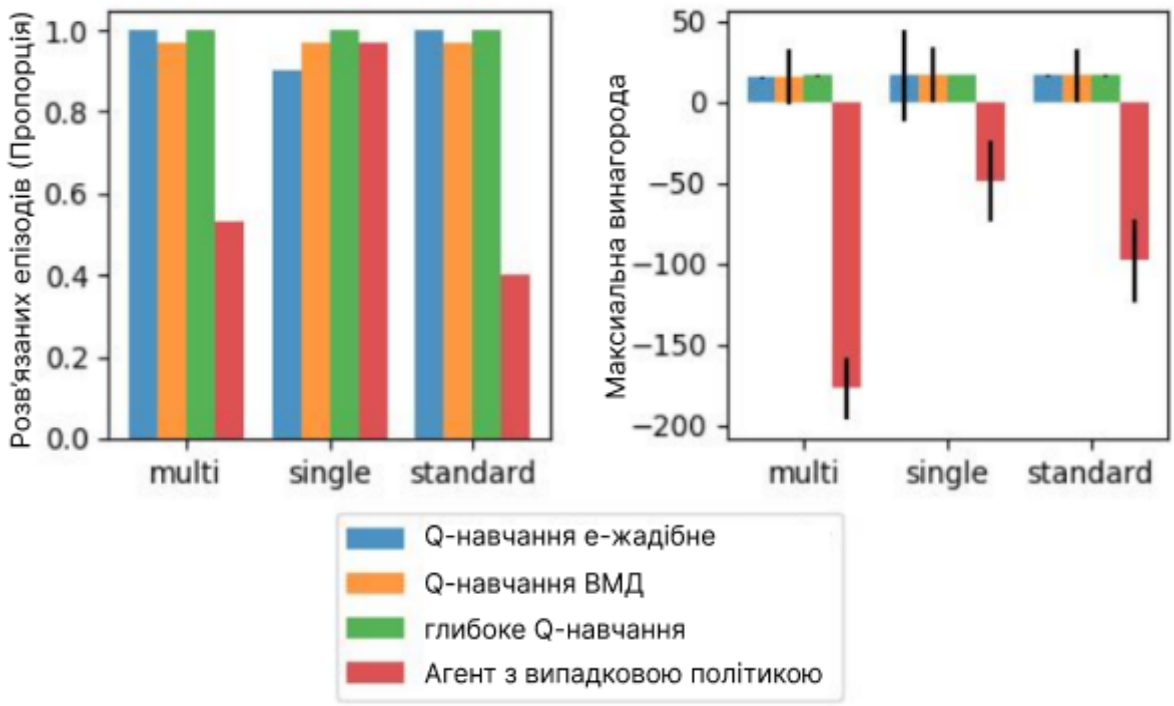


Рисунок 4.10 Оцінка ефективності алгоритмів навчання з підкріпленням для кожного сценарію

Пропорція розв'язаних епізодів (ліворуч), показує частку з 30 вирішених епізодів. На (правому) зображенні показана максимальна винагорода, досягнута за 30 епізодів.

Важливим показником тестування була продуктивність масштабування кожного алгоритму з точки зору розміру мережі та кількості доступних дій. Для кожного сценарію мережі було перевірено 10 різних випадкових конфігурацій, а потім виміряно продуктивність у кожній конфігурації після двох хвилин навчання з використанням 10 оціночних епізодів. Ефективність вимірювалась за допомогою двох показників. Перший це середня розв'язана пропорція, яка представляла собою середнє значення пропорції розв'язаних епізодів для кожної відрізняючоючись конфігурації, де мережа була розв'язана, коли всі чутливі машини було зламано в межах 500 дій. Другим показником

була середня винагорода, отримана під час розв’язання епізодів.

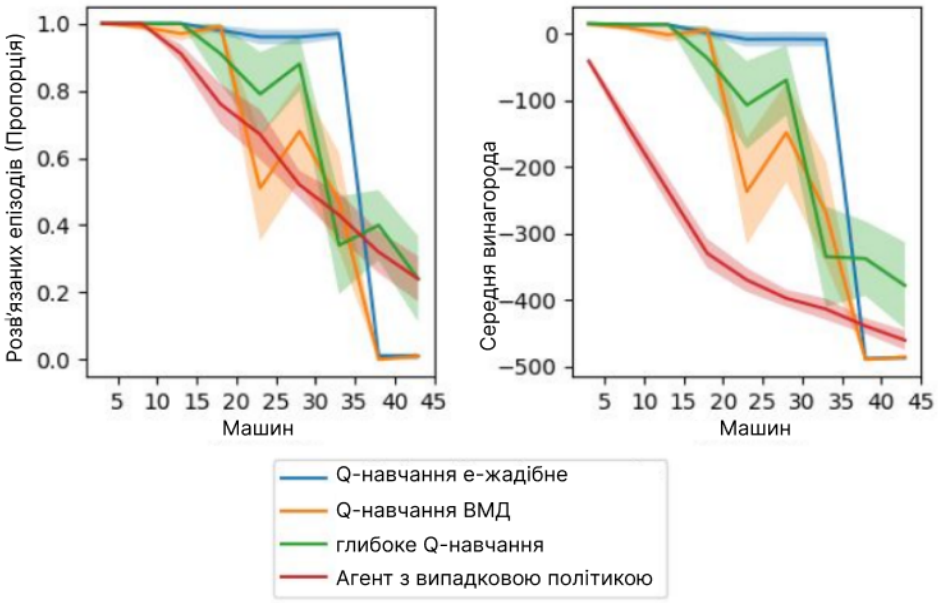


Рисунок 4.11 Оцінка ефективності алгоритмів RL для згенерованих сценаріїв стандартного типу залежно від кількості вразливих машин

Можно побачити продуктивність алгоритму навчання підкріплення залежно від кількості машин, автоматично створених у стандартній мережі з фіксованою кількістю сервісів(5), які можна використовувати. Вплив розміру мережі було перевірено шляхом збільшення кількості машин намережі від 3 до 43 машин з інтервалом по 5 із збереженням кількості вразливих сервісів зафіксованих на 5. На рис. 4.11 показані результати експериментів. Продуктивність була рівною і кращою у агентів з RL, ніж у агентів з випадковою політикою для всіх трьох алгоритмів до мереж, що містять 18 машин. Для протестованих мережі з більш ніж 18 машинами, продуктивність DQL і табличного ВМД алгоритмів швидко занепадали, хоча обидва алгоритми все ще могли вирішити понад 50% сценарії для мереж з 23 і 28 машинами і призводити середні винагороди все що є краще, ніж випадковий вибір дій. Продуктивність для табличного ε-жадібного алгоритму була стабільно високою для мережі включно до 33 машини, після чого продуктивність швидко впала до гіршого значення, аніж у політиці випадковх дії для мереж з 43 машинами.

Виміряно продуктивність зі збільшенням кількості експлойтів(дій), доступних для агентів використовуючи мережу фіксованого розміру з 18

машин і розширюючи доступні для експлуатації сервіси 1 до 51 з інтервалами по 5. Результати експерименту показані на малюнку 4.12. Ефект від збільшення кількості експлойтів відрізнялася для кожного алгоритму. Продуктивність відносно не вплинула для табличного ε-жадібного алгоритму, який підтримував майже оптимальну продуктивність для будь-якої кількості експлойтів. Так само табличні ВМД мали нижчу за оптимальну продуктивність, але продуктивність залишилася відносно постійно, оскільки кількість експлойтів зростала. Найбільше постраждав DQL, який мав продуктивність порівняну з випадковою для всіх перевірених значень.

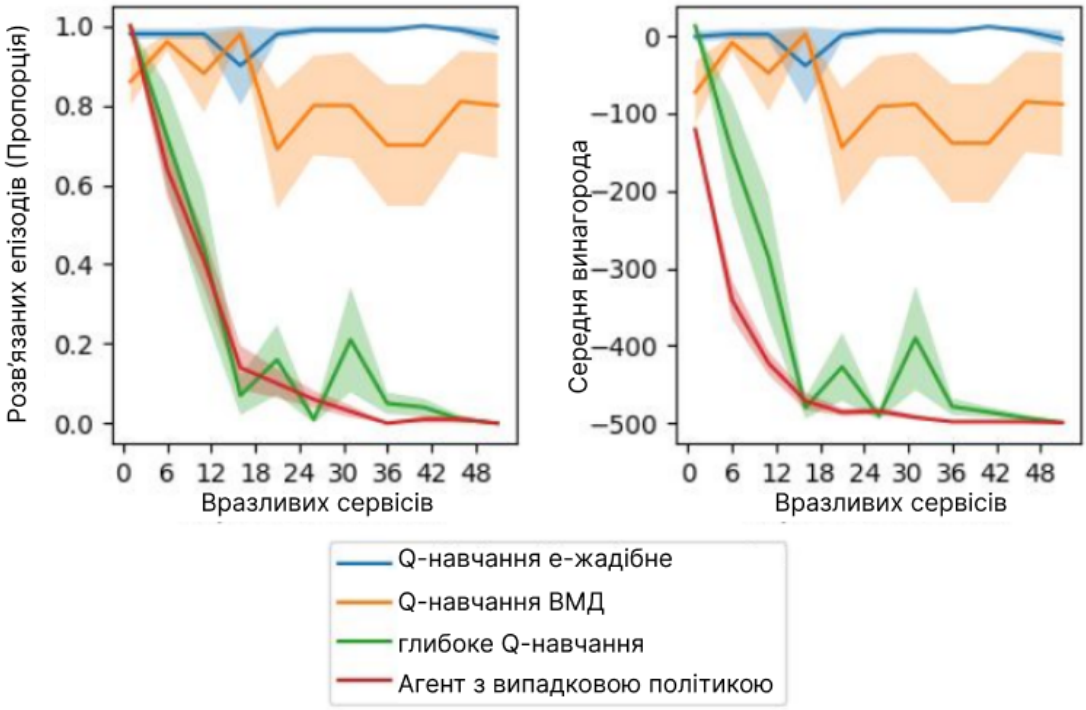


Рисунок 4.12 Оцінка ефективності алгоритмів RL для згенерованих сценаріїв залежно від кількості вразливих сервісів

Для кожного сценарію всі тестовані алгоритми RL змогли знайти політику атаки, яка призведе до того, що всі чутливі машини в цільовій мережі будуть скомпрометовані, з мінімальною кількістю вжитих дій, роблячи значно кращі результати, ніж агент з випадковою політикою з точки зору досягнутої максимальної винагороди. На основі цих результатів, у межах змодельованого

середовища, RL може знайти дійсний шлях атаки (1), який також є оптимальним у плані досягнутої винагороди, зважаючи на вартості дій. Алгоритми також можуть бути застосовані до різних конфігурацій мереж.

Хоча ці результати дають багатообіцяючі докази використання RL в автоматизованому тестуванні, варто слід не забувати про деякі обмеження. Зокрема, сценарії були обмежені мережами, що містять 16 машин з п'ятьма доступними для експлуатації службами і тому були відносно невеликими за розміром у порівнянні з великими комерційними мережами, що налічують сотні машини, які можна знайти в реальних умовах. Також, агентам було дозволено лише дві хвилини тренування з найповільнішим часом для зближення, що становив приблизно 80 секунд для алгоритму DQL. Тому слід очікувати, що агенти RL можна застосувати до задач більшого розміру, за умови більшого часу навчання. Іншим обмеженням було те, що перевірялись лише три різні топології мережі, у той час як можливі топології, які зустрічаються в реальному світі є нескінченними. Однією з ключових переваг RL є його загальність, про що свідчить його успіх у досягненні людського рівня або кращої продуктивності в широкому діапазоні дуже різних ігр з використанням єдиного алгоритму проектування.

Щоб дослідити, як продуктивність RL відрізняється від розміру мережі, було розглянуто масштабованість RL. Масштабованість різних алгоритмів RL була виміряна в обох термінах кількості машин у мережі, а також кількості дій, доступних для агента. Було виявлено, що для всіх перевірених алгоритмів їх продуктивність впадала швидко, як тільки розмір мережі виходить за межі певного розміру (18 машин для табличних ВМД і DQL і 38 машин для табличного ε-жадібний алгоритму). Падіння продуктивності було очікуване, оскільки для задачі пентестування, як було визначено, розмір простору станів зростає експоненціально з кількістю машин в мережі, а час навчання залишався постійним і становив дві хвилини. Для більших мереж, збільшення часу навчання, ймовірно, призведе до покращення продуктивності, принаймні до якоїсь точка. Відомо, що продуктивність табличних алгоритмів RL

погіршується, коли простір станів стає дійсно великим, оскільки вони не узагальнюють оцінку дії між подібними станами. Це можна побачити у різкому падінні продуктивності до нижчої, ніж у агента з випадковою політикою для табличного RL алгоритму для мережі з 43 машинами, тоді як алгоритм DQL все ще здатний перевершувати політику випадкового використання для мережі такого розміру. Можно очікувати, що використання DQL або пов'язаного функціонально-апроксимаційного підходу до RL буде мати можливість масштабуватися до набагато більшимих мереж, якщо буде достатньо часу на навчання. Як приклад масштабованості, DQL агент AlphaGo RL здатний видавати кращу, за людський рівень продуктивность в грі з простором станів близько 10^{174} станів.

Іншим підходом до обробки експоненціально збільшуваного розміру простору станів було розкладення проблеми на двох окремих агентів. Один агент буде працювати над топологією мережевого рівня, вирішуючи, на яку машину слід орієнтуватися наступною, тоді як другий агент працює на індивідуальною машиною, вирішуючи, які дії використовувати проти даної машини. Такий підхід добре працює для автоматизованого пентестування на основі моделі з використанням POMDP. Один перспективних підходів до автоматизованого пентестування без моделі було б використання ієрархічної RL, який є технікою, яку можна застосувати до MDP для розкладання дій на різні рівні абстракцій, однак це все ще активно досліджуется у сфері RL, і тому може пройти деякий час, перш ніж він стане надійним та ефективним методом. Можливість збільшення області станів/дій RL за допомогою більш ефективних методів наближення функцій або шляхом декомпозиції задачі, дає основу вважати, що RL добре застосовується і до дуже великих мереж.

Крім великого розміру мережі, ще однією з проблем, з якими стикається пентестування, є складність і швидка еволюція сервісів та набору доступних експлойтів. Щороку знаходять нові експлойти для програмного забезпечення та, щоб бути корисним для автоматизованого агента пентестування, потрібно буде обробляти велику базу даних експлойтів, що розростається. Було

досліджено продуктивність алгоритмів RL у міру збільшення кількості доступних експлойтів. Було виявлено, що на продуктивність табличних алгоритмів RL відносно не вплинула збільшена кількість експлойтів, при цьому табличний $\boldsymbol{\varepsilon}$-жадібний алгоритм підтримує майже оптимальну продуктивність і вирішує майже 100% з перевірених сценаріїв.

ВИСНОВКИ

У процесі виконання магістерської роботи на тему «Тестування безпеки комп'ютерних систем з використанням навчання з підкріпленням» був розроблений симулятор мережі для навчання моделей штучного інтелекту з атаки на машини, програму-агент для тестування на проникнення на базі навчання з підкріпленням.

При реалізації даного проекту були використані сучасні технології, такі як машинне навчання. Симулятор мережі був розроблений на базі віртуальних моделей із спрощеними інтерфейсами. Програма для тестування на проникнення була розроблена на базі Q-алгоритму та бази даних вразливостей Metasploit. Розробка проводилась на базі мови програмування Python. Дана розробка повністю відповідає тематиці магістерської роботи проекту, а всі поставлені в роботі задачі було виконано в повному обсязі.

Подальший розвиток розробленого програми можна забезпечити за допомогою виконання наступних покращень:

– Додати можливість кастомізації симулятора мережі через інтерфейс користувача;
– Оптимізація та збільшення швидкості роботи алгоритмів;
– Покращення функціоналу для кращої роботи з великомасштабними мережами;
– Розширення фунціоналу;

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Python Machine Lerning [Електронний ресурс]. – Режим доступу : www/ URL: https://www.w3schools.com/python/python_ml_getting_started.asp – Загол. з екрану

2. Machine Lerning [Електронний ресурс]. – Режим доступу : www/ URL: https://www.ibm.com/cloud/learn/machine-learning – Загол. з екрану

3. Машинное обучение в Offensive Security [Електронний ресурс]. – Режим доступу : www/ URL: https://habr.com/ru/company/pm/blog/419617/ – Загол. з екрану

4. Reinforcement learning [Електронний ресурс]. – Режим доступу : www/ URL: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ – Загол. з екрану

5. Разработка приложений на Python [Електронний ресурс]. – Режим доступу : www/ URL: https://visualstudio.microsoft.com/ru/vs/features/python/ – Загол. з екрану

6. Metasploit [Електронний ресурс]. – Режим доступу : www/ URL: https://www.metasploit.com – Загол. з екрану

7. B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010

8. МОДЕЛИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ [Електронний ресурс]. – Режим доступу : www/ URL: http://nmetau.edu.ua/file/kapp_5435.pdf – Загол. з екрану

9. T. J. Holt, O. Smirnova, and Y. T. Chua, “Exploring and Estimating the Revenues and Profits of Participants in Stolen Data Markets,” Deviant Behav., pp. 353–367. 2016.

10. Goodfellow, L. Bengio, Y. & Courville, A. (2016). Deep learning. MIT Press. URL: http://www.deeplearningbook. Org

11. B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” IEEE Secur. Priv., vol. 3, no. 1, pp. 84–87, 2005.

12. Cisco, “Mitigating the Cyber Skills Shortage,” CISCO, 2015.

13. C. Sarraute, Automated Attack Planning. 2012.

14. C. Sarraute, O. Buffet, and J. Hoffmann, “POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing,” AAAI, 2012

15. R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” 2011.

16. D. Silver et al., “Mastering the game of Go without human knowledge,” Nature, vol. 550, no. 7676, pp. 354–359, 2017.

17. F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, “Effective penetration testing with Metasploit framework and methodologies,” presented at the CINTI 2014 - 15th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings, 2014, pp. 237–242

18. M. S. Boddy, J. Gohde, T. Haigh, and S. A. Harp, “Course of Action Generation for Cyber Security Using Classical Planning,” in ICAPS, 2005, pp. 12–21.

19. R. Bellman, “A Markovian Decision Process,” Journal of Mathematics and Mechanics, vol. 6, no. 5, pp. 679–684, 1957.

20. M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender, “Complexity of finite-horizon Markov decision process problems,” J. ACM, vol. 47, no. 4, pp. 681–720, 2000.

21. V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, p. 529, Feb. 2015

22. G. F. Riley and T. R. Henderson, “The ns-3 Network Simulator,” in Modeling and Tools for Network Simulation, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

23. B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, p. 19.

24. A. Futoransky, F. Miranda, J. Orlicki, and C. Sarraute, Simulating Cyber-Attacks for Fun and Profit. 2010.

25. T. E. Oliphant, Guide to NumPy. USA: Trelgol Publishing, 2006.

26. Martín Abadi et al., “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.” 2015.

27. M. Backes, J. Hoffmann, R. Künnemann, P. Speicher, and M. Steinmetz, “Simulated Penetration Testing and Mitigation Analysis,” arXiv:1705.05088, May 2017.

28. S. Donnelly, “ ’Soft Target: The Top 10 Vulnerabilities Used by Cybercriminals,” Recorded Future, 2018.

29. P. Mell, K. Scarfone, and S. Romanosky, “Common Vulnerability Scoring System,” IEEE Secur. Priv., vol. 4, no. 6, pp. 85–89, 2006.

30. First, “Common Vulnerability Scoring System SIG,” First. [Online]. Available: https://www.first.org/cvss/. [Accessed: 31-Oct-2018].

31. Oracle, Oracle VM VirtualBox. 2017.

32. Rapid, Metasploitable 2 Virtual Machine. 2015.

33. C. J. Watkins and P. Dayan, “Q-learning,” Mach. Learn., vol. 8, no. 3–4, pp. 279–292, 1992.

34. Adamov, A., Carlsson, A., “Reinforcement Learning for Anti-Ransomware Testing”, 2020