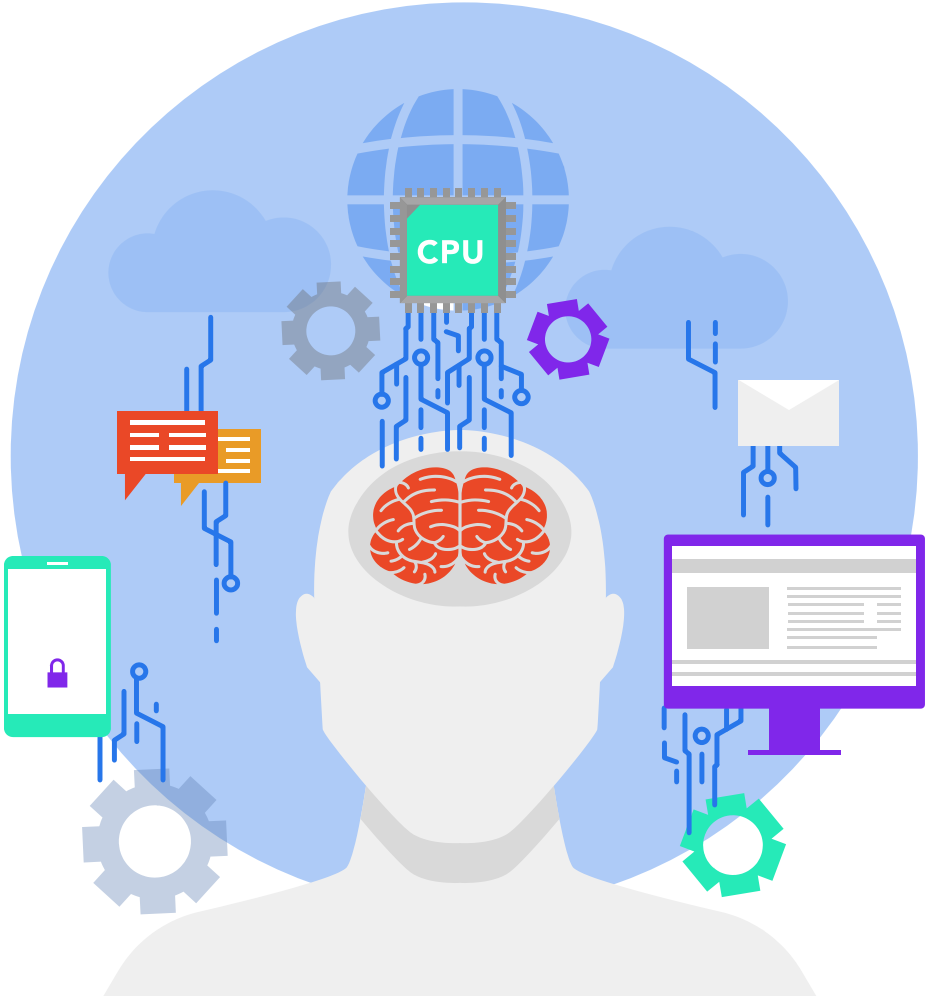
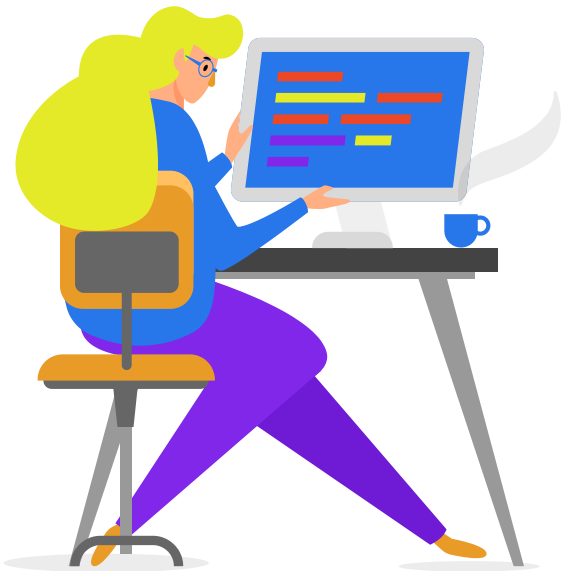


Machine Learning in Pentesting

NT522.O21.ANTT

GVHD: ThS Phan Thế Duy

Machine Learning in Pentesting



01 Tổng quan

02 Phương pháp

03 Triển khai

04 Đánh giá

05 Kết luận

Tổng quan

Bài báo nghiên cứu chính

- Massimo Zennaro, F., & Erdodi, L. (2020). Modeling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges: Trade-offs between Model-free Learning and A Priori Knowledge. arXiv e-prints, arXiv-2005.

Tổng quan

Thực trạng

- Nhu cầu pentest lớn
- Pentest đa phần được thực hiện thủ công, tốn thời gian
- Yêu cầu các chuyên gia, có hiểu biết và kiến thức

Vs

Thách thức

- Tự động hóa Pentest gặp nhiều khó khăn: kiến thức, phương pháp, state & action space lớn và rời rạc
- Các ứng dụng RL vào pentest chỉ mới cho thấy hiệu quả trên một số trường hợp lý tưởng trong môi trường ảo

Tổng quan

Đề xuất

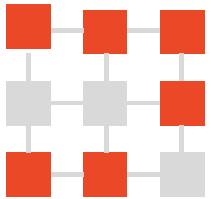
- Áp dụng Reinforcement learning vào Pentest.
- Vì khó khăn bởi khối lượng lớn kiến thức; không gian trạng thái và hành động lớn và rời rạc; kiến trúc hệ thống cần pentest phức tạp, nhiều ẩn số. Do đó, nhóm tác giả chỉ xét tới những ngữ cảnh đơn giản.
- Các kịch bản sử dụng RL để giải quyết các vấn đề CTF gồm nhiều cơ chế bảo vệ, ngăn chặn và chống lại hành vi pentest của RL agent.

Phương pháp

Reinforcement learning

Q-learning

Modeling PT as RL problem



Penetration Testing

CTF

Phương pháp

Modeling PT as a RL problem

- Penetration Testing (PT): Bài kiểm tra bảo mật nhằm đánh giá mức độ an toàn của hệ thống thông qua việc mô phỏng các cuộc tấn công mạng.
- Reinforcement Learning (RL): Phương pháp học máy trong đó một agent học cách tối ưu hóa hành vi của mình trong một môi trường nhất định bằng cách nhận phản hồi từ các hành động đã thực hiện.

Q-learning	PT	RL
State (S)	Trạng thái của hệ thống đang được kiểm tra.	Trạng thái của môi trường tại một thời điểm cụ thể.
Action (A)	Các hành động như tấn công một dịch vụ, thu thập thông tin hệ thống.	Các hành động mà agent có thể thực hiện để thay đổi trạng thái của môi trường.
Transition Function (T)	Xác suất chuyển đổi trạng thái hệ thống khi thực hiện một hành động.	Xác suất chuyển đổi giữa các trạng thái trong môi trường.
Reward Function (R)	Phần thưởng khi phát hiện lỗ hổng bảo mật.	Phần thưởng nhận được khi thực hiện một hành động trong một trạng thái cụ thể.

Phương pháp

Modeling PT as a RL problem

PT vs Game khi áp dụng RL

Điểm giống:

- Đều dễ dàng xác định được người chơi, luật chơi và điều kiện để chiến thắng.
- PT: information gathering + exploitation
  Game: exploration actions + exploitation actions

Điểm đặc trưng khác biệt của PT:

- PT có không gian (state, action) rất lớn, kiến trúc môi trường bị ẩn và không ổn định so với các bài toán RL thông thường.
- Các chuyên gia trong thực tế dựa vào nhiều nguồn để đưa ra quyết định, RL agent chỉ học dựa trên trial-and-error

Phương pháp

Modeling PT as a RL problem

- Agent: Đóng vai trò là pentester trong PT, học cách tấn công hệ thống thông qua các hành động và phản hồi.
- Môi trường: Hệ thống server mà agent tương tác.

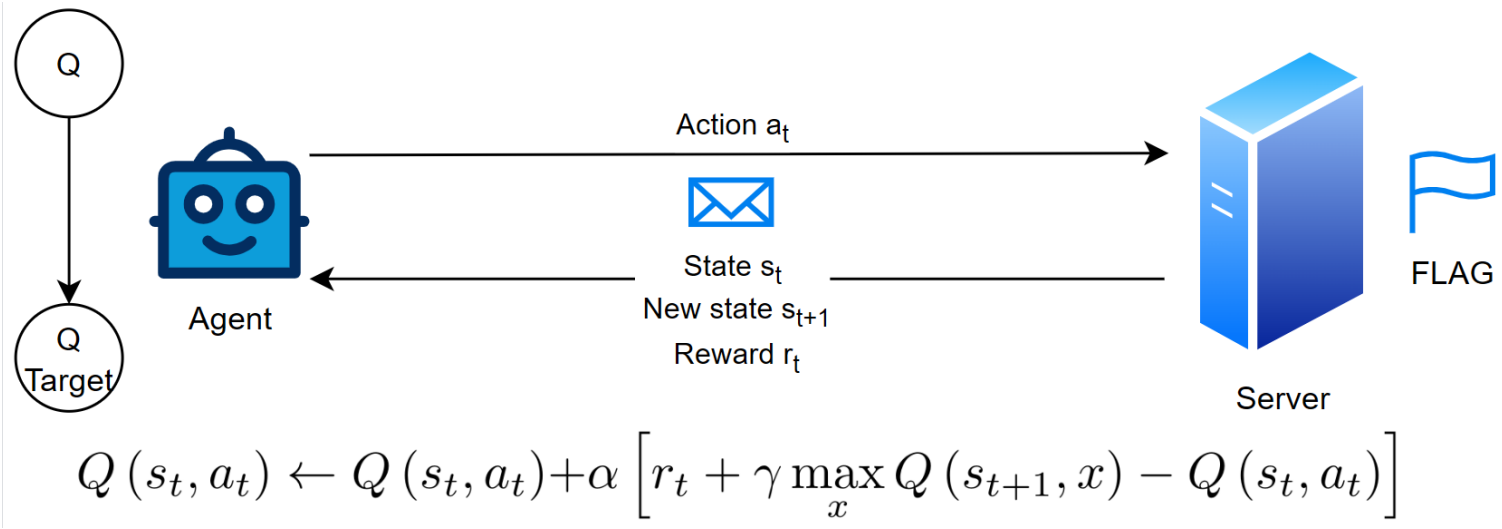


Fig: Our formalism in modeling CTF challenges as RL problems

Triển khai

Types of CTF problems

1. Port Scanning and Intrusion:

- Mô tả: Máy chủ mở một số cổng trên mạng. Attacker cần kiểm tra các cổng này, xác định cổng nào có lỗ hổng và dùng một phương thức khai thác đã biết để lấy cờ (flag) từ cổng có lỗ hổng.
- Quá trình: Quét cổng, xác định lỗ hổng, khai thác lỗ hổng.

2. Server Hacking:

- Mô tả: Máy chủ cung cấp một số dịch vụ trên mạng. Attacker cần tương tác với các dịch vụ này, phát hiện lỗ hổng (có thể là lỗ hổng không tham số hoặc có tham số) và lấy cờ bằng cách khai thác lỗ hổng đã phát hiện.
- Quá trình: Quét cổng, tương tác với dịch vụ, phát hiện lỗ hổng, khai thác lỗ hổng.

3. Website Hacking:

- Mô tả: Máy chủ mục tiêu cung cấp một trang web. Attacker cần kiểm tra các trang có sẵn, đánh giá xem trang nào có lỗ hổng và lấy cờ từ một trong các trang bằng cách khai thác lỗ hổng đã phát hiện.
- Quá trình: Kiểm tra trang web, phát hiện lỗ hổng, khai thác lỗ hổng.

Triển khai

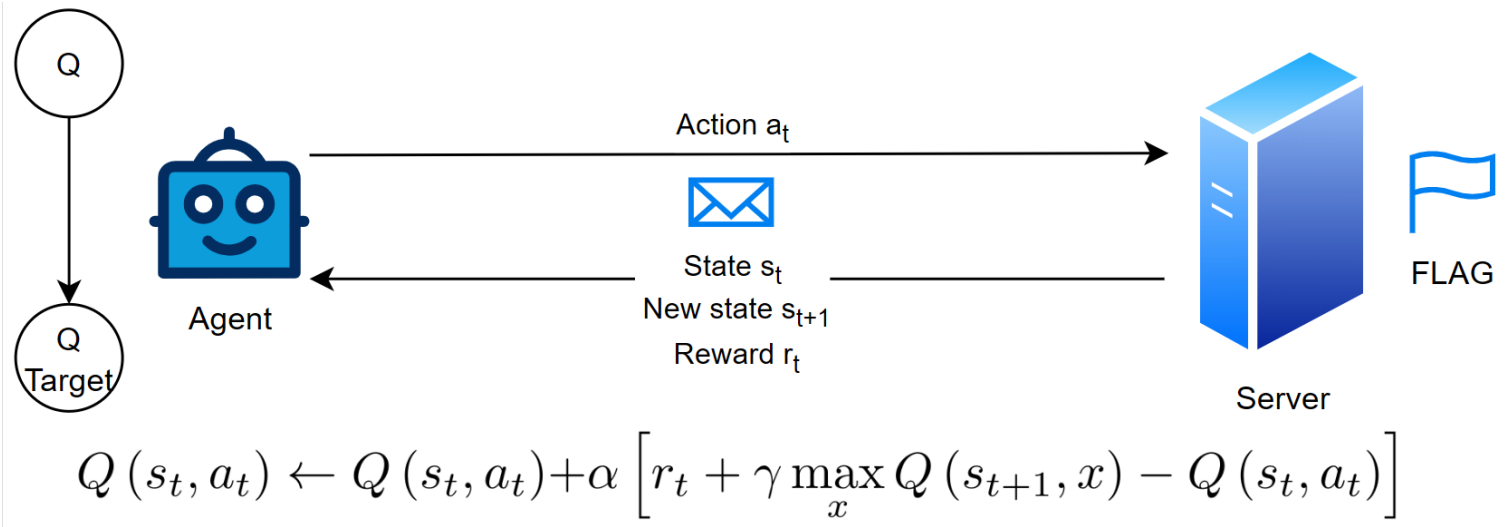


Fig: Our formalism in modeling CTF challenges as RL problems

Triển khai

Pentest Simulation

Port scanning 01

Non-stationary Port scanning 02

Server hacking With lazy loading 03

Website hacking with State Aggregation 04

Server hacking with Imitation Learning 05

Website hacking with Imitation Learning 06

Kỹ thuật:

- Lazy Loading: chỉ lưu trạng thái đã gặp.
- State Aggregation: Tập hợp các trạng thái tương tự.
- Imitation Learning: Học từ ví dụ có sẵn

Triển khai

Simulation 1: Port Scanning CTF Problem

- **Ngữ cảnh**:
  Server chạy N port, trong đó chỉ có 1 vulport chứa FLAG.
  Attacker (agent) tương tác với server:
    + Scan: phát hiện vulport (không cần tham số đầu vào).
    + Attack: tham số truyền vào là target port, trả về kết quả cuộc tấn công.
  Agent chưa biết gì về Server, học dựa trên bảng Q-learning

- **RL setup**:
  A: N+1 action (1 scan, N exploit cho N port)
  S: N+1 biến nhị phân (1 scan, N exploit cho N port)
  reward: 100 khi lấy được FLAG, -1 cho các trường hợp còn lại

Triển khai

Simulation 1: Port Scanning CTF Problem

- **Kết quả**: N=64 port, chạy 1000 episodes, thống kê từ trung bình 100 lần chạy

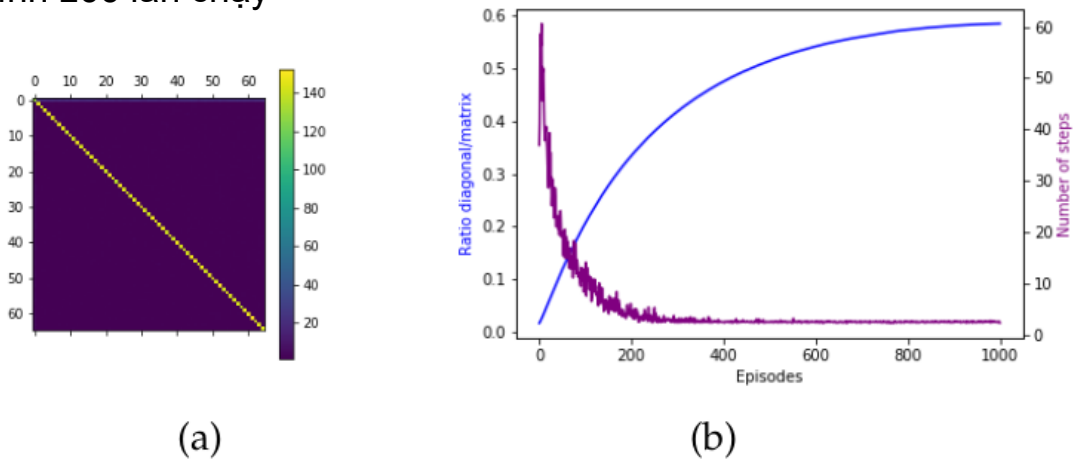


Fig. 1: Results of Simulation 1. (a) Learned action-value matrix Q. (b) Plot of ratio $\frac{\sum_{i=0}^{N} Q_{ii}}{\sum_{i,j=0}^{N} Q_{ij}}$ as a function of the number of episodes (in blue), and number of steps as a function of the number of episodes (in purple).

Triển khai

Simulation 2: Non-stationary Port-scanning CTF Problem

- **Ngữ cảnh**: nâng cao của Simulation 1, mô phỏng môi trường **không ổn định** (*non-stationary* environment), trong đó server có xác suất p phát hiện hành động Scan của Agent, nếu hành động Scan bị phát hiện, FLAG sẽ chuyển sang một port bất kỳ khác.
- **RL setup**:
  - A: N+1 action (1 scan, N exploit cho N port)
  - S: N+1 biến nhị phân (1 scan, N exploit cho N port)
  - reward: 100 khi lấy được FLAG, -1 cho các trường hợp còn lại
  - p: xác suất phát hiện tấn công
- **Kết quả**: N=16 port, chạy 1000 episodes, thống kê từ trung bình 100 lần chạy

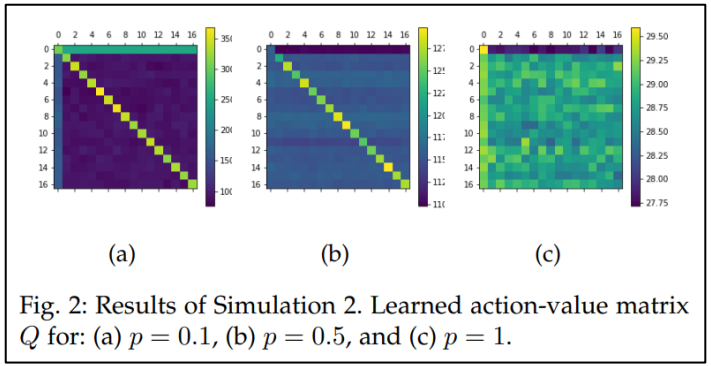


Fig. 2: Results of Simulation 2. Learned action-value matrix Q for: (a) $p = 0.1$, (b) $p = 0.5$, and (c) $p = 1$.

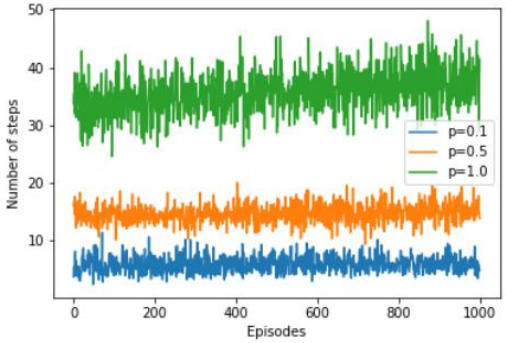


Fig. 3: Results of Simulation 2. Number of steps as a function of episodes for $p = 0.1$, $p = 0.5$, $p = 1$.

Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- **Ngữ cảnh**:
  Server chạy N port, các port đóng hoặc mở
  có 1 vulport chứa FLAG, vulport có 1 trong 2 loại lỗ hổng:
  + chạy service lắng nghe có lỗ hổng -> chỉ cần access là có được FLAG.
  + lổ hổng tham số -> trả về FLAG khi truyền tham số đặc biệt

  Attacker (agent) có thể thực hiện các hành động:
  + Scan: phát hiện port mở
  + Read: 1 port cụ thể để phát hiện lổ hổng ở service hoặc lổ hổng tham số (nếu hiển thị rõ ràng)
  + DeepRead: 1 port cụ thể để phát hiện lổ hổng tham số (hidden)
  + Access: 1 port cụ thể để truy cập vào service.
  + Send: 1 port cụ thể để khai thác lổ hổng tham số.

- **RL setup**:
  N port, V service, M param
  A: 1 + N + N + (N * V) + (N * M) action
  S: ma trận N * (1 + V + 1)
  R: 100 cho FLAG, -1 cho những action khác

Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- Ban đầu:

```
class Agent():
    def __init__(self, nports, verbose=True):
        self.nports = nports  # Số lượng cổng có thể có
        self.Q = np.ones((nports+1, nports+1))  # Khởi tạo bảng Q-learning
        self.verbose = verbose  # Hiển thị thông tin chi tiết nếu đặt là True
```

- Áp dụng **Lazy loading**

```
class Agent():
    def __init__(self,verbose=True):
        self.Q = {}
        self.nactions = const.N_ACTIONS
        self.verbose = verbose
```

Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- Ban đầu:

```
self.Q[oldstate, action] = self.Q[oldstate, action] + self.alpha * (
    reward + self.gamma * self.Q[newstate, best_action_newstate] - self.Q[oldstate, action]
)
```

- Áp dụng **Lazy loading**

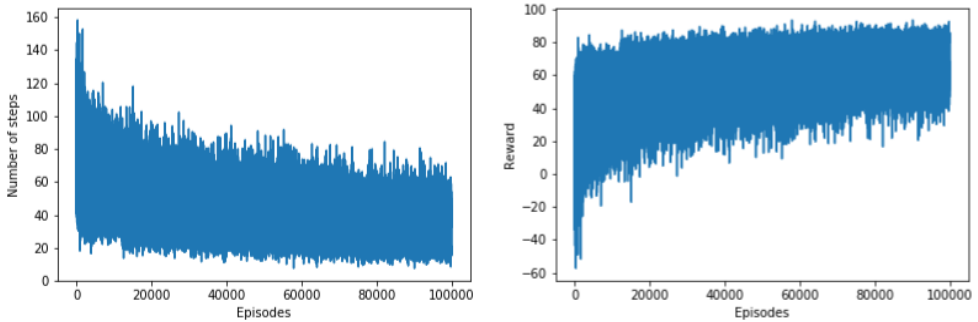
```
self.Q[oldstate.tobytes()][action] = self.Q[oldstate.tobytes()][action] + self.alpha * (
    reward + self.gamma*self.Q[newstate.tobytes()][best_action_newstate] -
self.Q[oldstate.tobytes()][action]
)
```

Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- **Kết quả**:

  N = 4 port
  V = 5 service
  M = 4 param
  chạy 10^6 episodes (e=0.3)



(a) (b)

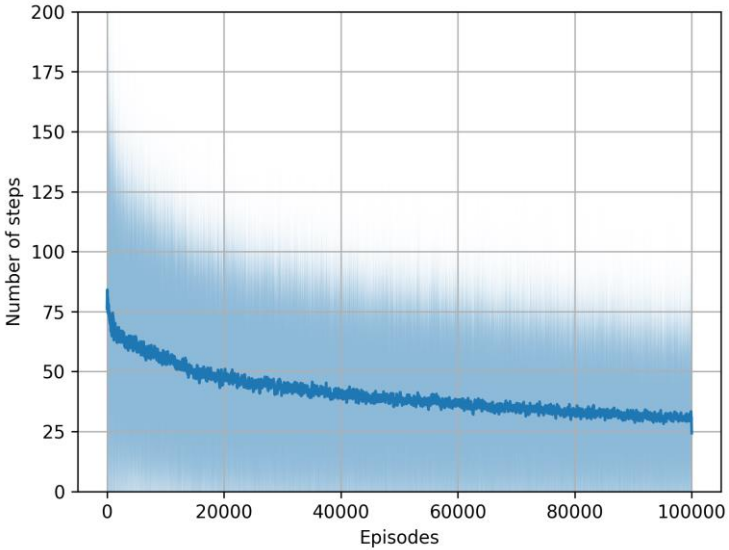
Fig. 4: Results of Simulation 3. (a) Plot of number of steps as a function of the number of episodes; (b) plot of reward as a function of the number of episodes.

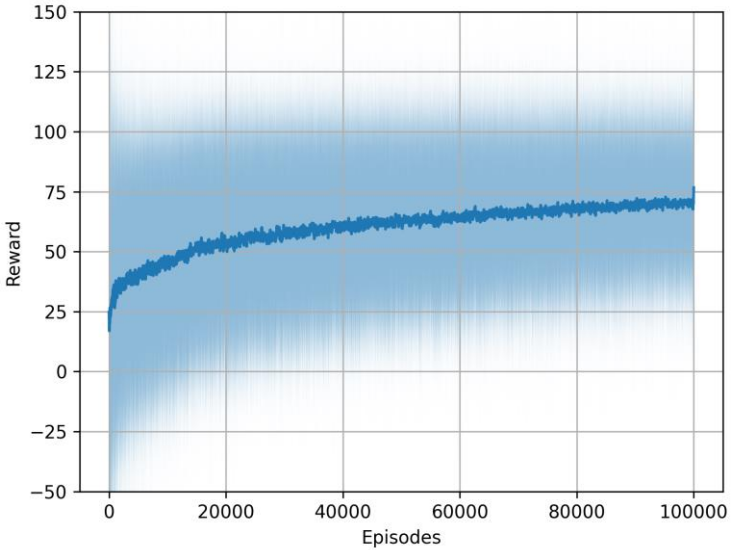
Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- **Kết quả**:

  N = 4 port
  V = 5 service
  M = 4 param
  chạy 10^6 episodes, thống kê kết quả từ trung bình 20 lần chạy (e=0.3)





Triển khai

Simulation 3: Server Hacking CTF Problem with Lazy Loading

- **Nhận xét**:
  Công thức ước tính tổng số state

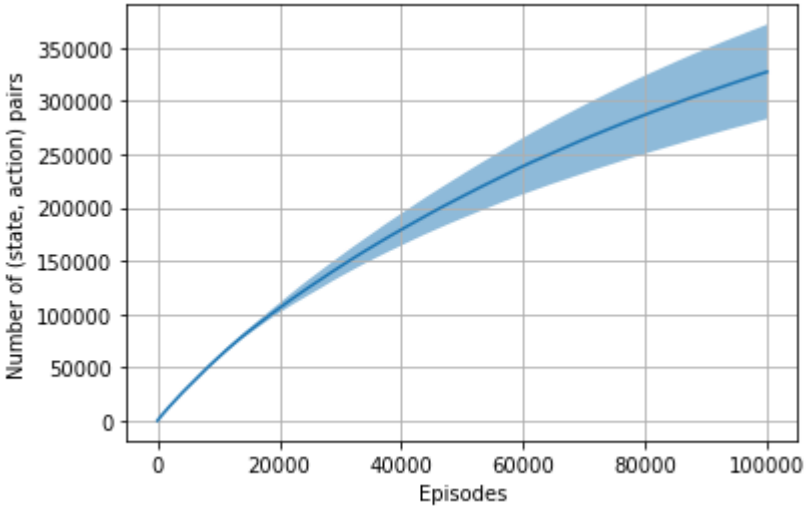
$$|\mathcal{S}| \approx 2^{11} N^3 V M. \qquad (2)$$

Theo công thức trên, với N=4, V=5, M=4

$$|S| \approx 2.6 * 10^6$$

số phần tử trong Q-table là:
350,000 < $2.6 * 10^6$

-> agent học nhanh, giảm mức tiêu thụ bộ nhớ



Number of entries in the action-value table Q

Triển khai

Simulation 4: Website Hacking CTF Problem with State Aggregation

- **Ngữ cảnh**:
  Server hosts N_{vis} tệp hiển thị có thể truy cập qua Internet, và N_{his} tệp ẩn không được truy cập qua Internet, nhưng được liên kết tới các tệp hiển thị. Chỉ 1 file bị lổ hỗng tham số có thể khai thác.

  Attacker có thể tương tác với server như sau:
  + Scan: truy xuất đồ thị liên kết các tệp hiển thị
  + Explore: 1 file cụ thể để tìm file ẩn liên kết với file đó.
  + Inspect: 1 file cụ thể để tìm lổ hỗng tham số (nếu có)
  + Send: 1 file cụ thể để gửi payload khai thác lổ hổng
  + Focus Next: di chuyển sang file tiếp theo
  + Focus Prev: lùi về file trước

- **RL setup**:
  N tệp: N_{vis} tệp hiển thị và N_{his} tệp ẩn.
  M vul parameter.
  A: 6 action của agent
  S: mỗi state được thể hiện bằng một vector
  R: 100 cho FLAG, -1 cho những action khác

Triển khai

Simulation 4: Website Hacking CTF Problem with State Aggregation

Áp dụng State Aggregation:

Thay vì yêu cầu agent học một chiến lược cụ thể trên mỗi tệp, ta hướng dẫn agent học một policy duy nhất sẽ được sử dụng trên tất cả các tệp

Tại mỗi thời điểm, agent sẽ chỉ tập trung vào một tệp duy nhất, tương tác với tệp đó và cập nhật policy toàn cục hợp lệ cho bất kỳ tệp nào.

Vd: **explore file 1, explore file 2,...**
inspect file 1, inspect file 2,...
send file 1, send file 2,...

explore
inspect
send file
focus next
focus prev

focusfile = 1
[2,3,...]

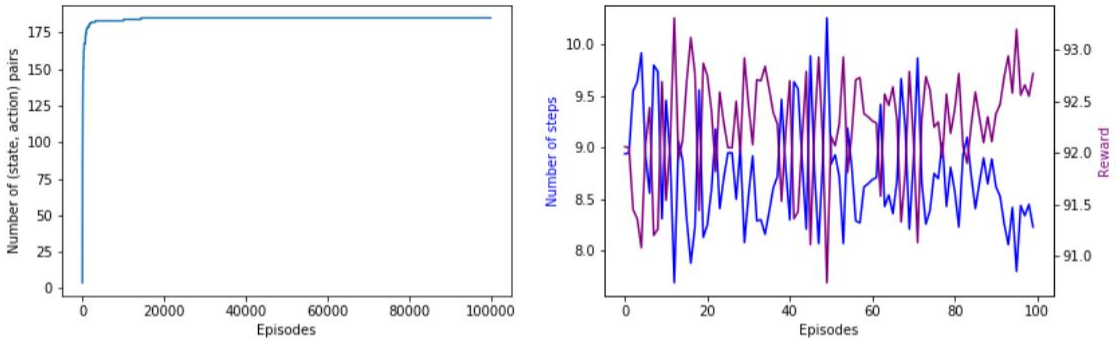
Triển khai

Simulation 4: Website Hacking CTF Problem with State Aggregation

- **Kết quả**:
  + $(2 \leq N_{vis} \leq 4)$ và $(0 \leq N_{his} \leq 2)$.
  + M = 4 vul param.

  + Kết quả chạy kiểm tra 100 tập với exploration e = 0, thống kê từ trung bình 100 lần chạy

- **Nhận xét**: Số lượng phần tử trong Q table đã giảm rất nhiều ~180 trong khi reward, và số step đạt được của agent khá hợp lý



(a) (b)

Fig. 6: Results of Simulation 4. (a) Number of entries in the action-value matrix Q as a function of the number of episodes; (b) reward and number of steps as a function of the number of episodes.

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

- **Ngữ cảnh**: tương tự Simulation 3
  Server chạy N port, các port đóng hoặc mở
  có 1 vulport chứa FLAG, vulport có 1 trong 2 loại lỗ hổng:
  + chạy service lắng nghe có lỗ hổng -> chỉ cần access là có được FLAG.
  + lổ hổng tham số -> trả về FLAG khi truyền tham số đặc biệt

  Attacker (agent) có thể thực hiện các hành động:
  + Scan: phát hiện port mở
  + Read: 1 port cụ thể để phát hiện lổ hổng ở service hoặc lổ hổng tham số (nếu hiển thị rõ ràng)
  + DeepRead: 1 port cụ thể để phát hiện lổ hổng tham số (hidden)
  + Access: 1 port cụ thể để truy cập vào service.
  + Send: 1 port cụ thể để khai thác lổ hổng tham số.

- **RL setup**:
  N port, V service, M param
  A: 1 + N + N + (N * V) + (N * M) action
  S: ma trận N * (1 + V + 1)
  R: 100 cho FLAG, -1 cho những action khác

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

Áp dụng Imitation Learning:

Cung cấp cho agent các mẫu về chuỗi action hiệu quả để lấy được FLAG -> thay vì bắt đầu với kiến thức trống rỗng, agent được cung cấp các ví dụ về chuỗi hành động có thể dẫn đến giải pháp.

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

Áp dụng Imitation Learning:

Hàm generate chuỗi action tối ưu để lấy được FLAG

```python
def generate_trajectory(server):
    trajectory = []

    # Perform a scan
    trajectory.append(0)
    # Perform a read
    trajectory.append(server.vulnport+1)
    # Perform a deep read
    trajectory.append(server.vulnport + const.N_PORTS +1)

    # Exploit the access point or the parametrized vulnerability
    if(server.vulntype == const.VULNTYPE_ACCESSPOINT):
        vulnport = server.vulnport
        vulnvalue = server.vulnvalue[0]
        #trajectory.append( (const.N_PORTS*2)+1 + vulnport + vulnvalue*const.N_PORTS )
        trajectory.append( (const.N_PORTS*2)+1 + (vulnport*const.N_ACCESSPOINTS) + vulnvalue)
    elif(server.vulntype == const.VULNTYPE_PARAM):
        vulnport = server.vulnport
        vulnvalue = server.vulnvalue
        trajectory.append( const.N_PORTS*(2+const.N_ACCESSPOINTS)+1 + (vulnport*const.N_VULNPARAMS) + (vulnvalue-1) )

    return trajectory
```

```
generate_trajectory(random_server())
[6]  ✓ 0.0s
...  [0, 1, 5, 12]
```

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

Áp dụng Imitation Learning:

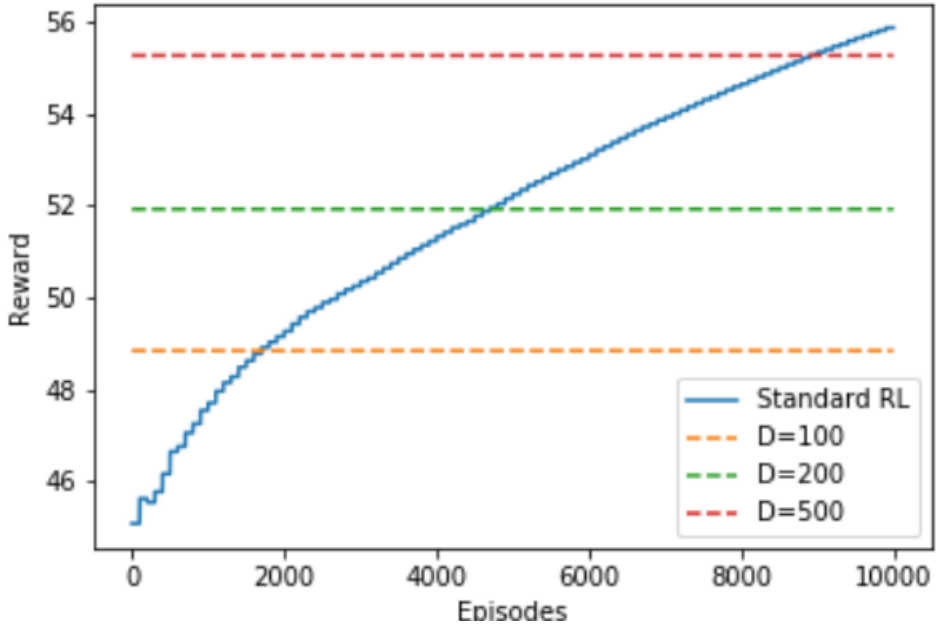
Phương thức cho phép agent học theo các chuỗi action được định nghĩa trước:

```
def run_trajectory(self,trajectory):
    _,_,self.terminated,s = self.env.reset()
    if(self.verbose): print('\nHACKER: Resetting...')
    if(self.verbose): print(s)

    for t in range(len(trajectory)):
        self.steps = self.steps+1
        action = trajectory[t]

        msg = self._package_action_into_msg(action)
        response,reward,termination,s = self.env.step(msg)
        self.rewards = self.rewards + reward

        self._analyze_response(action,msg,response,reward)
        self.terminated = termination
        if(self.verbose): print(s)

    return
```

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

- **Kết quả**:
  Huấn luyện một standard RL với 10^5 episode
  Huấn luyện 3 imitation learning với lần lượt 100, 200 và 500 demonstrations (D trajectories) sau đó chạy thêm 100 episodes để quan sát

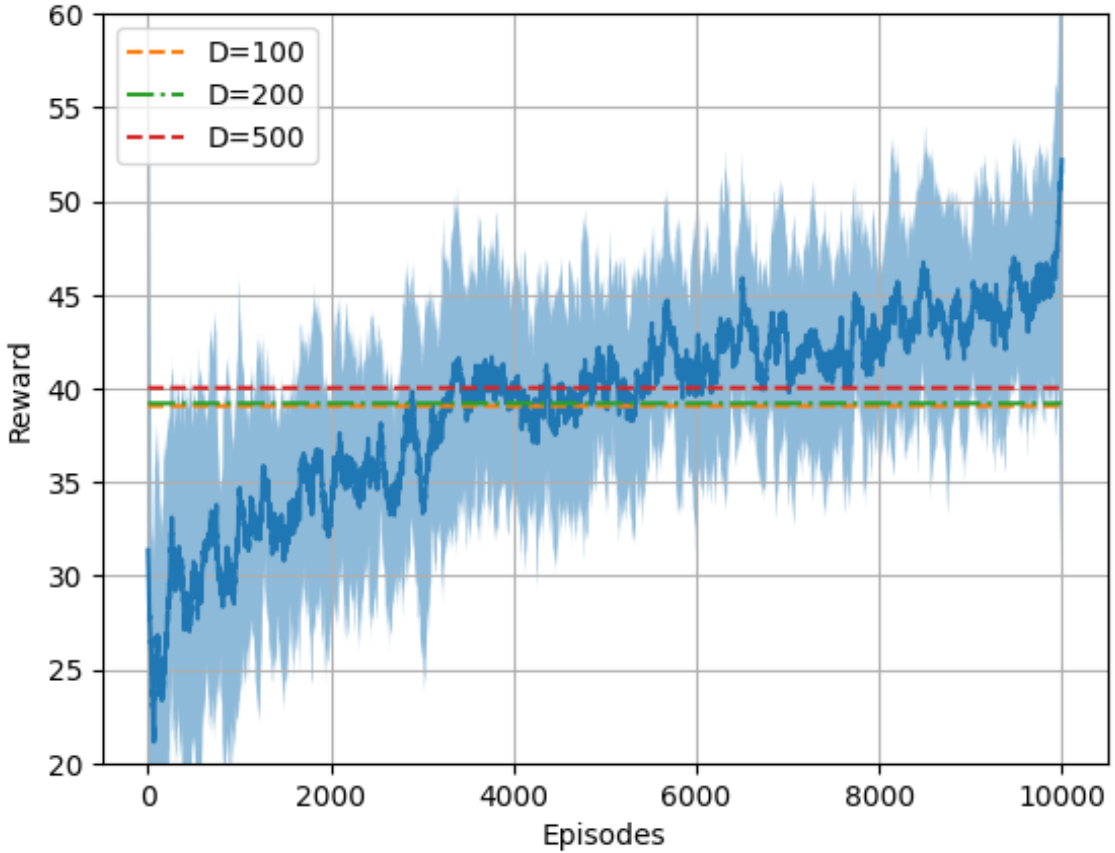


- **Nhận xét**:
  100 trajectories ~ 1800 episodes
  200 trajectories ~ 4900 episodes
  500 trajectories ~ 9000 episodes

Triển khai

Simulation 5: Server Hacking CTF Problem with Imitation Learning

- **Kết quả**:
  Lấy thống kê trung bình 20 lần chạy



- **Nhận xét**:
  cả 3 model imitation learning ~ 4000 episodes

Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning

- **Ngữ cảnh**: tương tự Simulation 4

  Server hosts N_{vis} tệp hiển thị có thể truy cập qua Internet, và N_{his} tệp ẩn không được truy cập qua Internet, nhưng được liên kết tới các tệp hiển thị. Chỉ 1 file bị lổ hỗng tham số có thể khai thác.

  Attacker có thể tương tác với server như sau:
  + Scan: truy xuất đồ thị liên kết các tệp hiển thị
  + Explore: 1 file cụ thể để tìm file ẩn liên kết với file đó.
  + Inspect: 1 file cụ thể để tìm lổ hỗng tham số (nếu có)
  + Send: 1 file cụ thể để gửi payload khai thác lổ hổng
  + Focus Next: di chuyển sang file tiếp theo
  + Focus Prev: lùi về file trước

- **RL setup**:

  N tệp: N_{vis} tệp hiển thị và N_{his} tệp ẩn.
  M vul parameter.
  A: 6 action của agent
  S: mỗi state được thể hiện bằng một vector 8 số nguyên
  R: 100 cho FLAG, -1 cho những action khác

Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning

Áp dụng Imitation Learning:

Hàm generate chuỗi action tối ưu để lấy được FLAG

```
def generate_trajectory(server):
    trajectory = []

    trajectory.append(const.ACTION_SCAN)

    for index in range(len(server.filesystem.nodes)):
        trajectory.append(const.ACTION_INSPECT)
        if index != server.vulnfile:
            trajectory.append(const.ACTION_EXPLORE)
            trajectory.append(const.ACTION_FOCUS_NEXT)
        else:
            trajectory.append(const.ACTION_SEND_PARAM1 -1 +server.vulnparam)
            return trajectory
```

```
generate_trajectory(random_server())
[5]  ✓ 0.0s
...  [0, 2, 1, 7, 2, 6]
```

Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning

Áp dụng Imitation Learning:

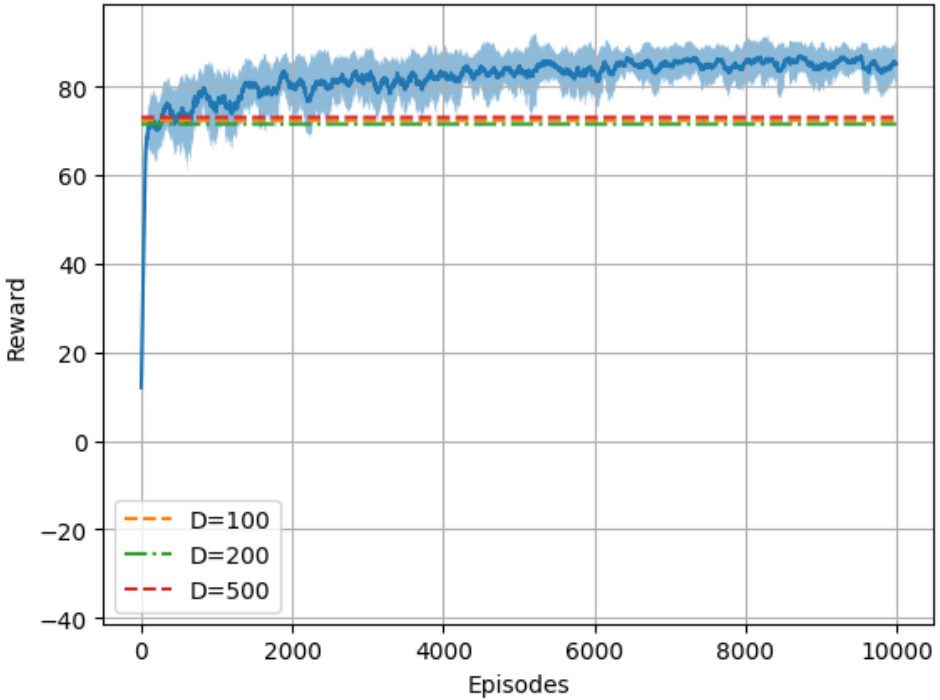
Phương thức cho phép agent học theo các chuỗi action được định nghĩa trước:

```
def run_trajectory(self,trajectory):
    _,_,self.terminated,s = self.env.reset()
    if(self.verbose): print('\nHACKER: Resetting...')
    if(self.verbose): print(s)

    for t in range(len(trajectory)):
        self.steps = self.steps+1
        action = trajectory[t]

        msg = self._package_action_into_msg(action)
        response,reward,termination,s = self.env.step(msg)
        self.rewards = self.rewards + reward

        self._analyze_response(action,msg,response,reward)
        self.terminated = termination
        if(self.verbose): print(s)

    return
```

Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning

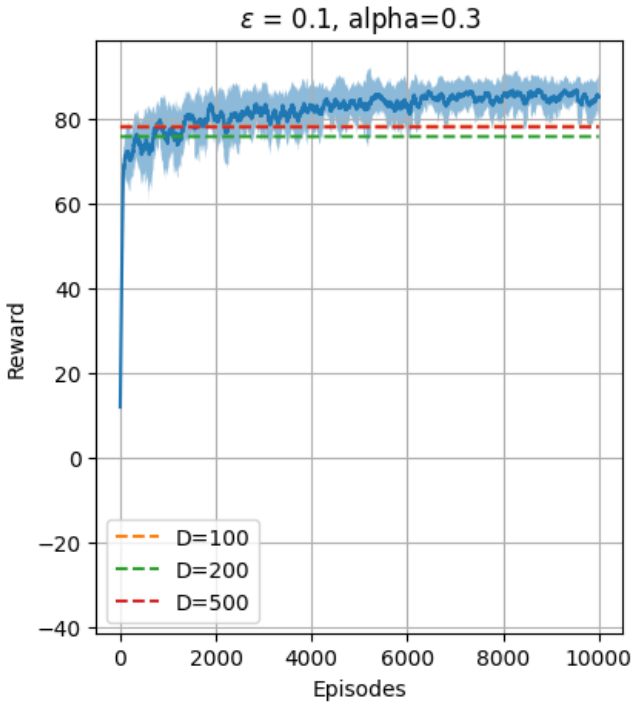
- **Kết quả**:
  Huấn luyện một standard RL agent với 10^5 episode
  Huấn luyện 3 imitation learning với lần lượt 100, 200 và 500 demonstrations (D trajectories) sau đó chạy thêm 100 episodes để quan sát
  Thống kê kết quả trung bình của 20 lần chạy.

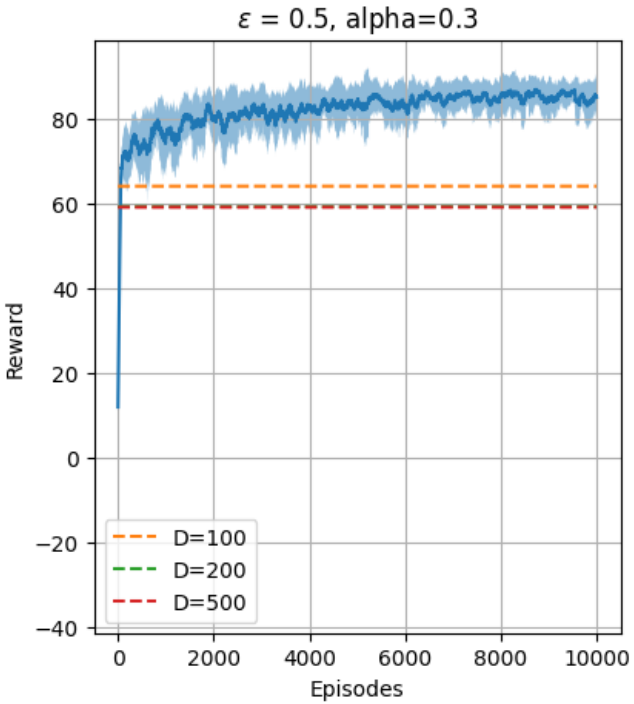


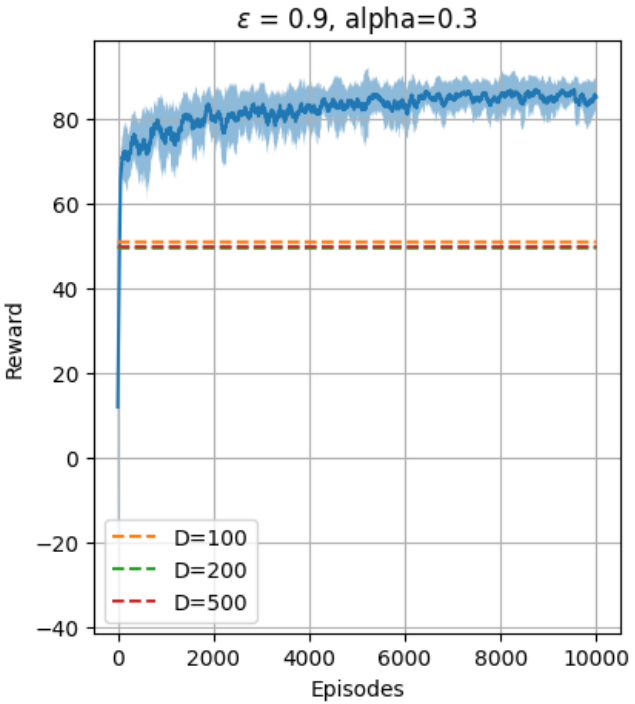
- **Nhận xét**: imitation learning mặc dù có hiệu quả nhưng mau chóng bị standard RL agent vượt qua

Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning

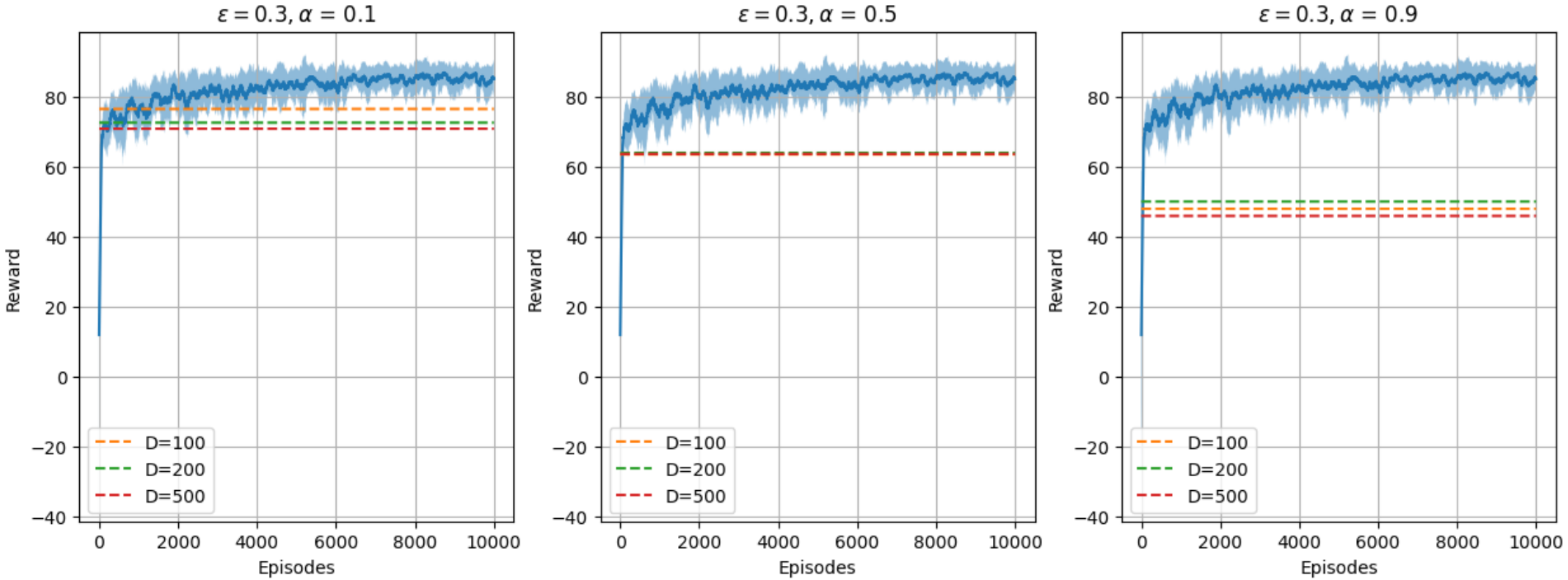






Triển khai

Simulation 6: Website Hacking CTF Problem with Imitation Learning



Đánh giá

Mô phỏng cho thấy

- Sử dụng RL để giải các bài toán CTF là khả thi,
- Vai trò và những thách thức trong việc khám phá cấu trúc bài toán PT và cung cấp kiến thức cho agent.

Đã xem xét hai cách mà cấu trúc của vấn đề có thể thay đổi và đặt ra những thách thức cụ thể:

1. Cấu trúc hệ thống ngày càng không xác định được, chuyển từ hệ thống CTF cố định sang hệ thống max-entropy

-> Giới hạn của việc học bằng suy luận

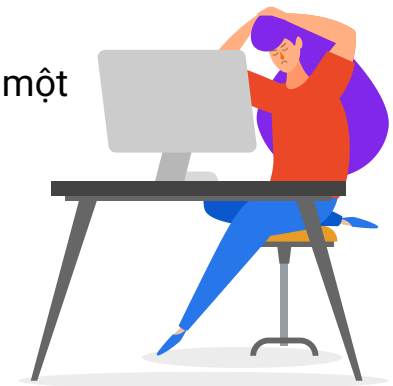
Đánh giá

2. Một bài toán CTF cố định nhưng cấu trúc ngày càng phức tạp hơn

➔ Giải pháp: cung cấp kiến thức ban đầu cho agent

- **Lazy loading** với giả định rằng một số cấu hình nhất định trong không gian vấn đề sẽ không bao giờ gặp phải và do đó có thể bị bỏ qua;
- **State aggregation** giả định rằng các cấu hình nhất định sẽ giống hệt về mặt thực tế với các cấu hình khác
- **Imitation learning** giả định rằng một giải pháp tối ưu sẽ không quá xa so với những minh chứng nổi tiếng---well-known demonstrations

Tất cả các dạng kiến thức ban đầu này không bị ràng buộc về mặt ngữ nghĩa với một vấn đề cụ thể và chúng có thể triển khai trên nhiều vấn đề CTF khác



Kết luận

Khám phá cấu trúc là một bước thiết yếu trong CTF -> Agent học lý tưởng cân bằng hợp lý giữa việc khám phá cấu trúc và dựa vào kiến thức tiên nghiệm.

Cân nhắc về độ phức tạp và những hạn chế thực tế cho thấy rằng việc đưa ra kiến thức tiên nghiệm có thể là điều cần thiết.

- Agent hoàn toàn dựa trên mô hình không lý tưởng vì khó mã hóa và không linh hoạt lắm
- Agent kết hợp thuật toán không có mô hình với kiến thức tiên nghiệm có thể đạt được sự cân bằng lý tưởng để làm cho chúng hiệu quả và hữu ích.