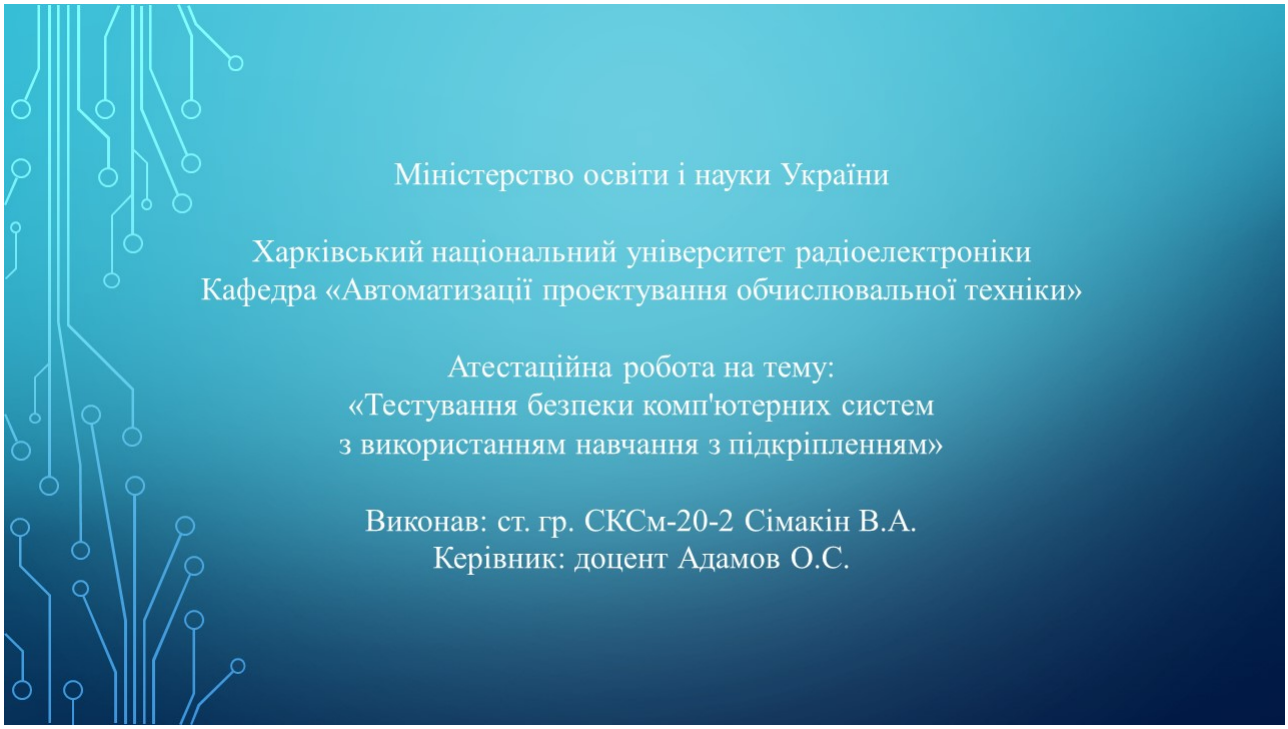
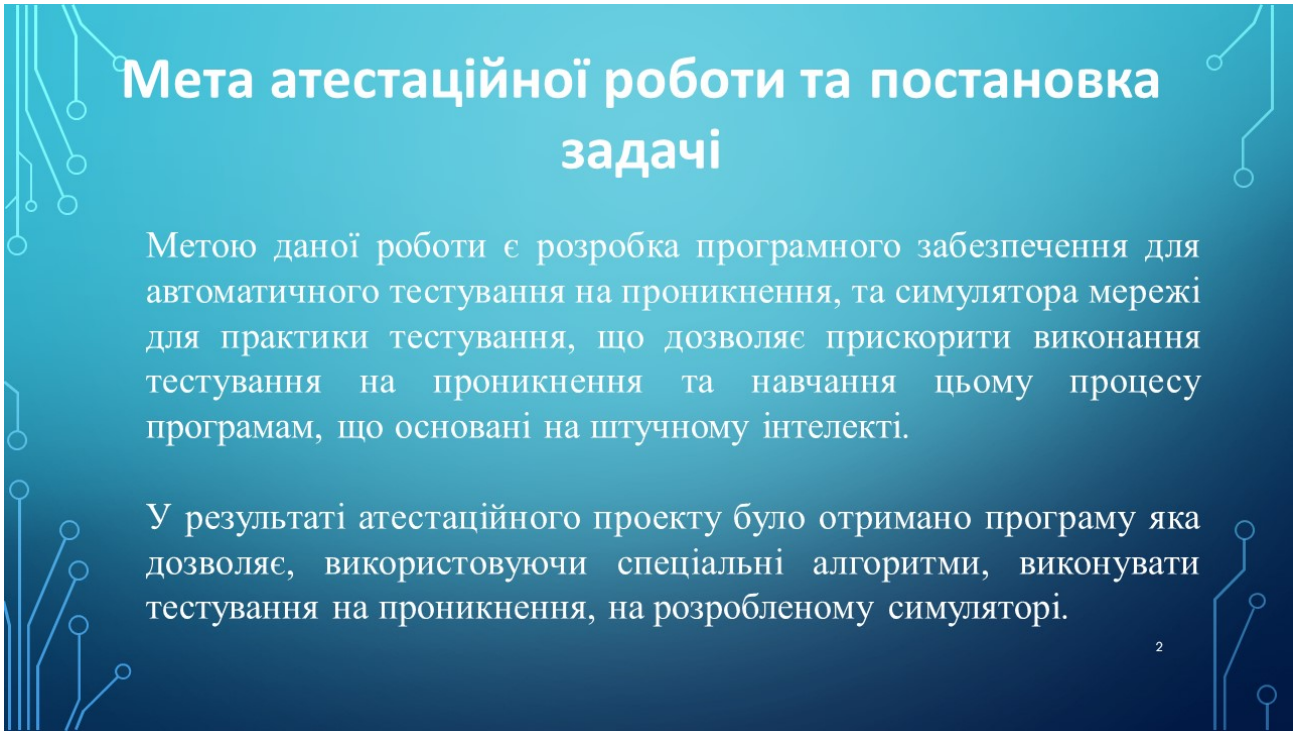
ДОДАТОК А


Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра «Автоматизації проектування обчислювальної техніки»
Атестаційна робота на тему:
«Тестування безпеки комп'ютерних систем
з використанням навчання з підкріпленням»
Виконав: ст. гр. СКСм-20-2 Сімакін В.А.
Керівник: доцент Адамов О.С.


Мета атестаційної роботи та постановка задачі
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 штучному інтелекті.
У результаті атестаційного проекту було отримано програму яка дозволяє, використовуючи спеціальні алгоритми, виконувати тестування на проникнення, на розробленому симуляторі.
2

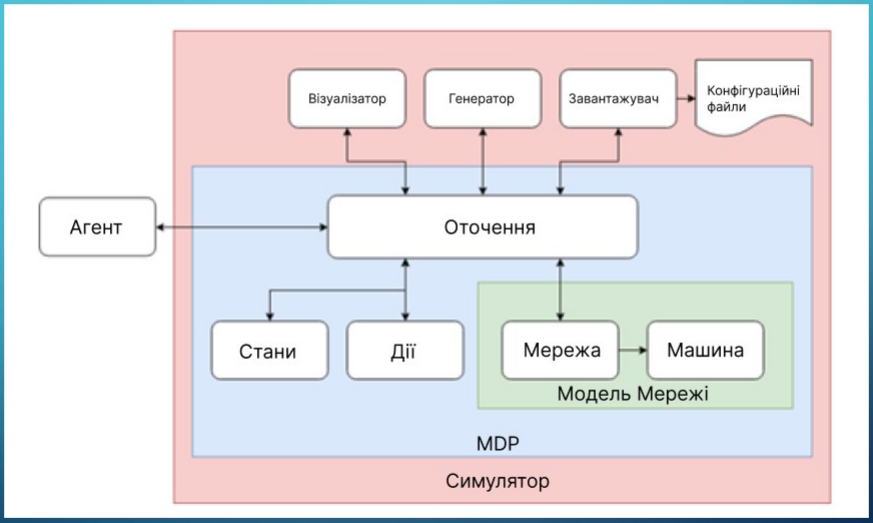
Актуальність

Тема тестування безпеки у сьогоднішніх реаліях є дуже актуальною. На сьогоднішній день існує багато видів програм для автоматичного тестування безпеки, але жодна з них не є повністю автономною.

На даний момент зростає нестача кваліфікованих фахівців у сфері тестування безпеки. Автоматизація тестування вже відомих вразливостей та повсякденних дій, таких як сканування портів системи дозволяє позбавитися від рутинної роботи висококваліфікованим фахівцям та частково вирішити проблему нестачі робітників.

3

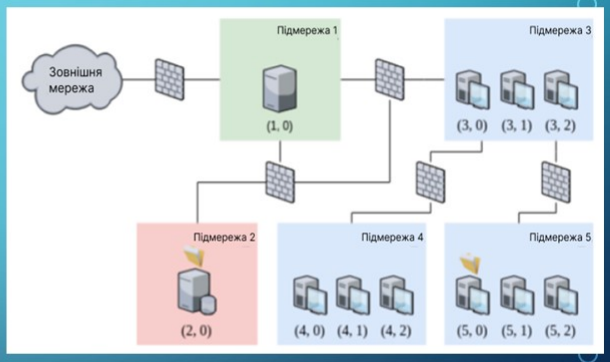
АРХІТЕКТУРА СИМУЛЯТОРА МЕРЕЖІ



4

ПІДМЕРЕЖІ

- Кожна мережа складається з декількох підмереж;
- Кожна підмережа має свою адресу підмережі, яка вказана як перше число в будь-якій машинній адресі (наприклад, 4 у адресі (4, 0)), що є спрощенною ip адресою;
- Зв'язок між підмережами контролюється топологією мережі та налаштуванням брандмауера.

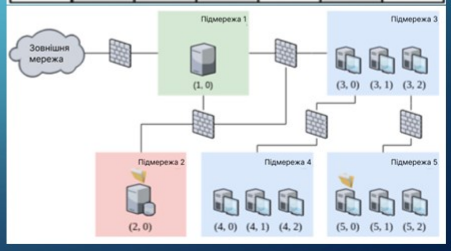


5

ТОПОЛОГІЯ МЕРЕЖІ

- Топологія мережі визначає, як підключені різні підмережі, та керує як підмережі можуть взаємодіяти безпосередньо між собою та із зовнішньою мережею;
- На рисунку мережі, підмережа 1 є єдиною мережею, яка підключена до зовнішнього світу і підмереж 2 і 3, що є пов'язані між собою. Тоді як для зв'язку з машинами в підмережах 4 і 5 через обов'язково використовувати машину в підмережі 3.
- Таким чином, зловмисникові, можливо, доведеться переміщайтеся по машинах у різних підмережах, щоб мати змогу досягти цільових машин.

Підмережа	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



МАШИНИ ТА БРАНДМАУЕРИ

- 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;
- Сервіси, доступні на кожному апараті визначаються його конфігурацією, і кожна машина в підмережі не обов'язково матиме однакові конфігурації.;
- Брандмауери, існують 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.
- Машина вважається скомпрометованою, якщо експлойт було успішно використано проти неї.

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

```
Firewall_3: {
        connection : (1, 3)
        permitted: {ssh}
}
```

```
Firewall_3: {
        connection : (3, 1)
        permitted: {ftp, http}
}
```

7

СЕРВІСИ

- Кожна машина запускає сервіси, з якими можна зв'язатись з інших машин;
- Сервіси аналогічні програмному забезпеченню, яке слухає на відкритому порту комп'ютера або підключеного пристрою;
- Для кожної можливої служби в мережі існує відповідна дія експлойту;
- 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;

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

8

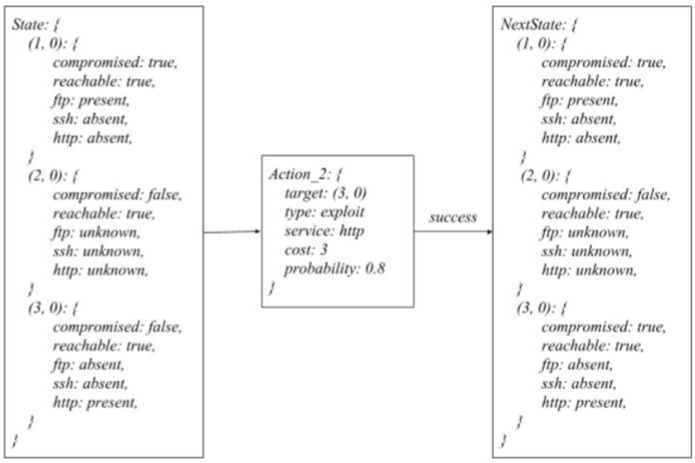
ОТОЧЕННЯ АГЕНТА

- Компонент оточення моделює проблему пентестування мережі як MDP і визначається кортежем $\{\boldsymbol{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{T}}\}$;
- Стани — це всі можливі комбінації скомпрометованих, доступних і сервісних знанньь для кожного сервісу та для кожної машини.;
- Простір дій, $\boldsymbol{\mathcal{A}}$, — це набір доступних дій у симуляторі і включає окремі сканування та експлойти для кожної служби та кожної машини в мережі;
- Функція винагороди — це вартість будь-яких експлуатованих машин мінус вартість виконаних дій;
- Функція переходу, $\boldsymbol{\mathcal{T}}$, визначає, як середовище розвивається з часом після виконанх дій.

9

ОТОЧЕННЯ АГЕНТА

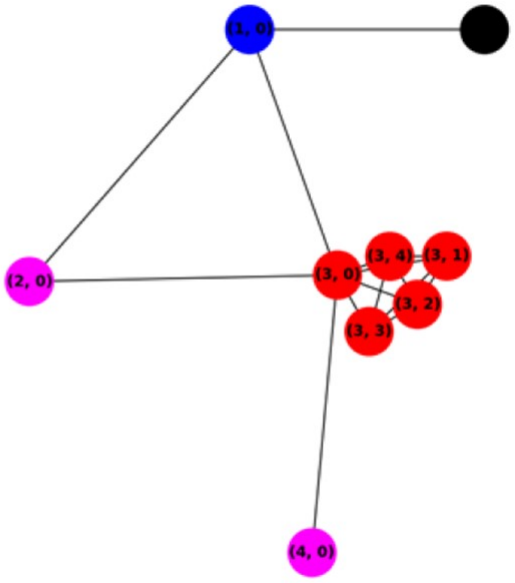
- Для симулятора наступний стан залежить від того, була дія успішною чи ні.



10

ВІЗУАЛІЗАЦІЯ МЕРЕЖІ

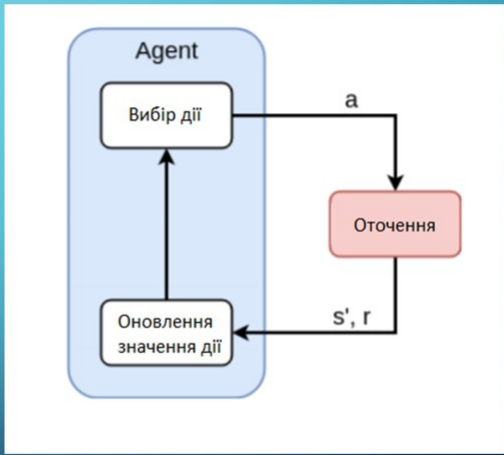
- Чорний вузол представляє позицію агента;
- Рожеві вузли — це цільові машини, які містять «конфіденційні документи»;
- Червоні і рожеві вузли це машини, які недоступні агенту;
- Сині вузли це машини, доступні для агента.



11

АЛГОРИТМ НАВЧАННЯ З ПІДКРІПЛЕННЯМ(RL)

- Алгоритми RL вивчають оптимальні дії завдяки взаємодії з оточенням.
- Окрім середовища та агенту, компонентами RL є політика π, функція винагороди, $\mathcal{R}$ (s ’, a, s), функція ваги(оцінки стану), V (s), і модель навколишнього середовища, модель переходів $\mathcal{T}$



12

АЛГОРИТМ Q-НАВЧАННЯ

- Q – навчання 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.
- Q-навчання - це поза політичний алгоритм для вивчення значень стан-дія та визначається рекурсивною функцією оновлення у рівнянні, де γ - коефіцієнт дисконтування, α це темп навчання.

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

13

ПОЛІТИКА ВИБОРУ ДІЙ ε-greedy

- Використані стратегії вибору дій 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.

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

14

ПОЛІТИКА ВИБОРУ ДІЙ ВМД

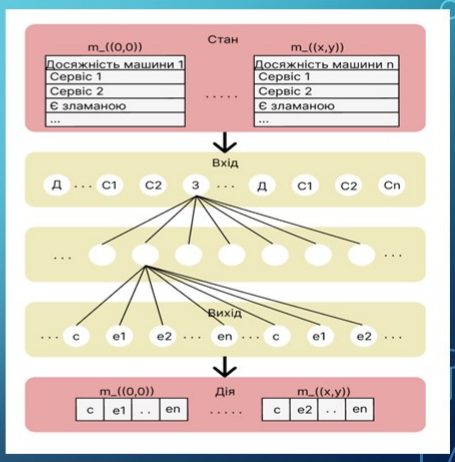
- Політика вибору дій верхніх меж довіри використовує додатковий вираз для дослідження, виконуючи вибір дій.
- Цей додатковий вираз збільшує цінність дій, які були прийняті рідше і діє для вимірювання визначеності оцінки вартості дії.

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

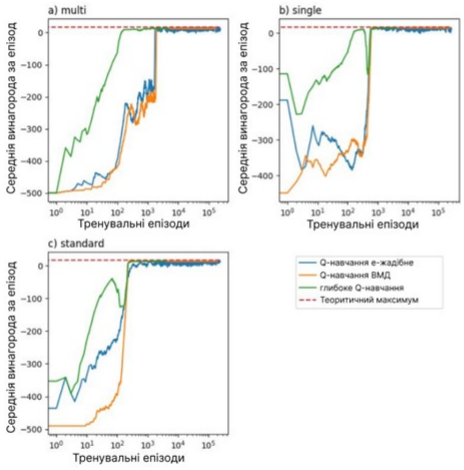
15

ГЛИБОКЕ Q НАВЧАННЯ

- Використано повністю з’єднану одношарову нейронну мережу як для основної, так і для цільової нейронної мережі;
- Приймає вектор стану як вхідні дані та виводить прогнозоване значення для кожної дії;
- Додатково була додана функція пам’яті;
- Використано ε-жадібну політику вибору дії для алгоритму DQL,

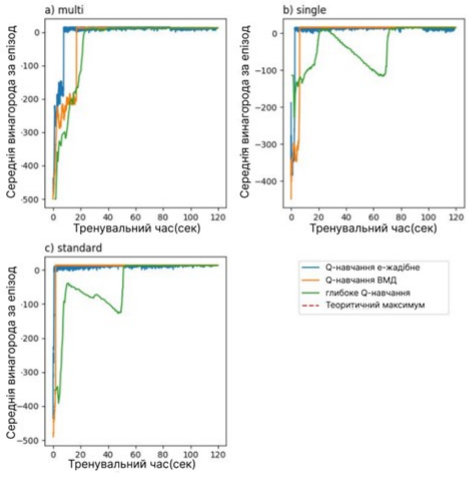


16



СЕРЕДНЯ ВИНАГОРОДА ПІД ЧАС НАВЧАННЯ АГЕНТІВ ЗА КІЛЬКІСТЮ ЕПІЗОДІВ

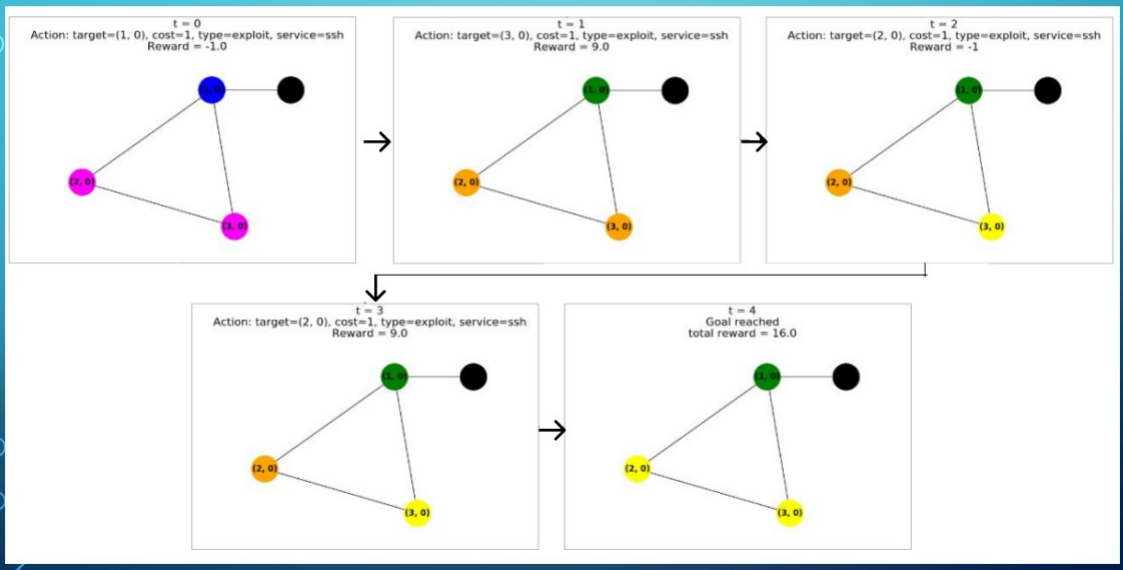
17

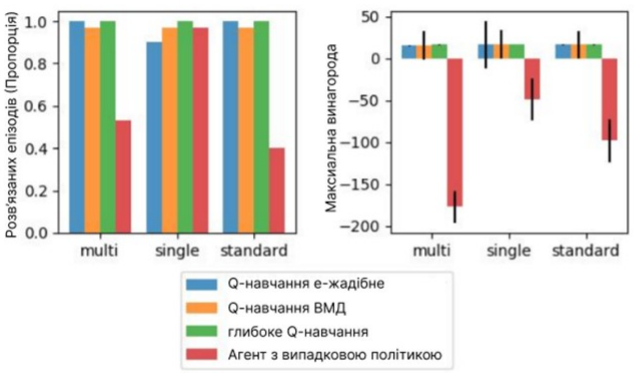


СЕРЕДНЯ ВИНАГОРОДА ПІД ЧАС НАВЧАННЯ АГЕНТІВ ЗА КІЛЬКІСТЮ ЧАСУ НАВЧАННЯ

18

РОБОТА АГЕНТУ

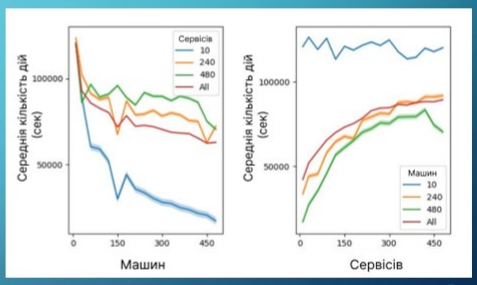




ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ КОЖНОГО СЦЕНАРІЮ, ПОРІВНЯНО З АГЕНТОМ ІЗ ПОЛІТИКОЮ ВИБОРУ ВИПАДКОВОЇ ДІЇ

ОЦІНКА ПРОДУКТИВНОСТІ СИМУЛЯТОРА МЕРЕЖІ

- Мінімальний час завантаження оточення агенту становив 0,008 сек для симулятора, перевіреному на 10 машинах і 10 сервісах, в той час, як середній час завантаження окремої віртуальної машини становив 0,249 сек;
- Для середнього значення дій за секунду в симуляторі мінімум становив 16329 ± 1907 дій (при 480 машинах і 10 сервісах)
- Для порівняння, значення часу, необхідного для сканування Nmap одного порту, становило 0,253 ± 0,03 с, що 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.



21

ВИСНОВКИ

- В ході виконання роботи були досліджені існуючі алгоритми машинного навчання, що були реалізовані у вигляді агентів для тестування безпеки;
- Також був розроблений симулятор мережі, що виступає оточенням для навчання та роботи агентів тестування
- Завдяки розробленому симулятору мережі, агенти штучного інтелекту здатні в рази швидше навчатись ніж з використанням віртуальних машин;
- Розроблені агенти здатні вчитись та виконувати оптимальні дії, для злому комп'ютерних систем.

22

ДОДАТОК Б

Код DQL агенту

```
import random
import numpy as np
from pprint import pprint

class ReplayMemory:

    def __init__(self, capacity, s_dims, device="cpu"):
        self.capacity = capacity
        self.device = device
        self.s_buf = np.zeros((capacity, *s_dims), dtype=np.float32)
        self.a_buf = np.zeros((capacity, 1), dtype=np.int64)
        self.next_s_buf = np.zeros((capacity, *s_dims), dtype=np.float32)
        self.r_buf = np.zeros(capacity, dtype=np.float32)
        self.done_buf = np.zeros(capacity, dtype=np.float32)
        self.ptr, self.size = 0, 0

    def store(self, s, a, next_s, r, done):
        self.s_buf[self.ptr] = s
        self.a_buf[self.ptr] = a
        self.next_s_buf[self.ptr] = next_s
        self.r_buf[self.ptr] = r
        self.done_buf[self.ptr] = done
        self.ptr = (self.ptr + 1) % self.capacity
        self.size = min(self.size+1, self.capacity)

    def sample_batch(self, batch_size):
        sample_idxs = np.random.choice(self.size, batch_size)
        batch = [self.s_buf[sample_idxs],
                 self.a_buf[sample_idxs],
                 self.next_s_buf[sample_idxs],
                 self.r_buf[sample_idxs],
                 self.done_buf[sample_idxs]]
        return [torch.from_numpy(buf).to(self.device) for buf in batch]


class DQN(nn.Module):
    """Мережа Q-Network """

    def __init__(self, input_dim, layers, num_actions):
        super().__init__()
        self.layers = nn.ModuleList([nn.Linear(input_dim[0], layers[0])])
        for l in range(1, len(layers)):
            self.layers.append(nn.Linear(layers[l-1], layers[l]))
        self.out = nn.Linear(layers[-1], num_actions)

    def forward(self, x):
        for layer in self.layers:
            x = F.relu(layer(x))
        x = self.out(x)
        return x

    def save_DQN(self, file_path):
        torch.save(self.state_dict(), file_path)

    def load_DQN(self, file_path):
        self.load_state_dict(torch.load(file_path))

    def get_action(self, x):
        with torch.no_grad():
```

```
            if len(x.shape) == 1:
                x = x.view(1, -1)
            return self.forward(x).max(1)[1]


class DQNAgent:
    """Агент DQN"""

    def __init__(self,
                 env,
                 seed=None,
                 lr=0.001,
                 training_steps=20000,
                 batch_size=32,
                 replay_size=10000,
                 final_epsilon=0.05,
                 exploration_steps=10000,
                 gamma=0.99,
                 hidden_sizes=[64, 64],
                 target_update_freq=1000,
                 verbose=True,
                 **kwargs):

        assert env.flat_actions
        self.verbose = verbose
        if self.verbose:
            print(f"\nRunning DQN with config:")
            pprint(locals())

        # set seeds
        self.seed = seed
        if self.seed is not None:
            np.random.seed(self.seed)

        # envirnment setup
        self.env = env

        self.num_actions = self.env.action_space.n
        self.obs_dim = self.env.observation_space.shape

        self.logger = SummaryWriter()
        self.lr = lr
        self.exploration_steps = exploration_steps
        self.final_epsilon = final_epsilon
        self.epsilon_schedule = np.linspace(1.0,
                                            self.final_epsilon,
                                            self.exploration_steps)
        self.batch_size = batch_size
        self.discount = gamma
        self.training_steps = training_steps
        self.steps_done = 0

        self.device = torch.device("cuda"
                                   if torch.cuda.is_available()
                                   else "cpu")
        self.dqn = DQN(self.obs_dim,
                       hidden_sizes,
                       self.num_actions).to(self.device)
        if self.verbose:
            print(f"\nUsing Neural Network running on device={self.device}:")
            print(self.dqn)

        self.target_dqn = DQN(self.obs_dim,
                              hidden_sizes,
```

```
                            self.num_actions).to(self.device)
        self.target_update_freq = target_update_freq

        self.optimizer = optim.Adam(self.dqn.parameters(), lr=self.lr)
        self.loss_fn = nn.SmoothL1Loss()

        self.replay = ReplayMemory(replay_size,
                                   self.obs_dim,
                                   self.device)

    def save(self, save_path):
        self.dqn.save_DQN(save_path)

    def load(self, load_path):
        self.dqn.load_DQN(load_path)

    def get_epsilon(self):
        if self.steps_done < self.exploration_steps:
            return self.epsilon_schedule[self.steps_done]
        return self.final_epsilon

    def get_egreedy_action(self, o, epsilon):
        if random.random() > epsilon:
            o = torch.from_numpy(o).float().to(self.device)
            return self.dqn.get_action(o).cpu().item()
        return random.randint(0, self.num_actions-1)

    def optimize(self):
        batch = self.replay.sample_batch(self.batch_size)
        s_batch, a_batch, next_s_batch, r_batch, d_batch = batch

        q_vals_raw = self.dqn(s_batch)
        q_vals = q_vals_raw.gather(1, a_batch).squeeze()

        with torch.no_grad():
            target_q_val_raw = self.target_dqn(next_s_batch)
            target_q_val = target_q_val_raw.max(1)[0]
            target = r_batch + self.discount*(1-d_batch)*target_q_val

        loss = self.loss_fn(q_vals, target)

        # optimize the model
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        if self.steps_done % self.target_update_freq == 0:
            self.target_dqn.load_state_dict(self.dqn.state_dict())

        q_vals_max = q_vals_raw.max(1)[0]
        mean_v = q_vals_max.mean().item()
        return loss.item(), mean_v

    def train(self):
        if self.verbose:
            print("\nStarting training")

        num_episodes = 0
        training_steps_remaining = self.training_steps

        while self.steps_done < self.training_steps:
            ep_results = self.run_train_episode(training_steps_remaining)
            ep_return, ep_steps, goal = ep_results
            num_episodes += 1
```

```
            training_steps_remaining -= ep_steps

            self.logger.add_scalar("episode", num_episodes, self.steps_done)
            self.logger.add_scalar(
                "epsilon", self.get_epsilon(), self.steps_done
            )
            self.logger.add_scalar(
                "episode_return", ep_return, self.steps_done
            )
            self.logger.add_scalar(
                "episode_steps", ep_steps, self.steps_done
            )
            self.logger.add_scalar(
                "episode_goal_reached", int(goal), self.steps_done
            )

            if num_episodes % 10 == 0 and self.verbose:
                print(f"\nEpisode {num_episodes}:")
                print(f"\tsteps done = {self.steps_done} / "
                      f"{self.training_steps}")
                print(f"\treturn = {ep_return}")
                print(f"\tgoal = {goal}")

        self.logger.close()
        if self.verbose:
            print("Training complete")
            print(f"\nEpisode {num_episodes}:")
            print(f"\tsteps done = {self.steps_done} / {self.training_steps}")
            print(f"\treturn = {ep_return}")
            print(f"\tgoal = {goal}")

    def run_train_episode(self, step_limit):
        o = self.env.reset()
        done = False

        steps = 0
        episode_return = 0

        while not done and steps < step_limit:
            a = self.get_egreedy_action(o, self.get_epsilon())

            next_o, r, done, _ = self.env.step(a)
            self.replay.store(o, a, next_o, r, done)
            self.steps_done += 1
            loss, mean_v = self.optimize()
            self.logger.add_scalar("loss", loss, self.steps_done)
            self.logger.add_scalar("mean_v", mean_v, self.steps_done)

            o = next_o
            episode_return += r
            steps += 1

        return episode_return, steps, self.env.goal_reached()
```

Відомість кваліфікаційної роботи

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК. 46741Х.001 ПЗ	Пояснювальна записка	83 с.
		Графічні документи	
2		Слайд-презентація	22 слайдів
		Носії інформації	
3		Файл пояснювальної записки	1 шт.

					ГЮІК. 46741Х.001 ВД	
Змін.	Арк	Номер докум.	Підп.	Дата		
Розроб.		Сімакін В.А.			Тестування безпеки комп'ютерних систем з використанням навчання з підкріпленням Відомість кваліфікаційної роботи	Літ: У; Аркуш; Аркушів
Перевір.		Шкіль О.С.				
Н.контр.		Шкіль О.С.				ХНУРЕ Кафедра АПОТ
Затв.		Чумаченко С.В.				