Applying Reinforcement Learning and Genetic Algorithms in Game-Theoretic Cyber-Security

Learning to Penetrate Networks

Ștefan P. Niculae

AFFILIATION

Experimental Research Unit, Bitdefender

Master Thesis Supervisor: M. Popescu, FMI UB

Research Project Supervisors: T. Bäck & K. Yang, LIACS

CONTENTS

1 INTRODUCTION

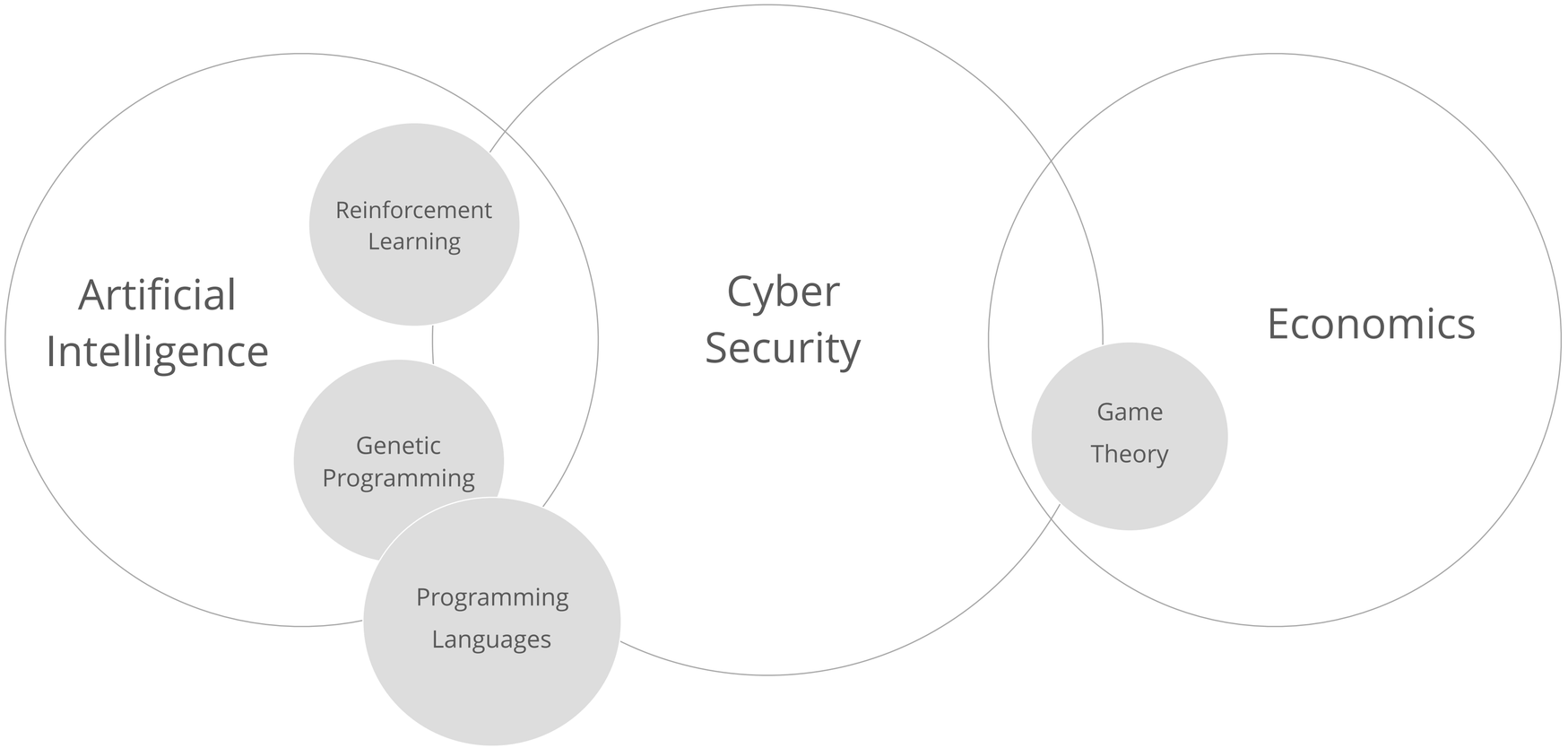
MOTIVATION

- Security solutions (especially AI) need to be validated
  - frequently, on demand
  - repeatably, in large amounts
- *Penetration testing (pentesting)* ≔ attack your own system in order to reveal its security flaws
- Imperfect solution 1 — manual pentesting:
  - slow, constrained by human interaction
  - runs not guaranteed to be identical, not always auditable
- Imperfect solution 2 — automated pentesting:
  - next to no frameworks exist
  - existing ones offer severely ineffective attacks
- **Solution**: automate pentesting using a learning-based approach

OBJECTIVE

- Given a network of machines
- Find its weaknesses
  - faster than randomly trying available exploits
  - comparable in strength to a real attacker
- Approach: model pentesting as a game and learn strategies to win it
- Ultimate goal: allow efficient evaluation (an essential development step) of robust security solutions

DOMAIN INTERSECTION

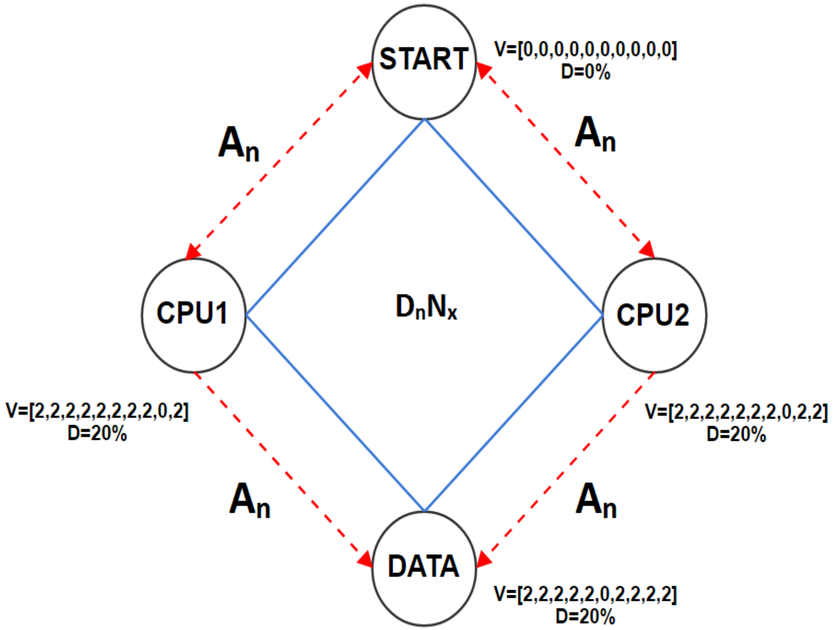


2

PRIOR APPROACHES

THE *SIMPLE* APPROACH

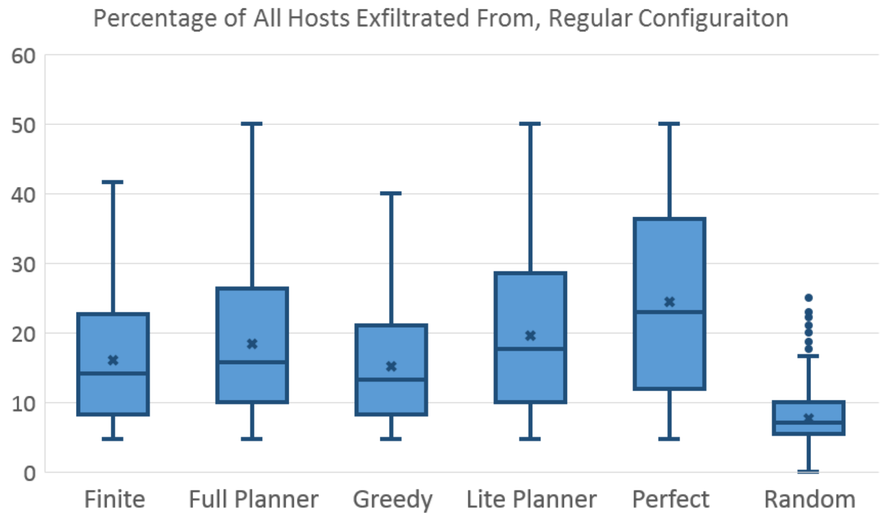
- Only 4 nodes, of which 1 is start and one is target
- Each node has a tuple of integers
- Attacker & defender can increment one value at a time
- An attack on a node is successful if $attk_{val} > def_{val}$
- Agents have no knowledge of the other's allocation



source: [5]

THE *MITRE* APPROACH

- Reconnaissance, exploit and cautionary actions
- Probabilistic attacker action success
- Include neutral user interaction
- Dynamic machine connections
- Addition/patching of vulnerabilities
- Both fixed-strategy and adaptive algorithms

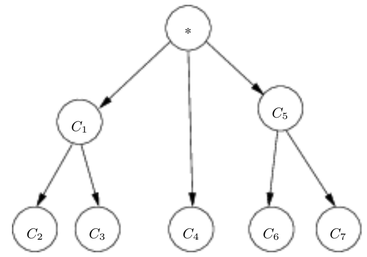


source: [6]

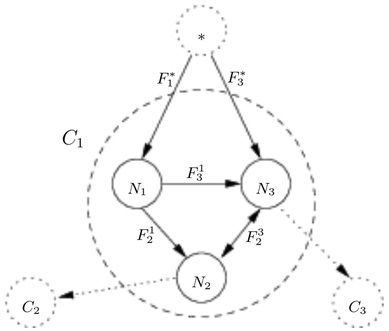
THE *CORE* APPROACH

- Asymmetrical machine connections
- Target larger topologies, machine clusters
- Abstraction levels for "network levels"
- Attacker actions are limited to scans and a homogenous list of exploits
- Defender not modeled

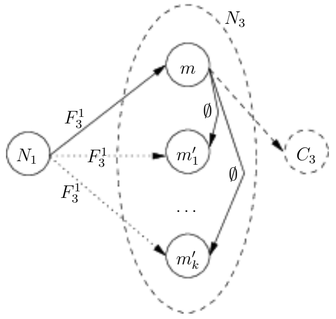
source: [7]



(a) LN as tree of components C.



(b) Paths for attacking C_1.



(c) Attacking N_3 from N_1, using m first.

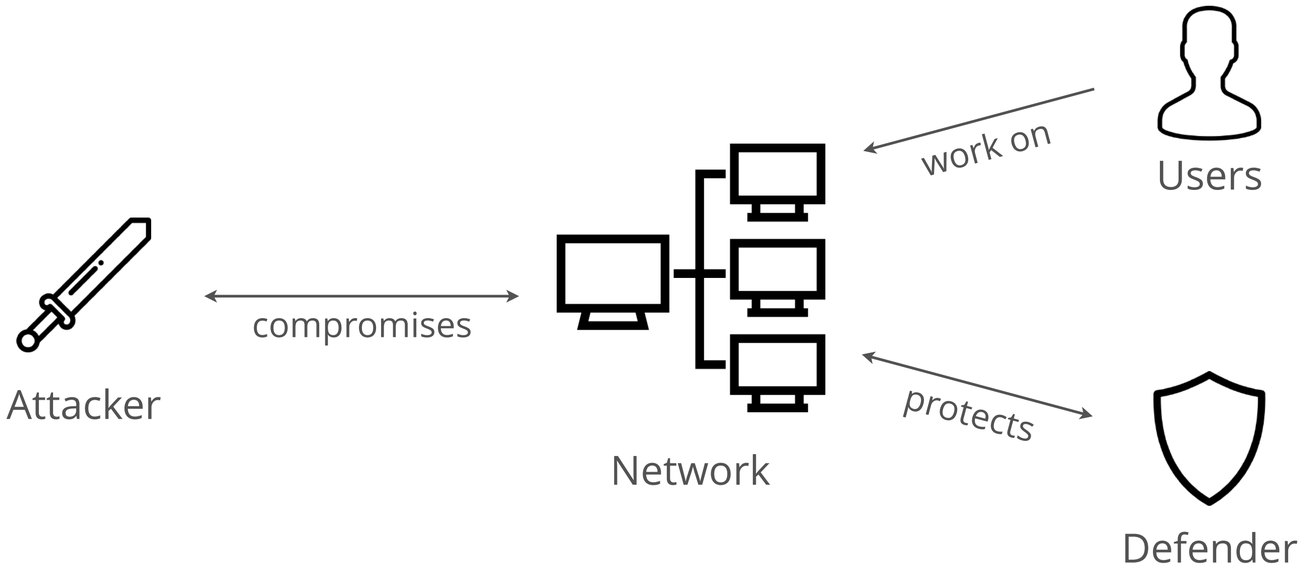
3

GAME DEFINITION

FORMALIZATION

- Model as a game between an attacker and a defender
- High-level abstraction
  - As close as possible to the real world
  - But still feasible to implement
- Described as a Partially Observable Markov Decision Process
- Outcome = the attacker's strategy:
  what action to pick, in a given environment state

GAME ACTORS


work on
Users
compromises
Attacker
Network
protects
Defender

NETWORK

- Models an enterprise network
- Star topology — most common
- Each machine has some local admins
- Each machine has a number of vulnerabilities (governed by the type of user)
- Connections are bi-directional and static

GREY AGENT

- Models noise generated by a normal users
- Has no objective, performs benign activity
- Probabilistically performs one of:
  - reboot a machine
  - log in to other machines
  - add vulnerabilities to a machine

ATTACKER

- Models a penetration tester
- (who mimics a malicious infiltrator)
- Follows a specific goal
  - exfiltrate some piece of sensitive data
  - gain as wide foothold as possible
  - depends on the scenario
- Actions modeled after Mitre's ATT&CK classifications

ATTACKER ACTIONS

- **enumerate** to reveal connections
- **scan** machine vulnerabilities

RECONNAISSANCE

- **exploit** a discovered vulnerability
- **migrate** to another machine
- **login** using dumped creds

GAINING FOOTHOLD

- **escalate** session
- **persist** against reboots & detection
- **dump** creds of local admins
- **exfiltrate** data on the machine
- **cleanup** to not get found later

POST-EXPLOIT

ATTACKER ACTIONS

- **wait** to let defender cool down
- **evade** next action will be stealthier
- **abandon** when payoff < risk

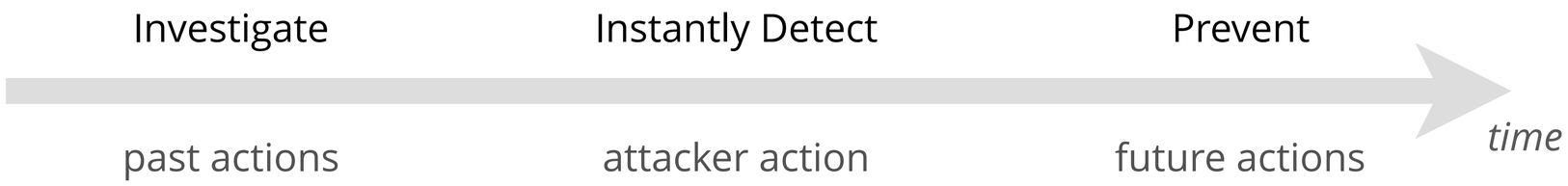
NON-TARGETED

ATTACKER ACTIONS COSTS

- **reliability**: probability of success
- **duration**: time steps taken
- **noise**: chance of detection
- **reward**: positive or negative
- **crash chance**: lose foothold, takes longer (only for exploits)

DEFENDER

- Models counteractions performed by:
  - an anti-virus solution (automatically) or
  - a security officer (manually)
- Aims to counter the attacker



DEFENSE MEASURES

- Instant Detection
  - based on action's noise
  - fended off by evasive maneuvers
  - kicks attacker off, warrants more attention
- Investigation
  - based on defender's suspicion
  - increases with each action, decreases with time
  - kicks attacker off, patches vulnerabilities

DEFENSE MEASURES

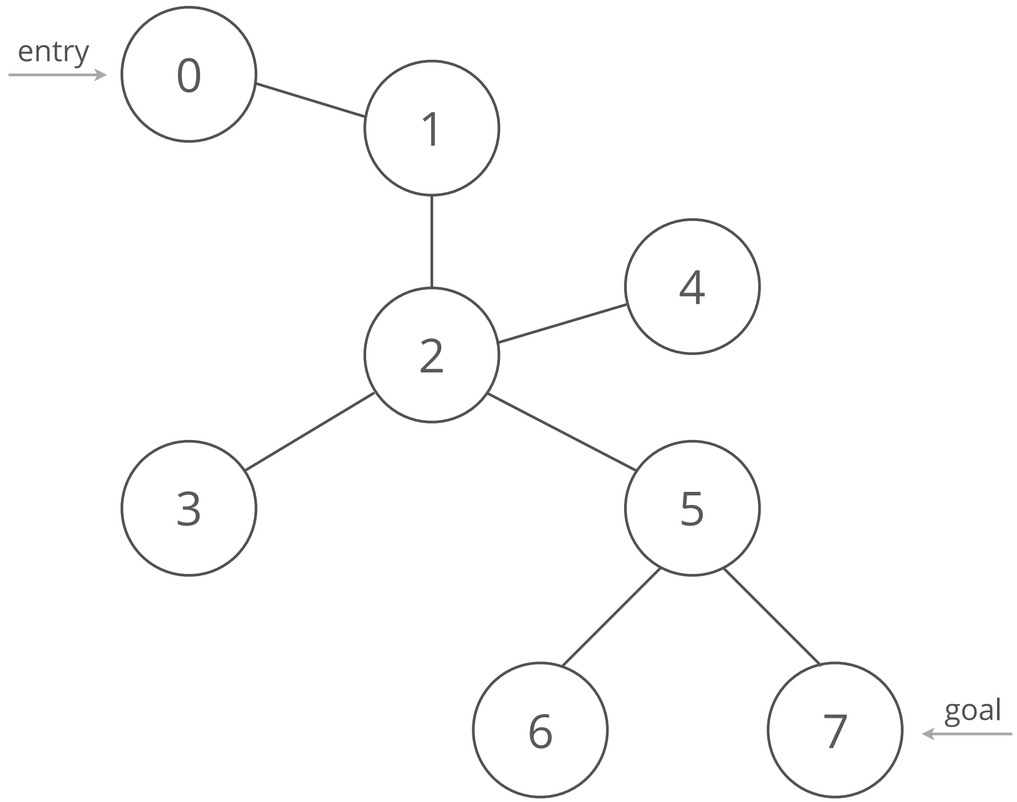
- Prevention
  - has a number of "attention resources"
  - allocates to key places
  - one resource blocks one action on a machine
- Protect those places that are valuable & often targeted
- Akin to a *Security Game* coming from Game Theory

GAME RULES

- Turn-based
- Actors act concomitantly
- End conditions:
  - Data exfiltrated from goal machine
  - Attacker gave up
  - Time limit reached
  - No more moves available

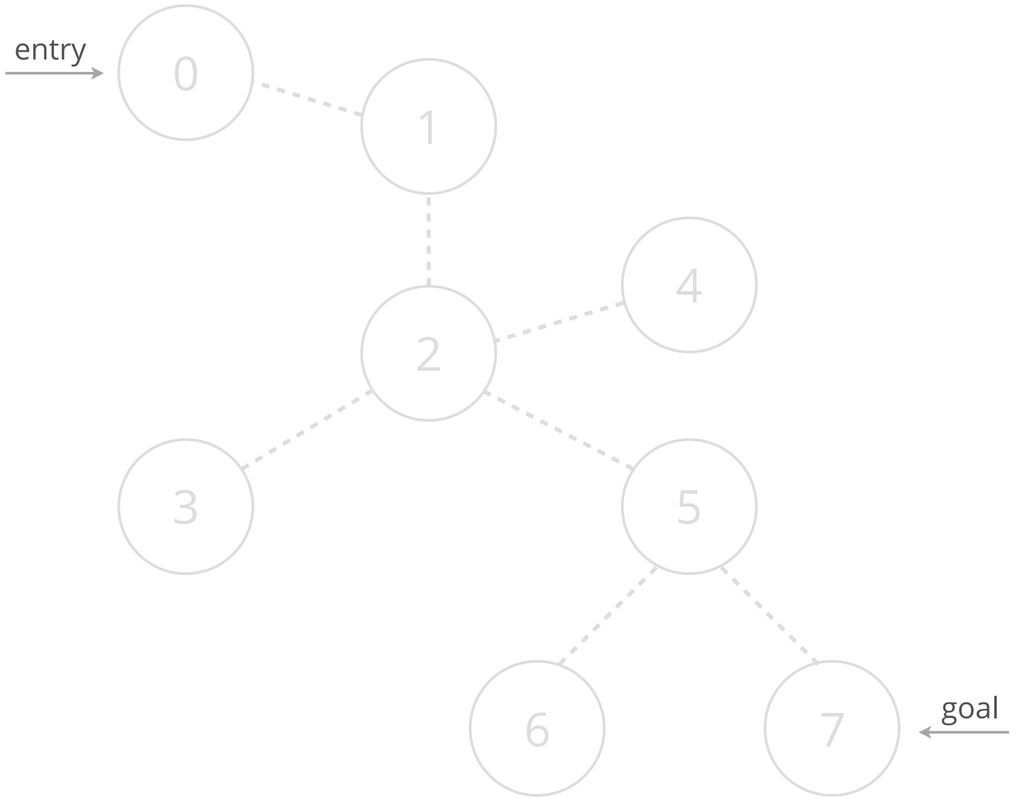
EXAMPLE RUN

- Best way to understand the task
- Manually perform steps
- On a simple network
- From start, to episode completion



EXAMPLE

⌛ action duration
💎 reward received
✗ failure reason



CONNECTION
- undiscovered
- discovered

MACHINE STATUS
- unknown
- scanned
- foothold
- elevated

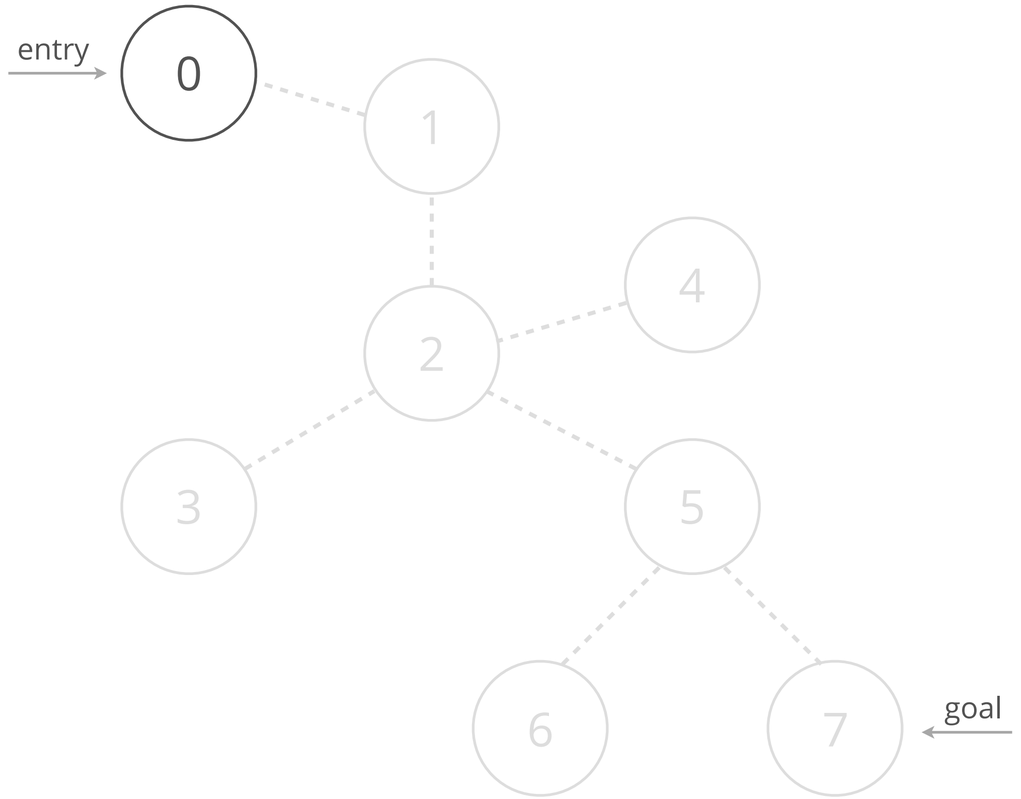
PERFORMED
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

scan: 0

4



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

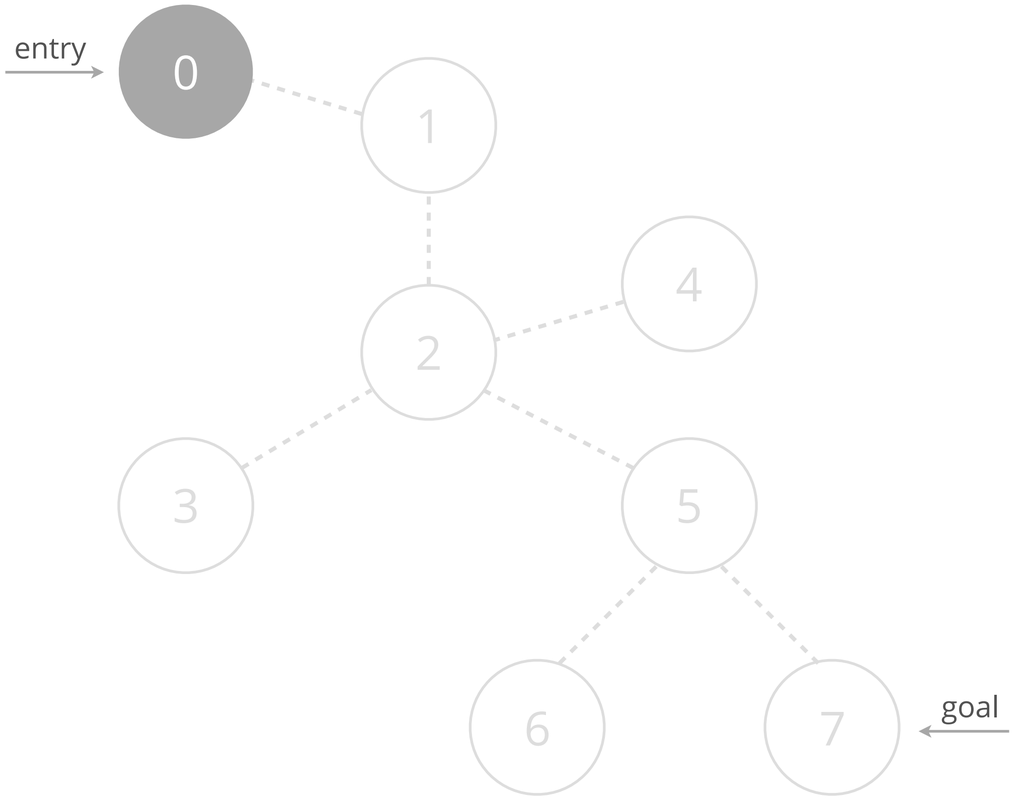
EXAMPLE

Action

exploit A: 0

2

+5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

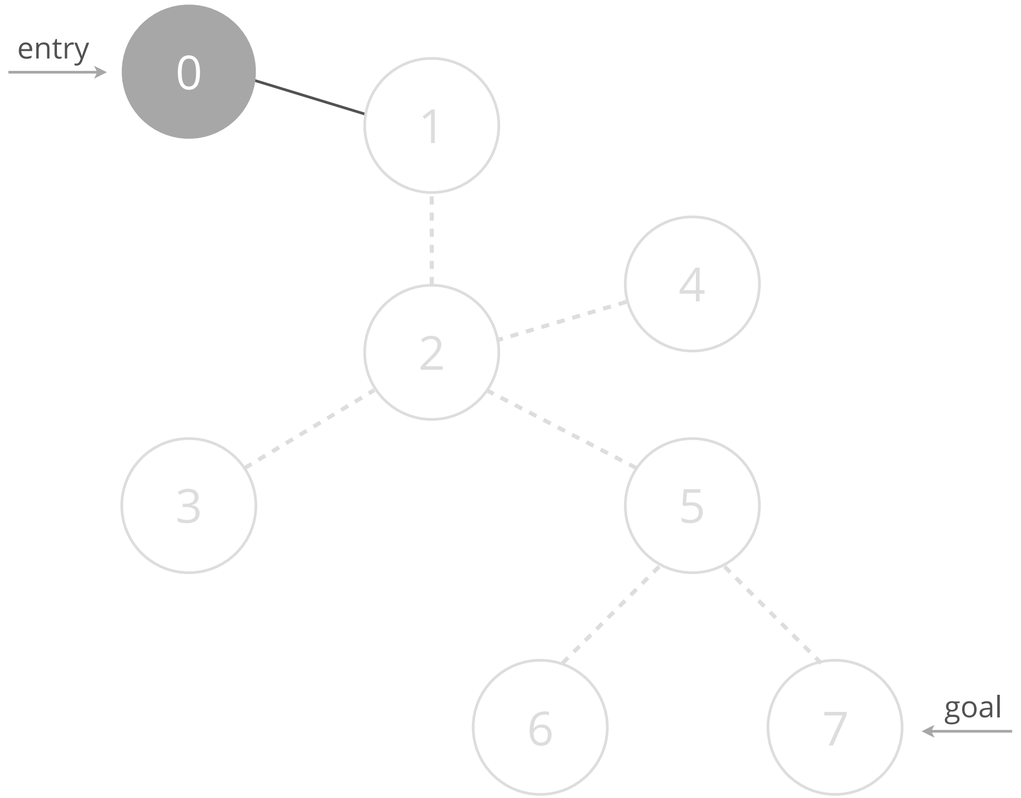
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

enumerate: 0

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

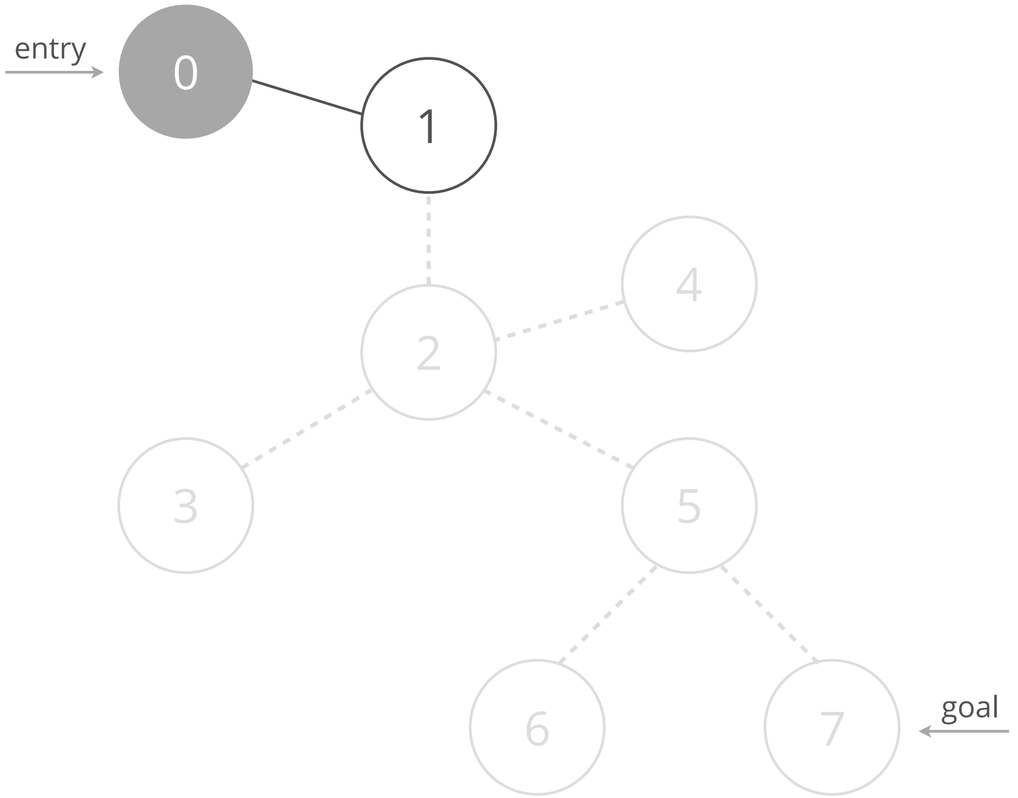
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

scan: 1

4



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

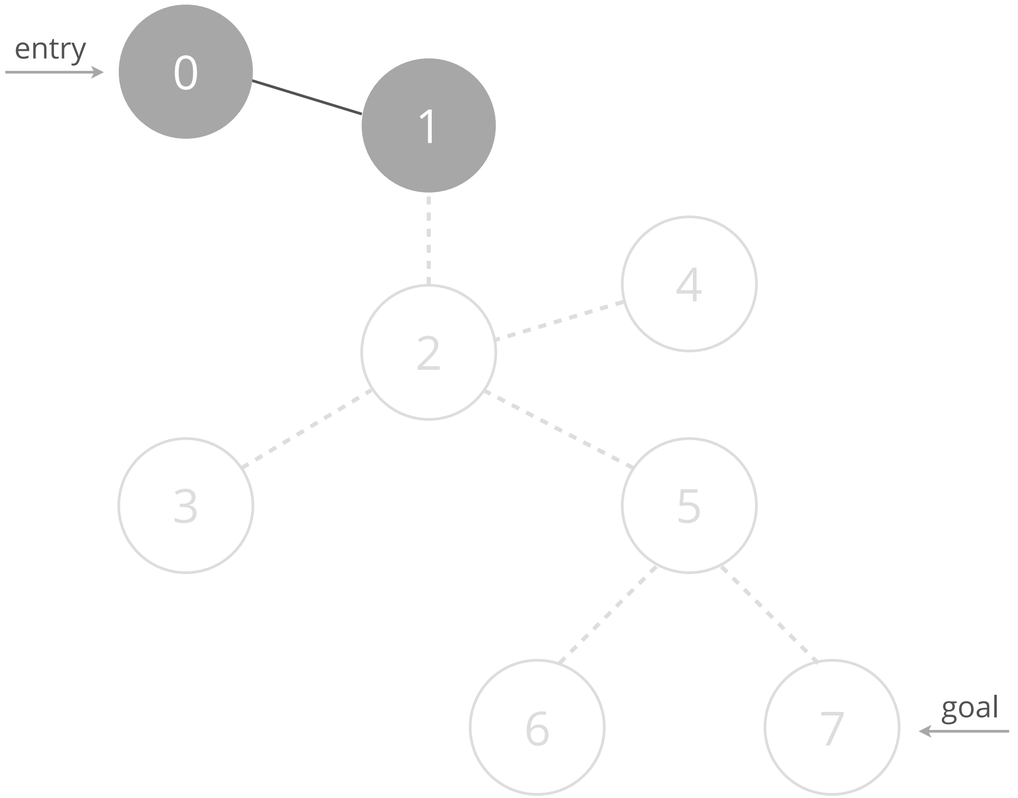
EXAMPLE

Action

exploit A: 1

2

✗ crashed



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

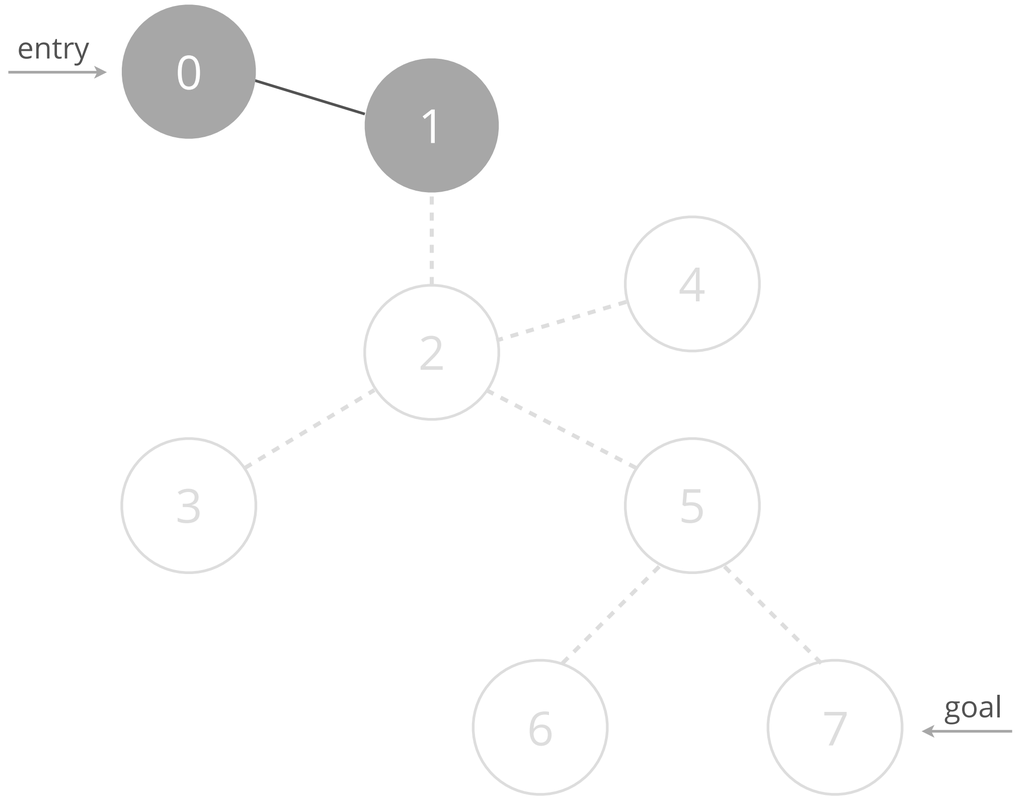
EXAMPLE

Action

exploit B: 1

3

+5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

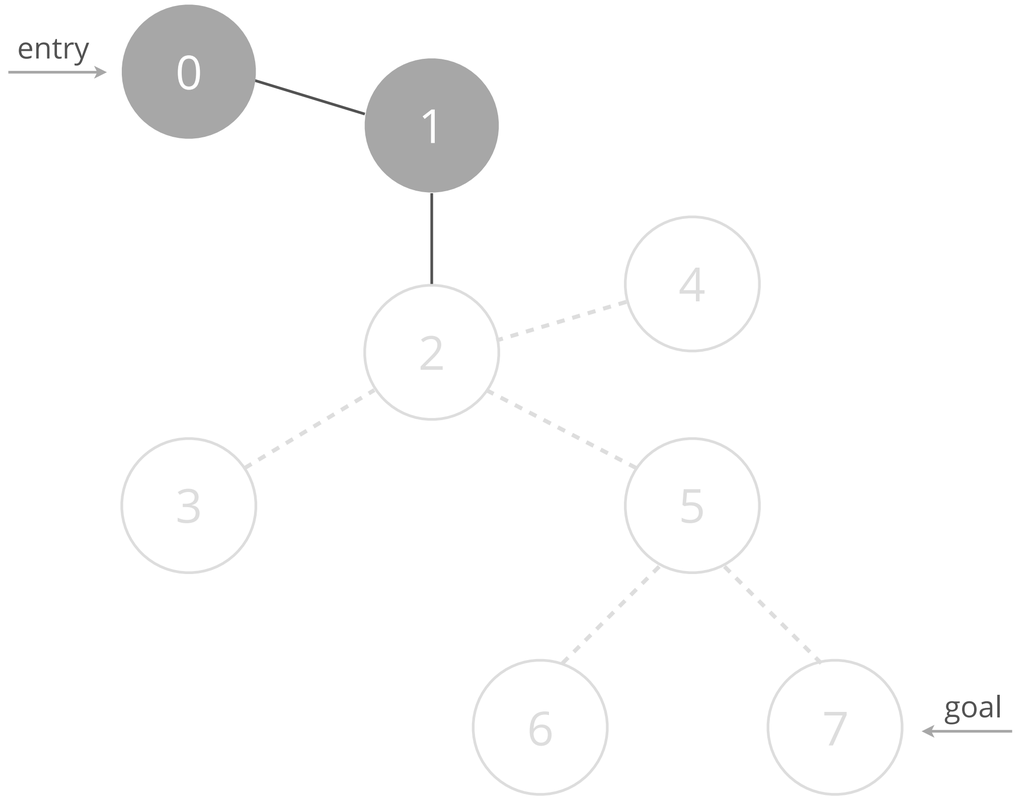
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

enumerate: 1

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

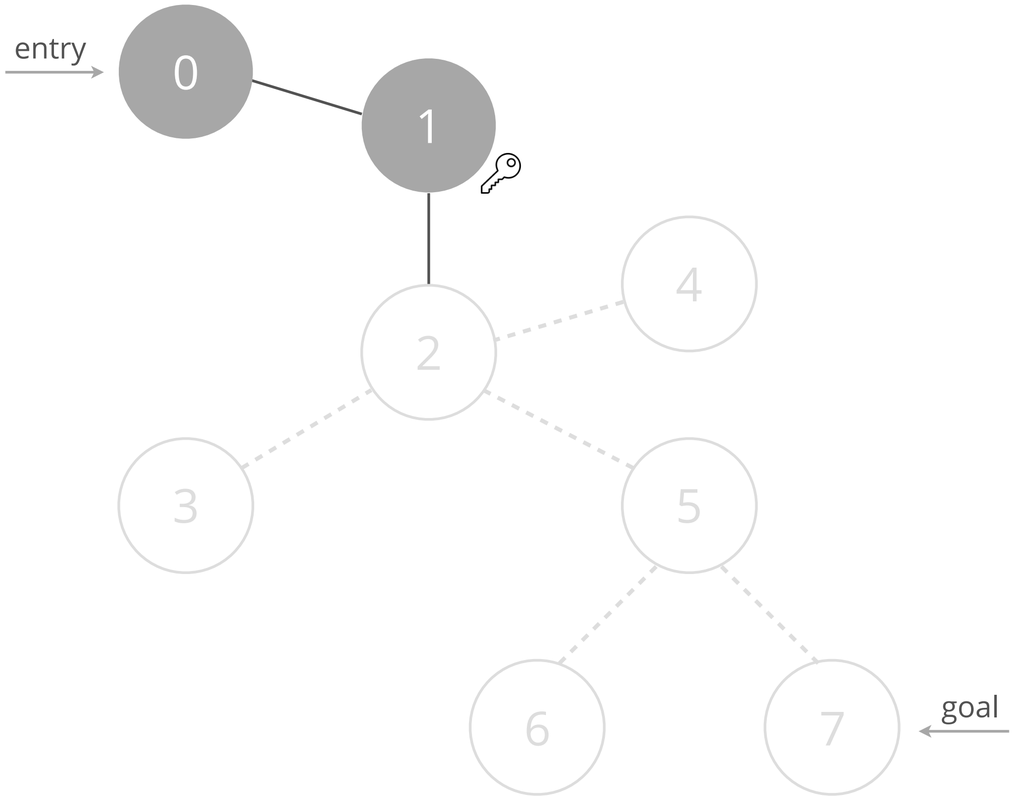
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

dump: 1

2



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

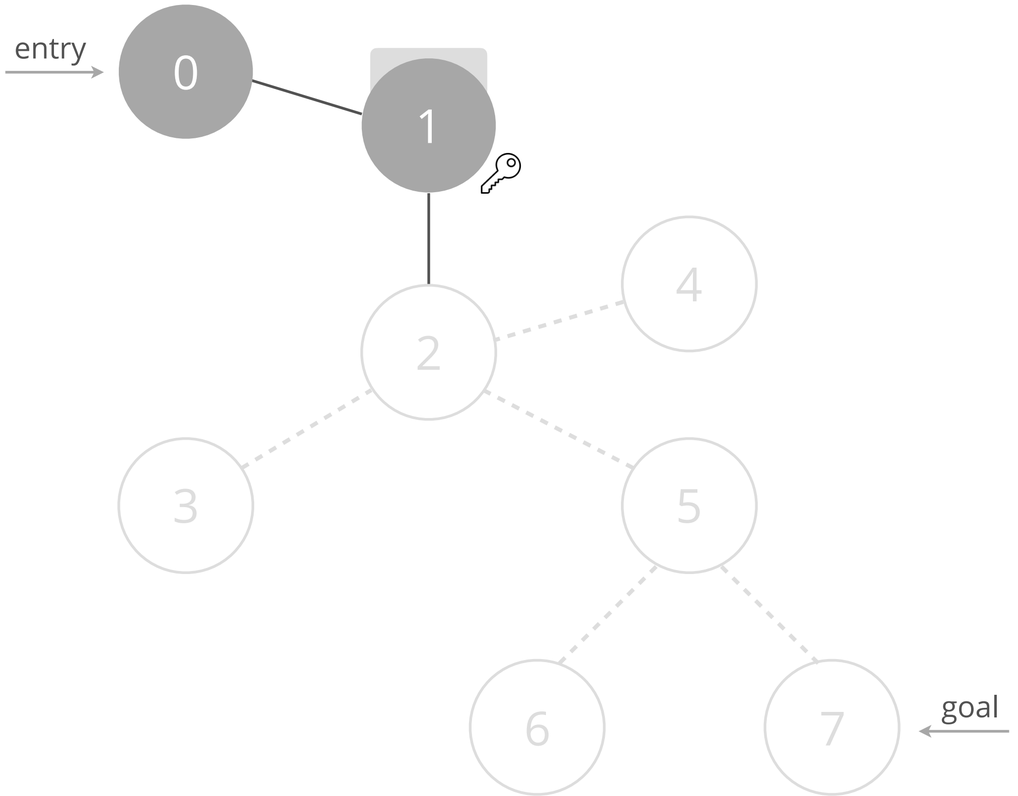
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

cleanup: 1

3



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

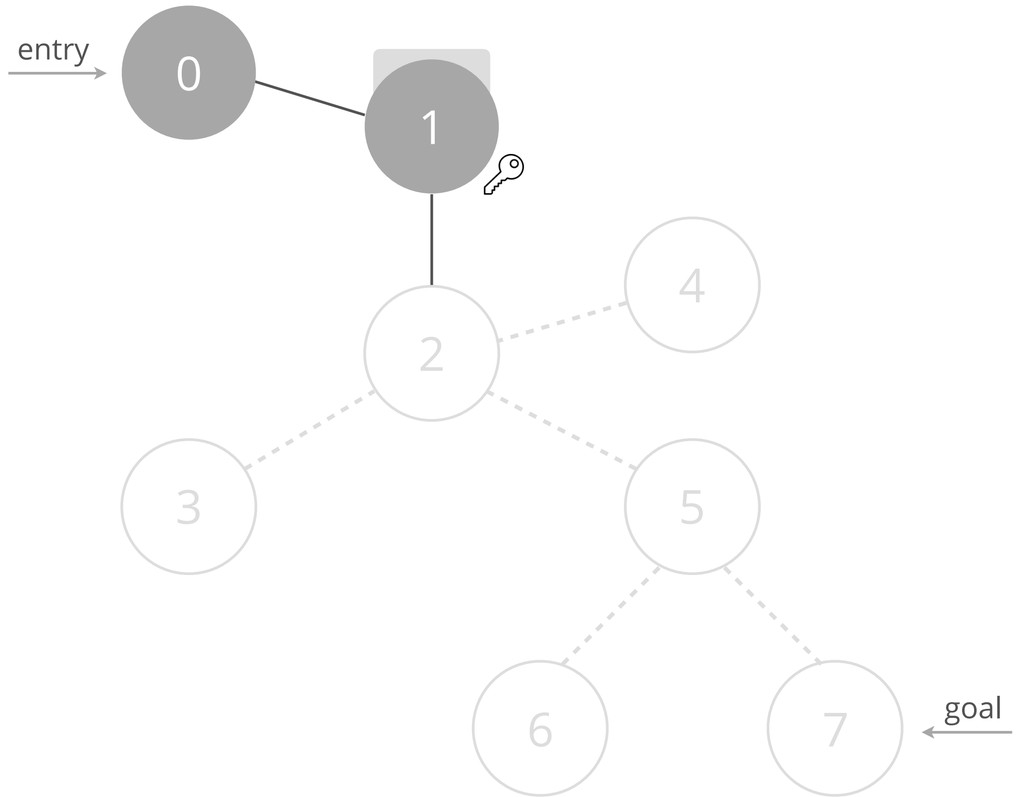
EXAMPLE

Action

scan: 2

4

✗ failed



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

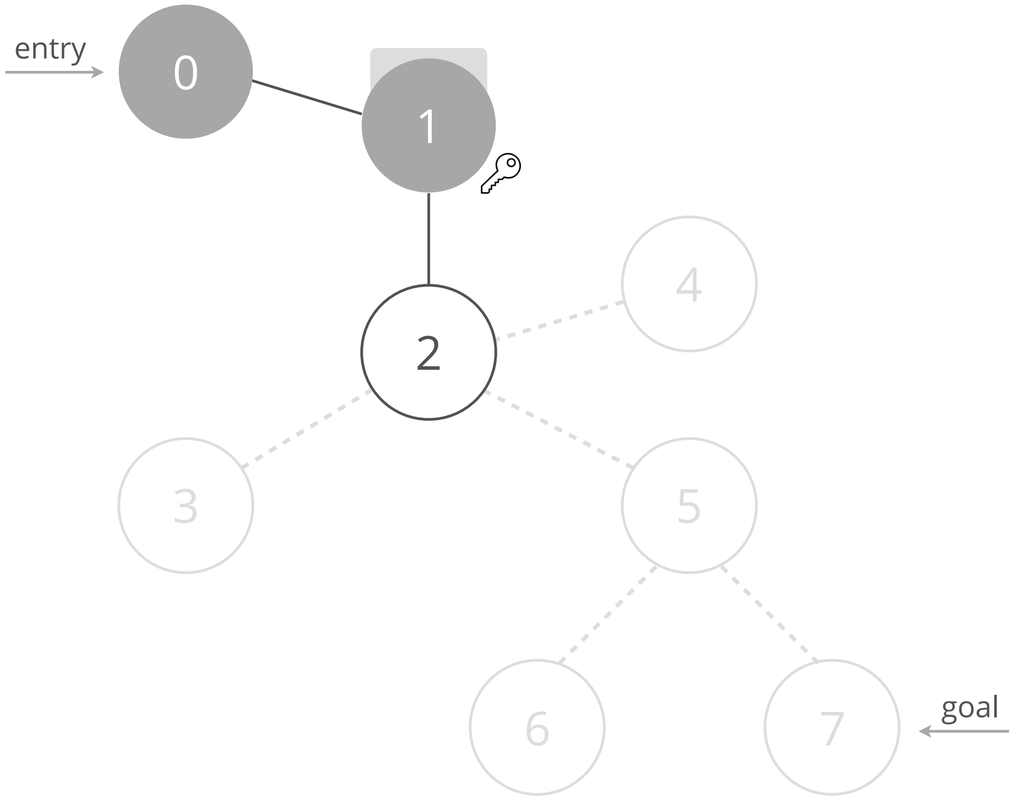
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

scan: 2

4



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

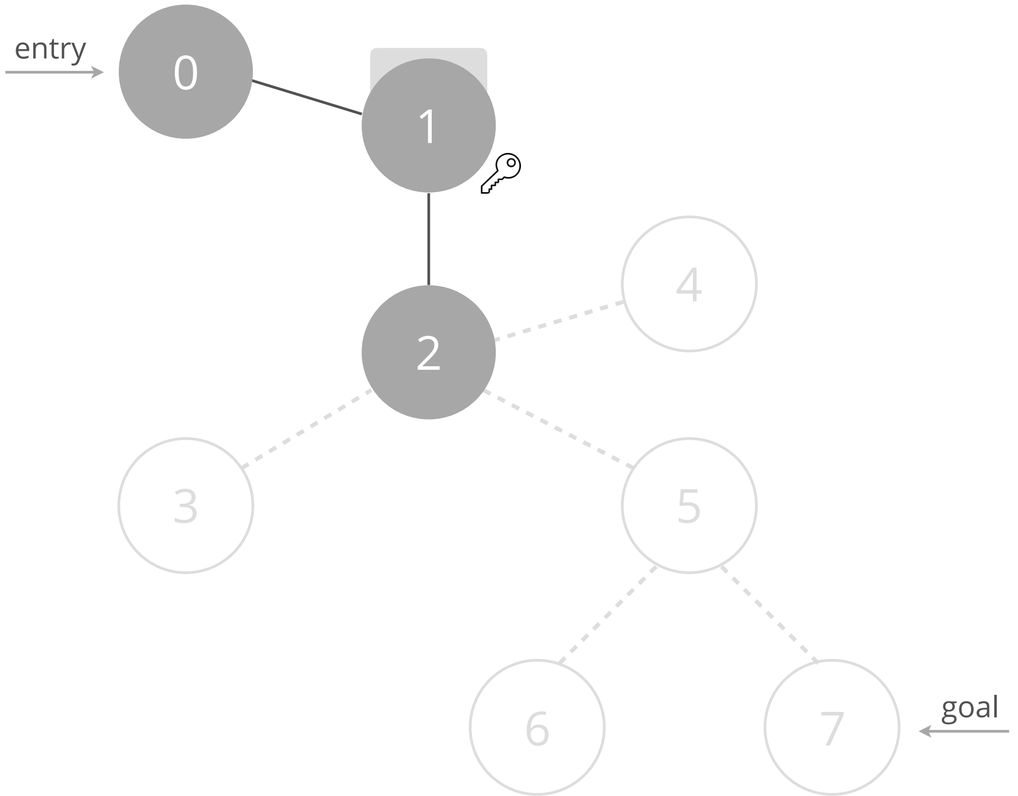
EXAMPLE

Action

exploit C: 2

2

+5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

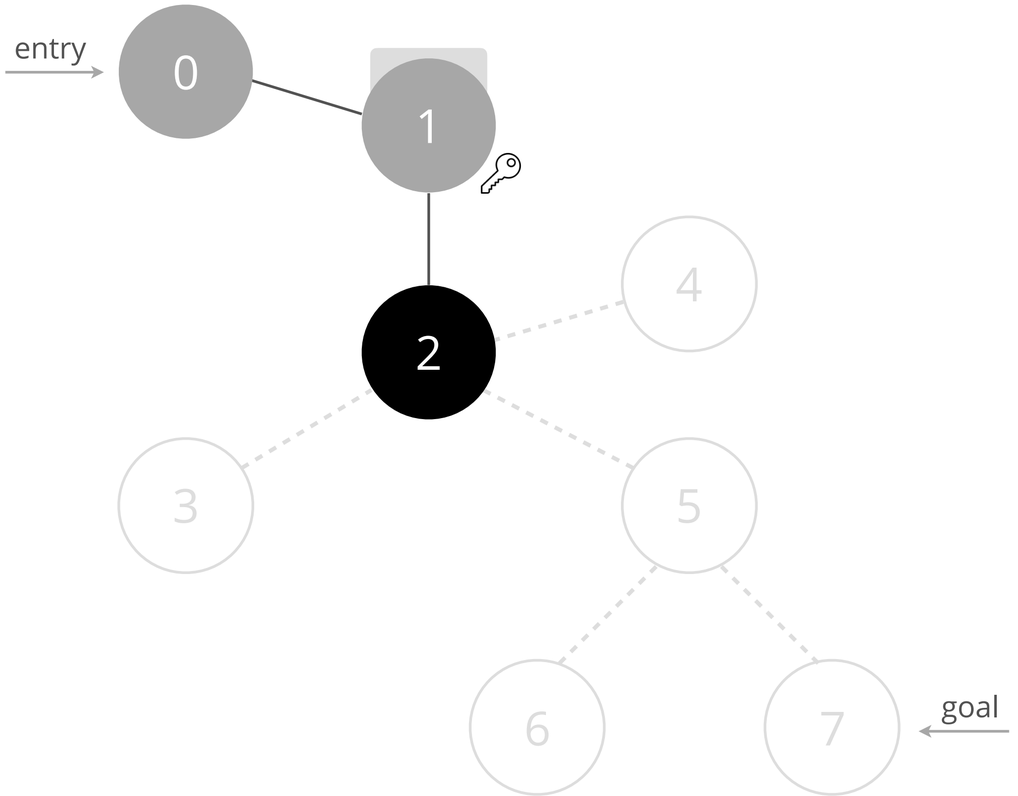
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

escalate: 2

2



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

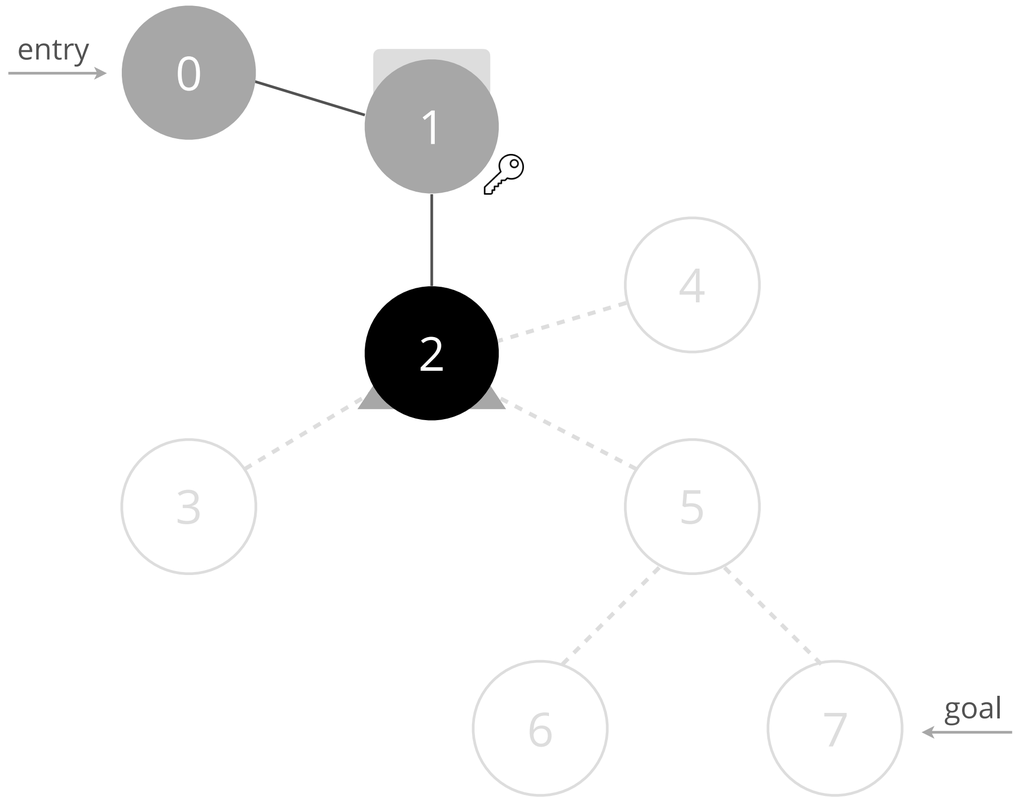
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

persist: 2

2



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

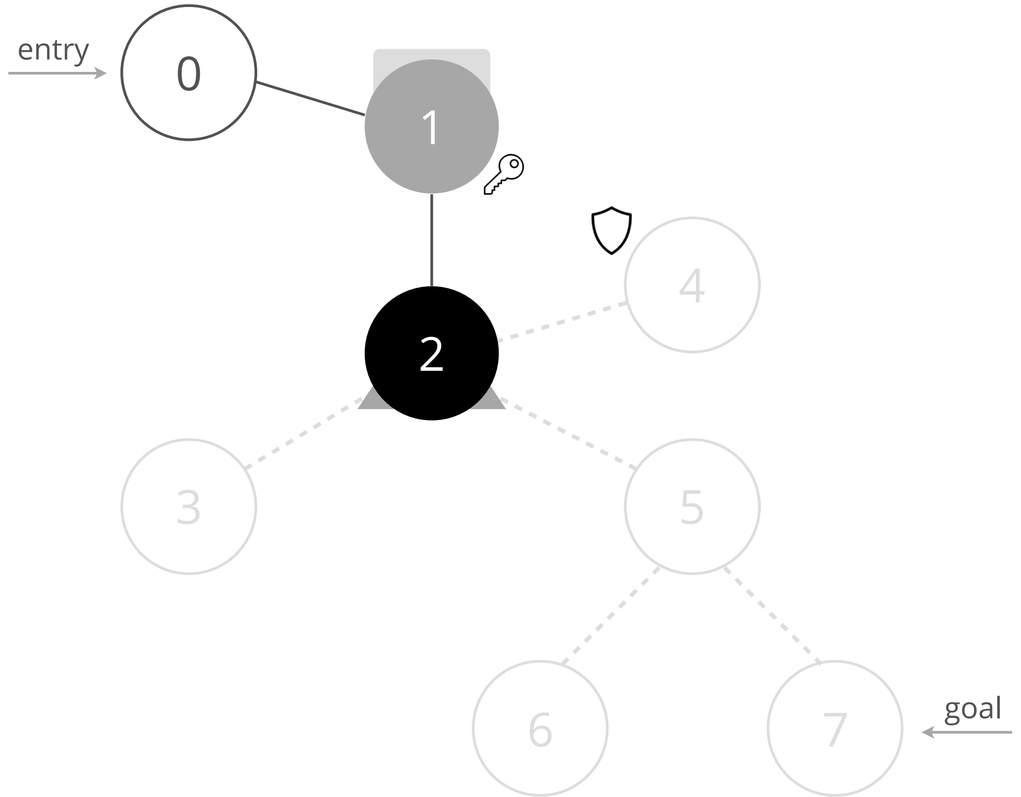
EXAMPLE

Action

investigate: 0

-10

✗ caught



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

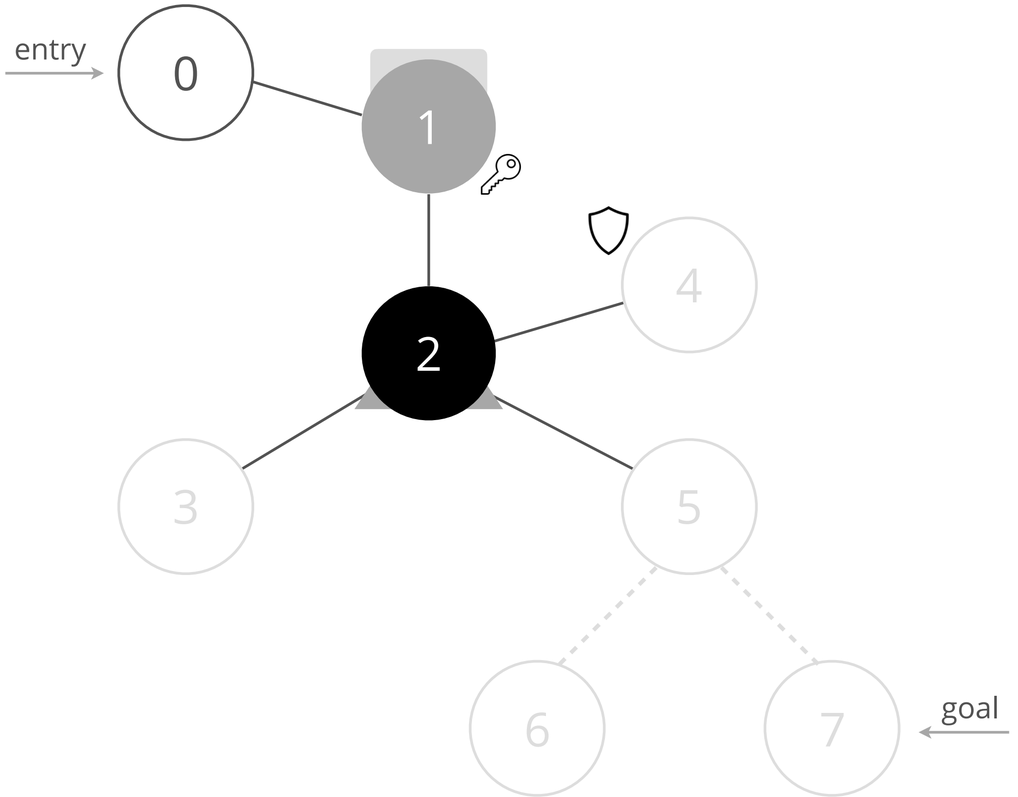
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

enumerate: 2

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

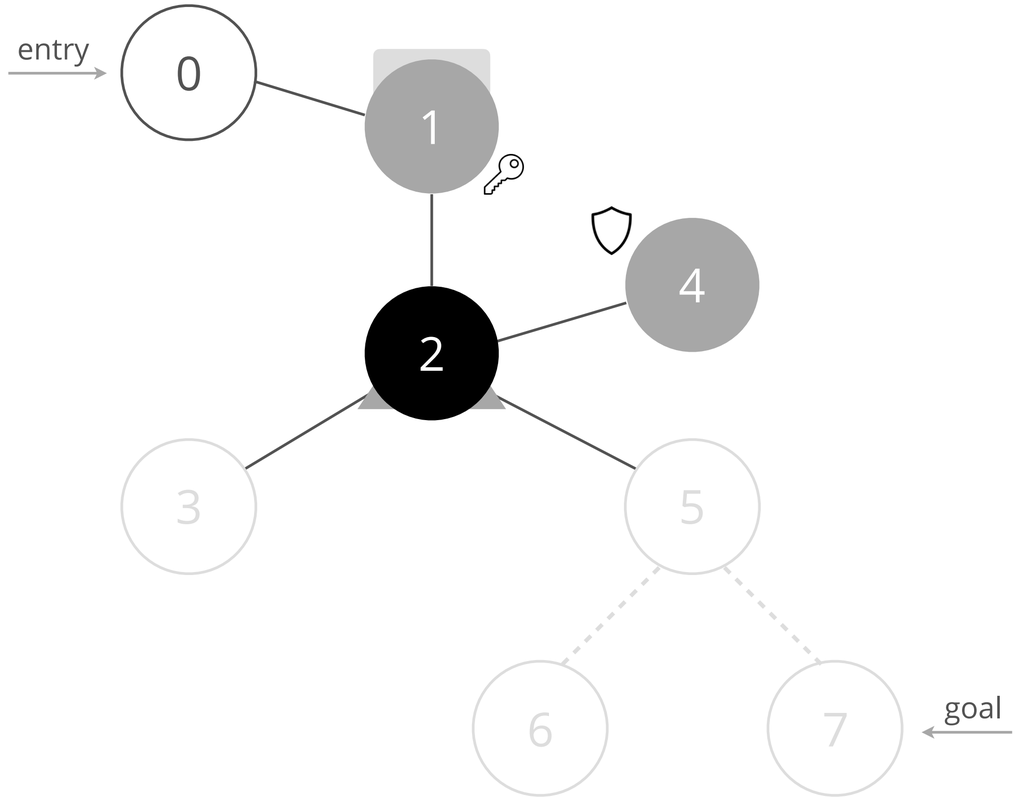
EXAMPLE

Action

migrate: 4

2

+5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

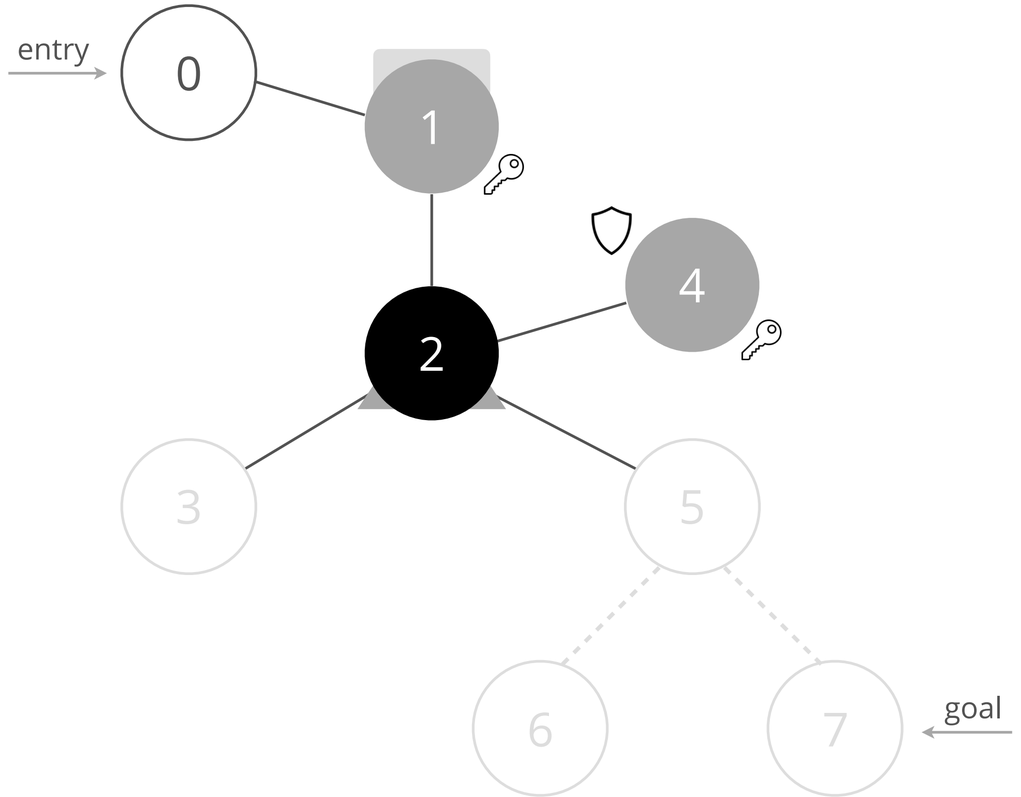
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

dump: 4

1



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

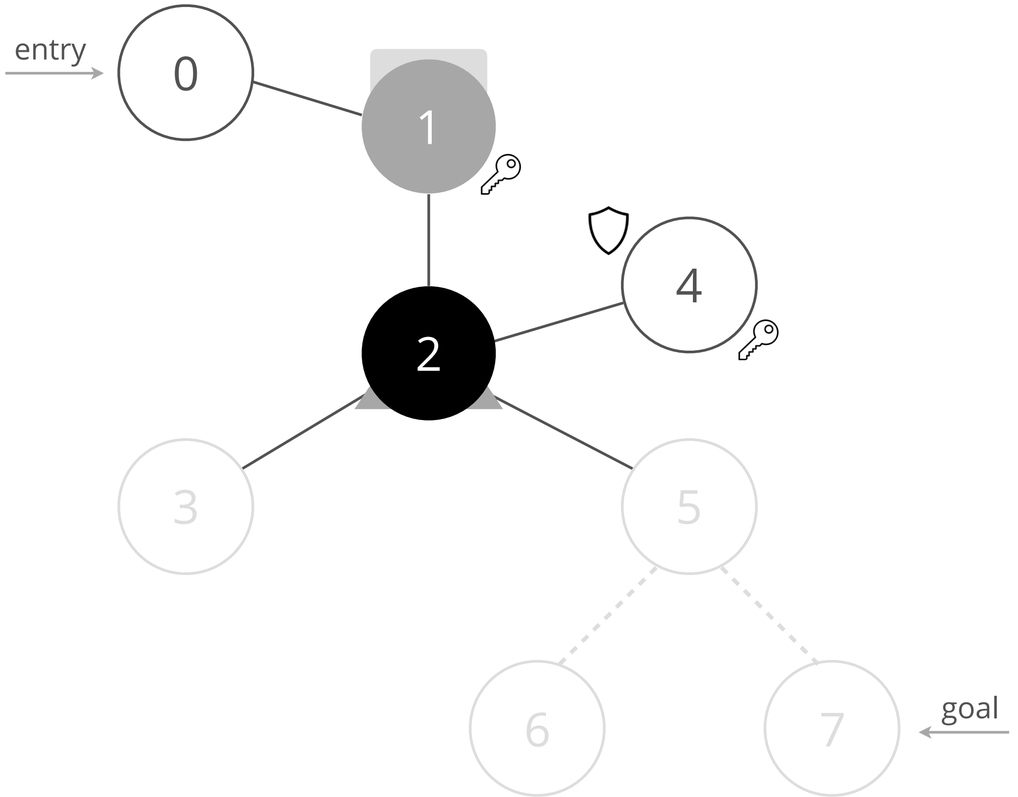
EXAMPLE

Action

exfiltrate: 4

1

✗ blocked



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

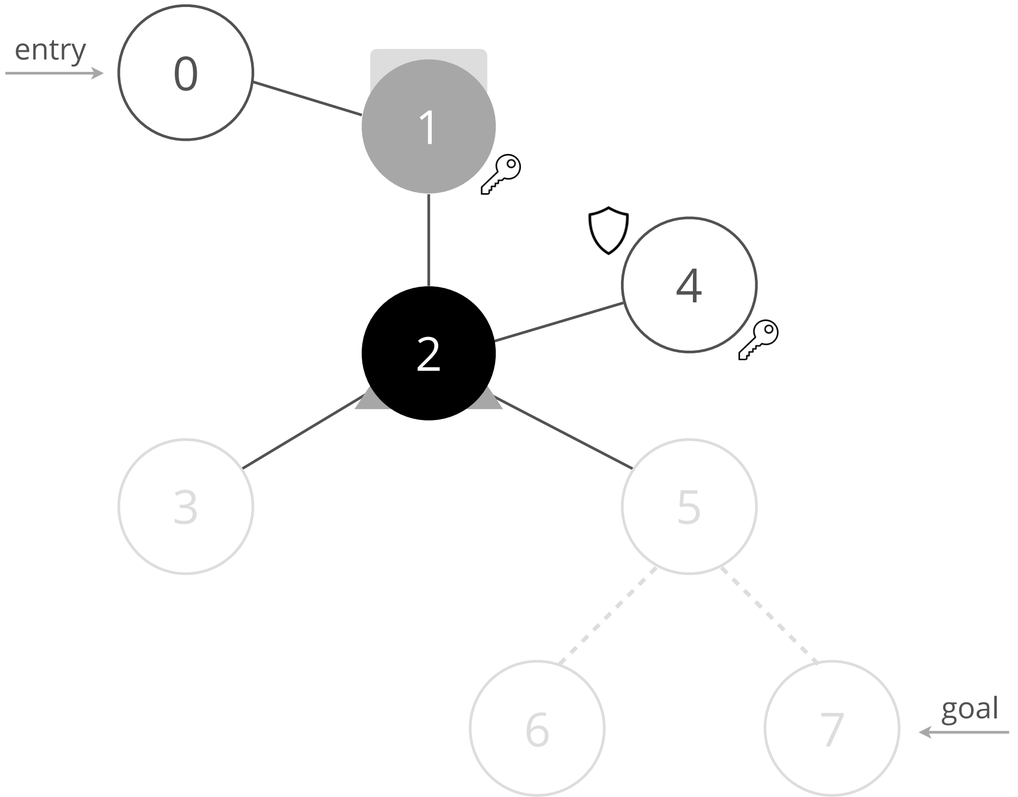
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

wait

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

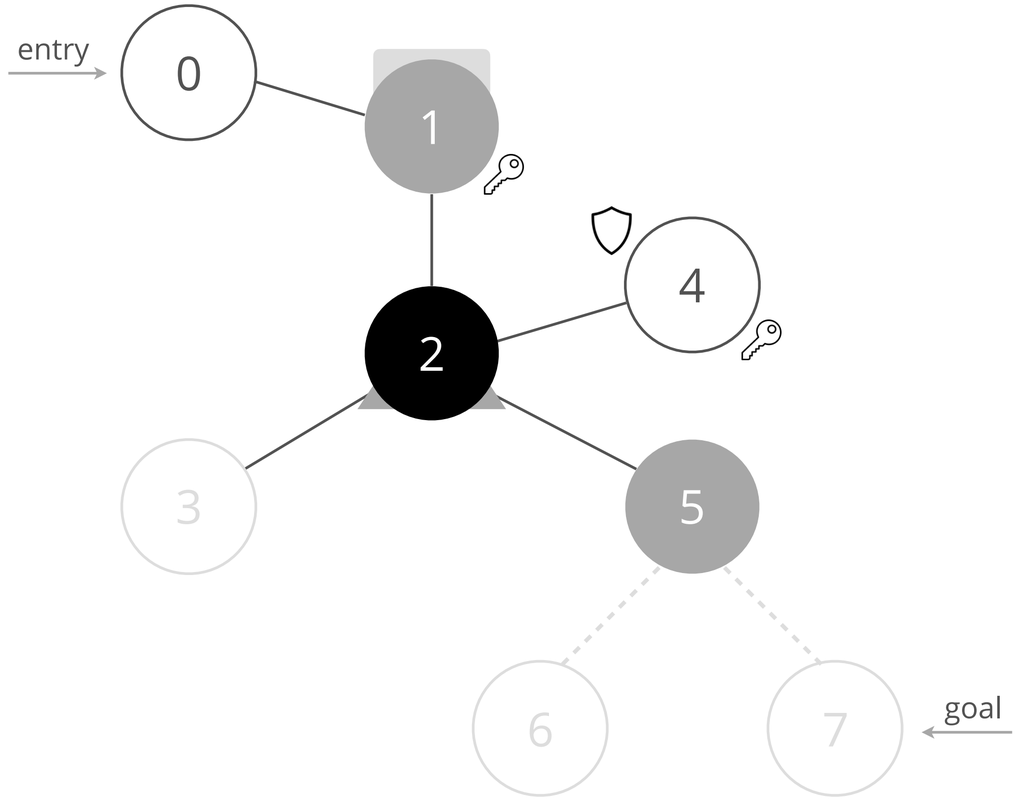
EXAMPLE

Action

migrate

2

+5 (foothold)



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

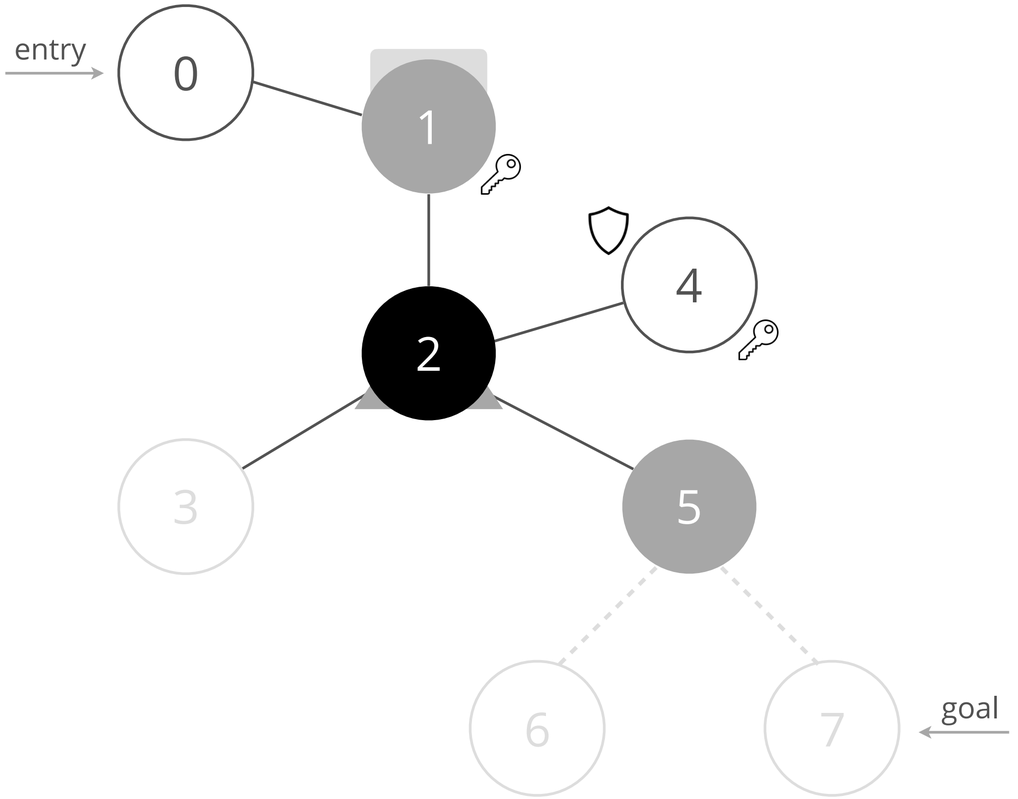
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

wait

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

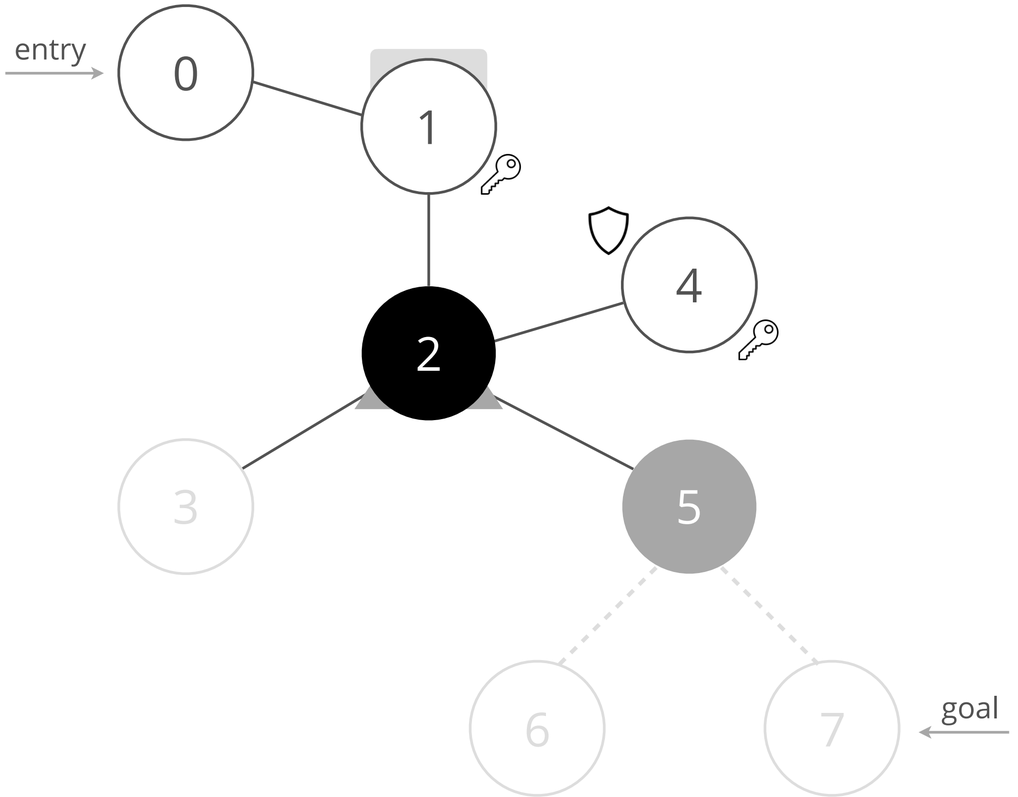
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

reboot: 1

✗ lost foothold



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

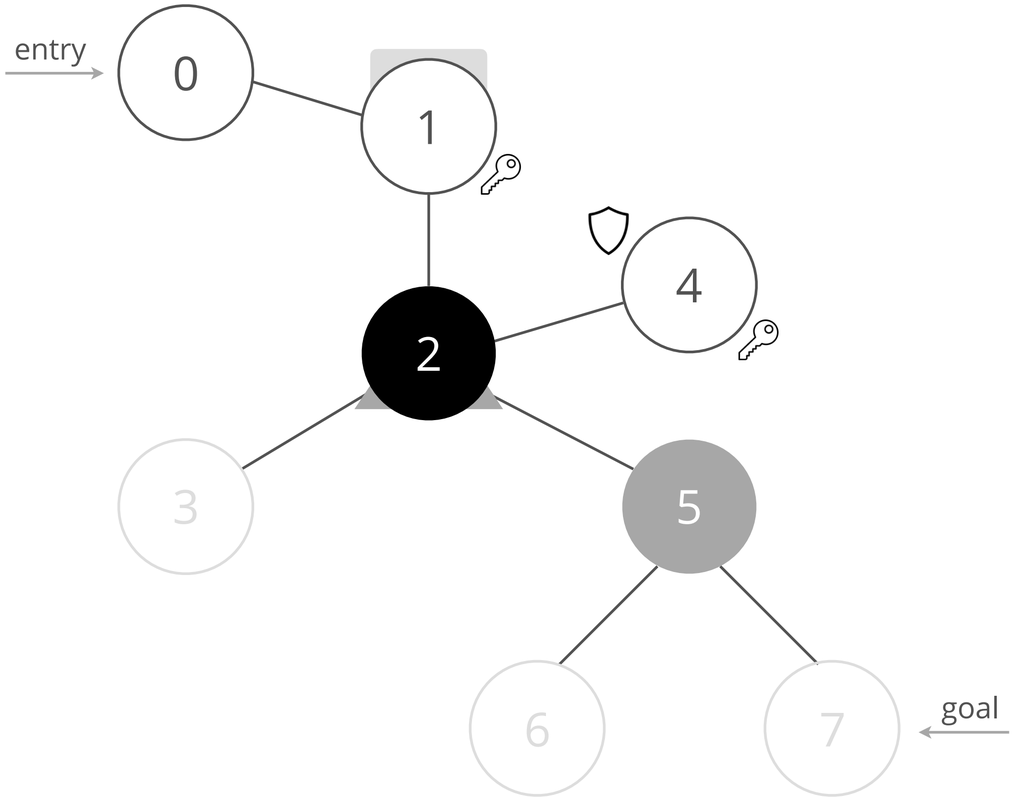
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

enumerate: 5

5



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

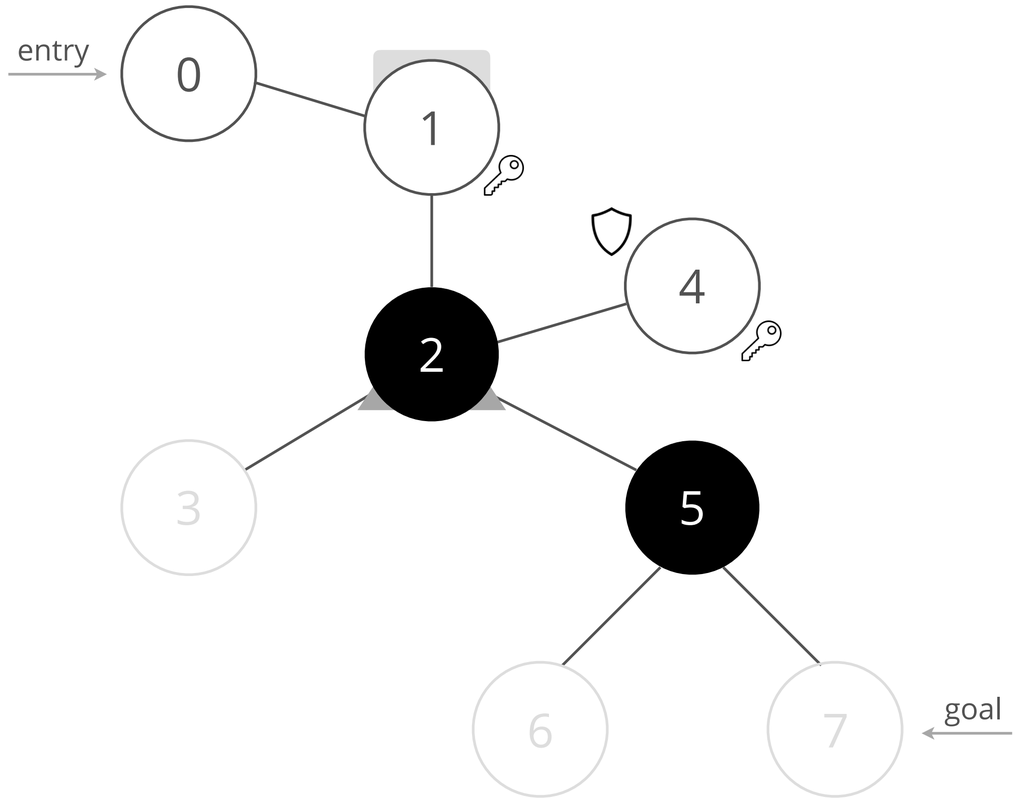
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

escalate: 5

2



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

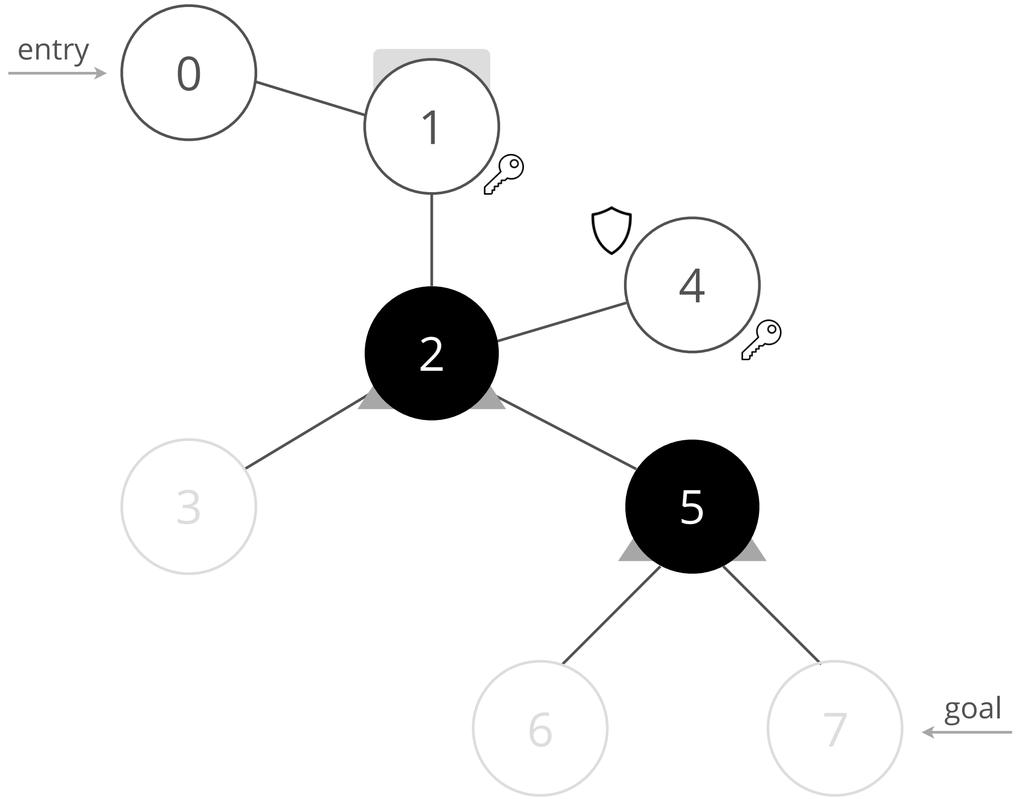
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

persist: 5

2



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

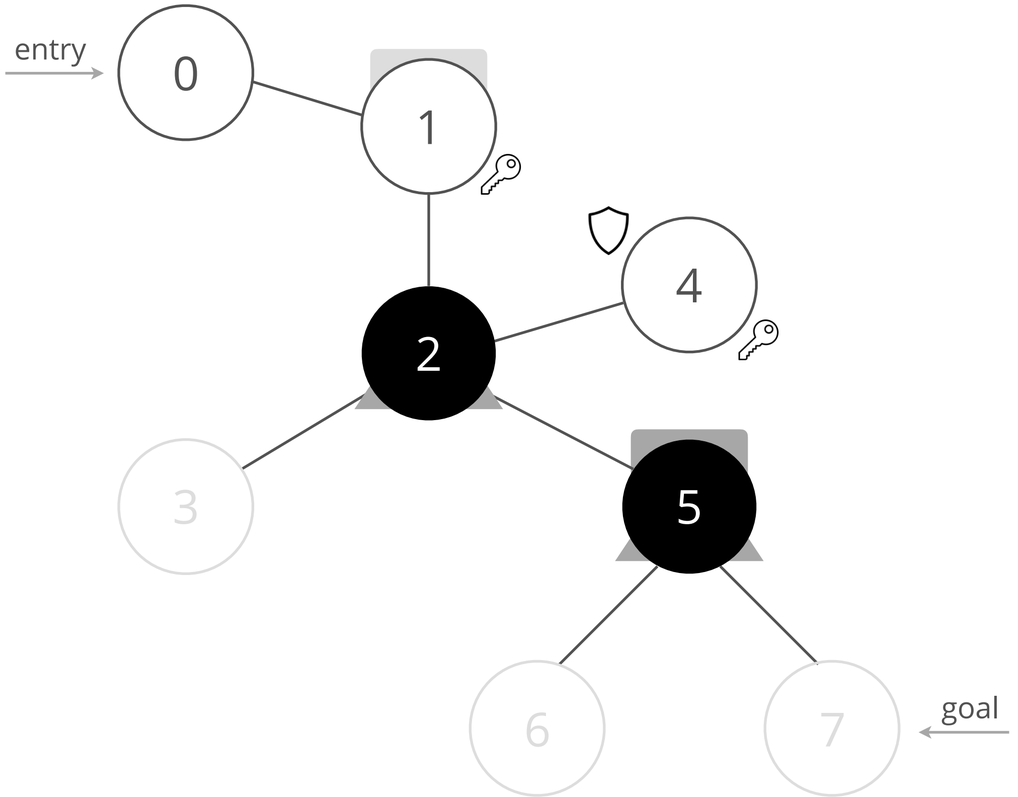
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

cleanup: 5

3



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

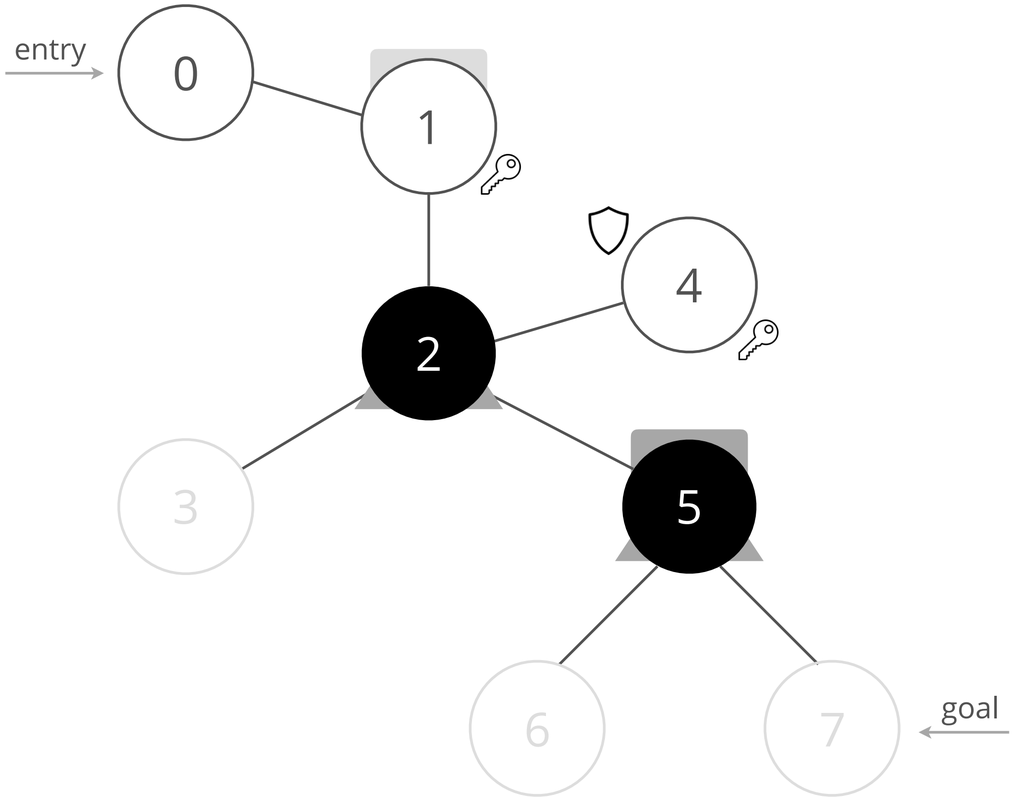
- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

evade

3



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

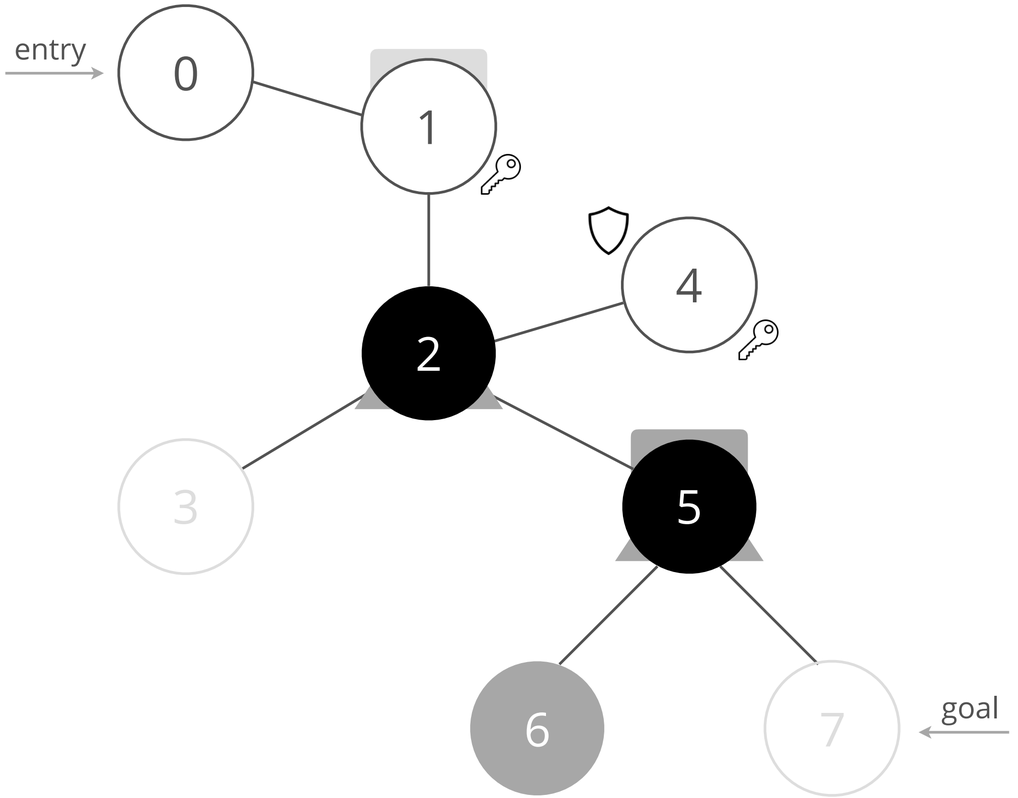
EXAMPLE

Action

login: 6

1

+5 (foothold)



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

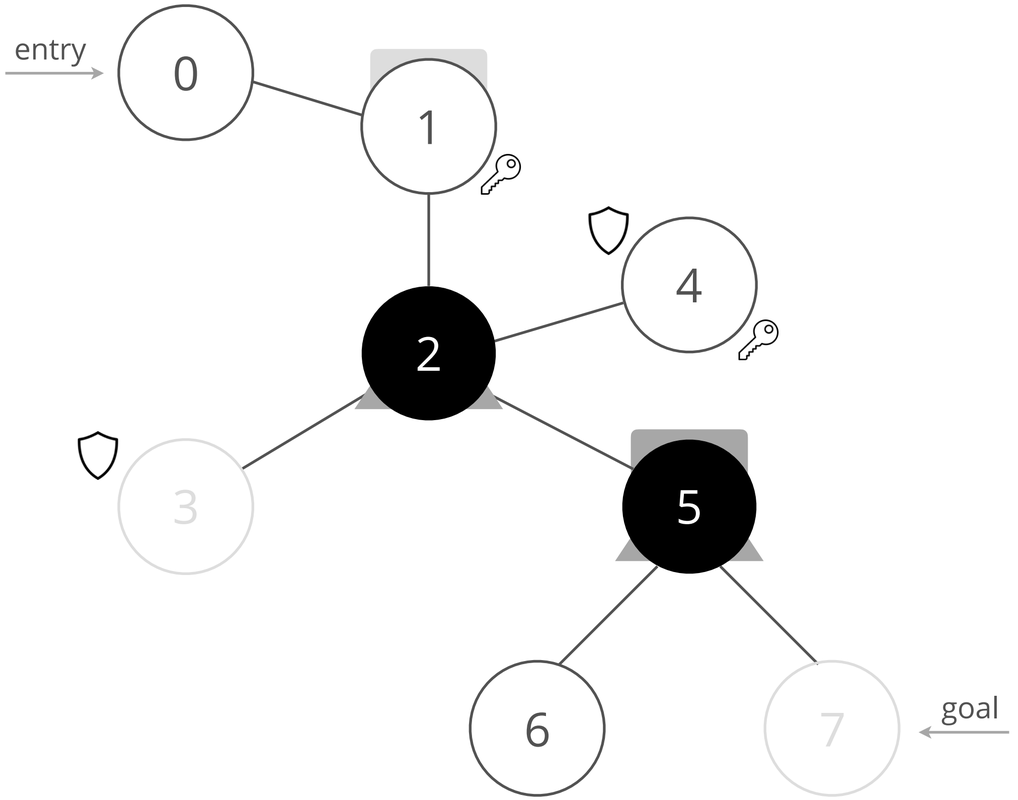
Action

exfiltrate: 6

1

-10

✗ detected



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

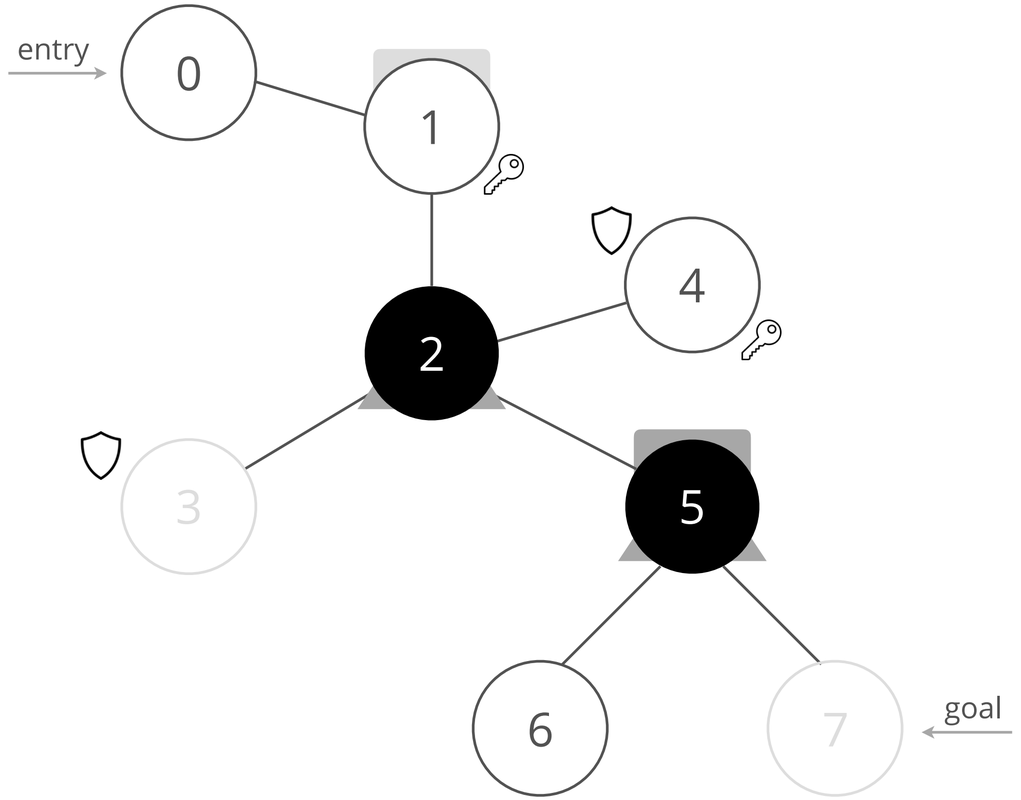
Performed

- cleanup
- persistence
- dump
- exfiltrate

EXAMPLE

Action

evade



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

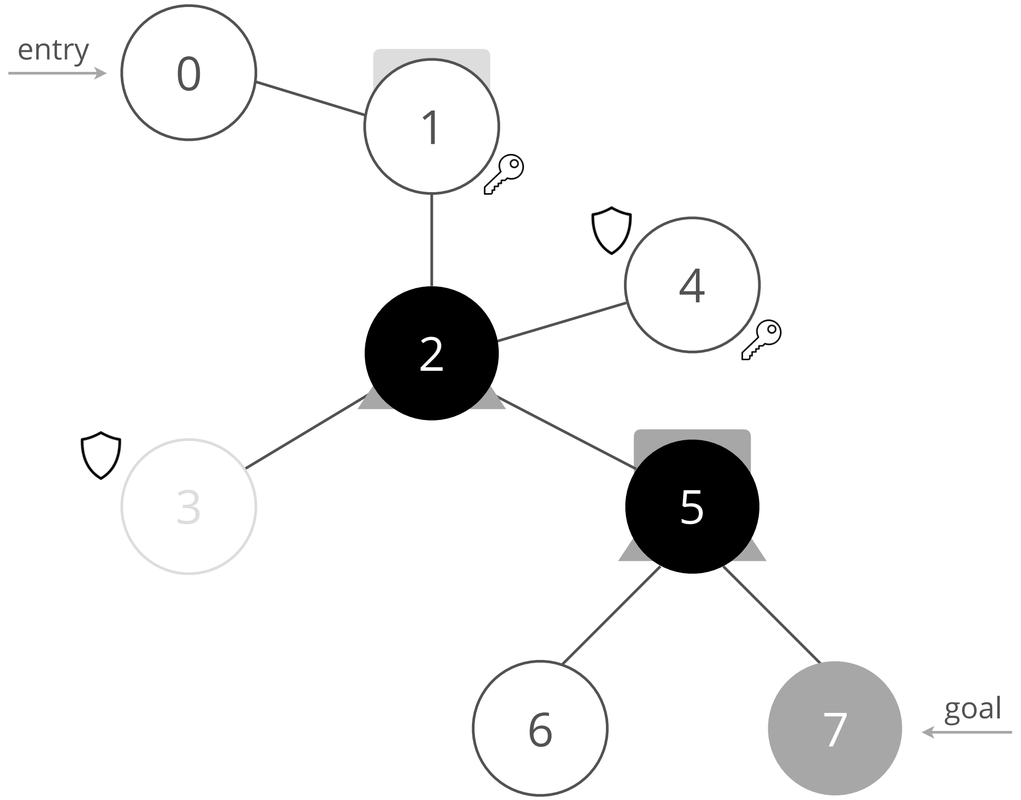
EXAMPLE

ACTION

login: 7

1

+5 (foothold)



CONNECTION

- undiscovered
- discovered

MACHINE STATUS

- unknown
- scanned
- foothold
- elevated

PERFORMED

- cleanup
- persistence
- dump
- exfiltrate

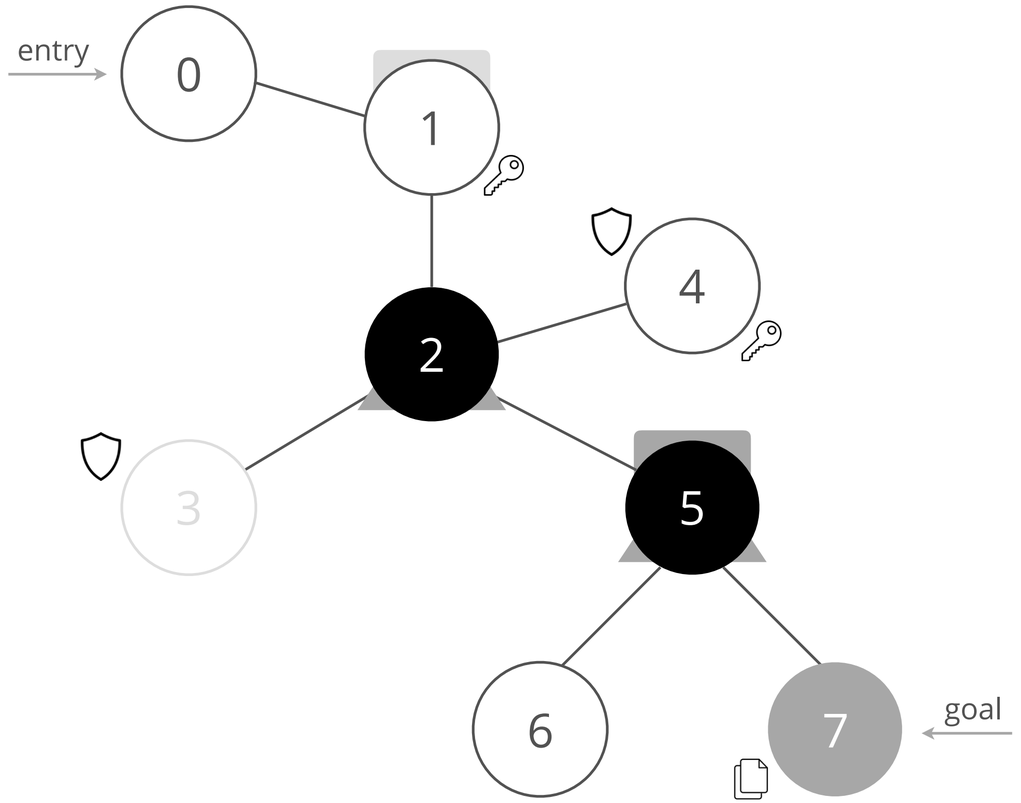
EXAMPLE

Action

exfiltrate: 7

1

+100 (goal)



Connection

- undiscovered
- discovered

Machine Status

- unknown
- scanned
- foothold
- elevated

Performed

- cleanup
- persistence
- dump
- exfiltrate

PERFORMANCE METRICS

- Objective reached:
  - exfiltrate data
  - compromise machines
  - steal credentials
- Stealthiness: times detected
- Swiftness: time steps taken
- Above encompassed by total reward
- Also encourage model parsimony

STATE

- Characterizes the env at the current time step
- Shown to the agent (what a human can deduce):
  - performed actions
  - machines compromised, connections discovered, user credentials obtained
  - available next actions
  - action properties
- Hidden:
  - network topology, position of the goal
  - what users have access where
  - defender strategy
  - grey agent probabilities

REWARDS

- Guides the agent towards desired objectives / penalizes unwanted behavior
- Positive (according to evaluation metrics / attacker objectives):
  - gain foothold (first time): small
  - exfiltrate data: small
  - exfiltrate goal: large
- Negative:
  - time: small — to encourage swiftness
  - detection: large

HOW SCENARIOS DIFFER

- Network topology
- Machines exploitability
- Action properties
- Grey agent probabilities
- Defender's strength

SCENARIO GENERATION

- Connections and vulnerabilities: synthetization of typical networks
- Exploits properties: mapping of *NVD* database
- Action properties: estimated by security experts
- Vulnerabilities presence dictated by user kind:
  - Network administrator
  - Software developer
  - Non-technical employee

4

TECHNIQUES

IMMEDIATE EXECUTING

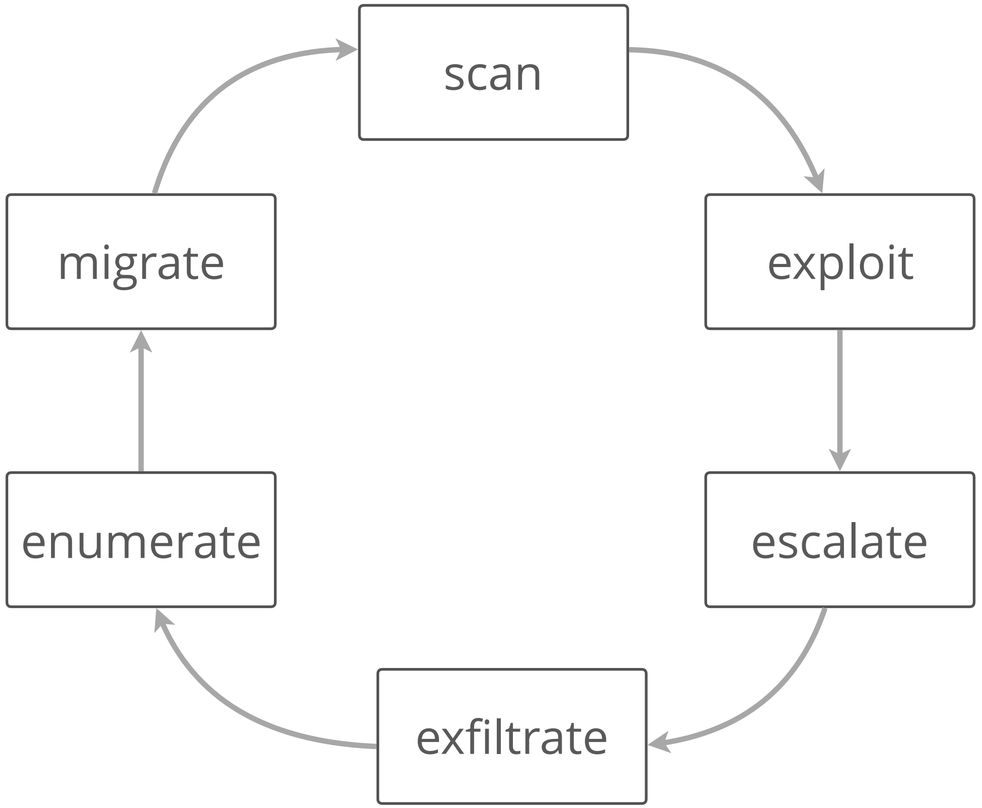
- Don't learn, fixed strategy
- Used as baseline
- Can provide initial knowledge for learning methods
- Simplest: Random Agent
    - picks one of the available actions, uniformly at random

IMMEDIATE: FSM

- Finite State Machine

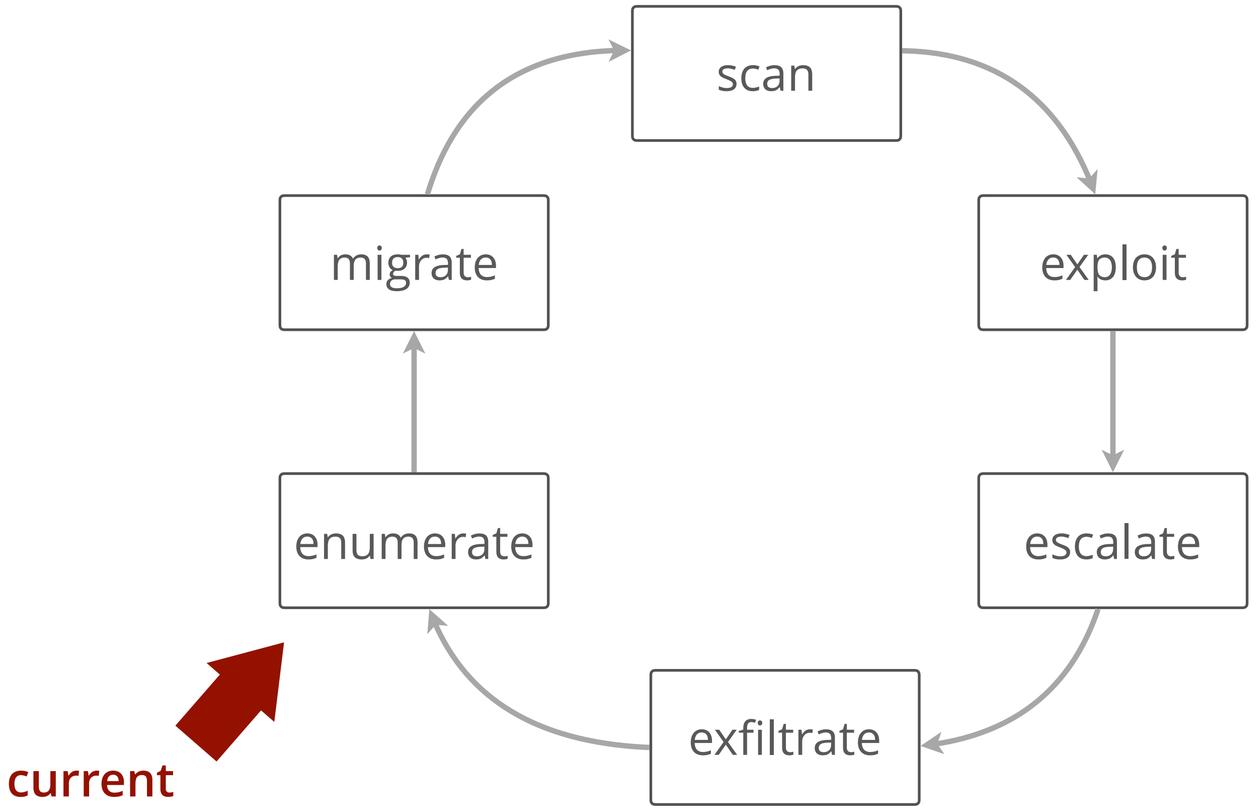
IMMEDIATE: FSM

- Finite State Machine
- Has an order of actions



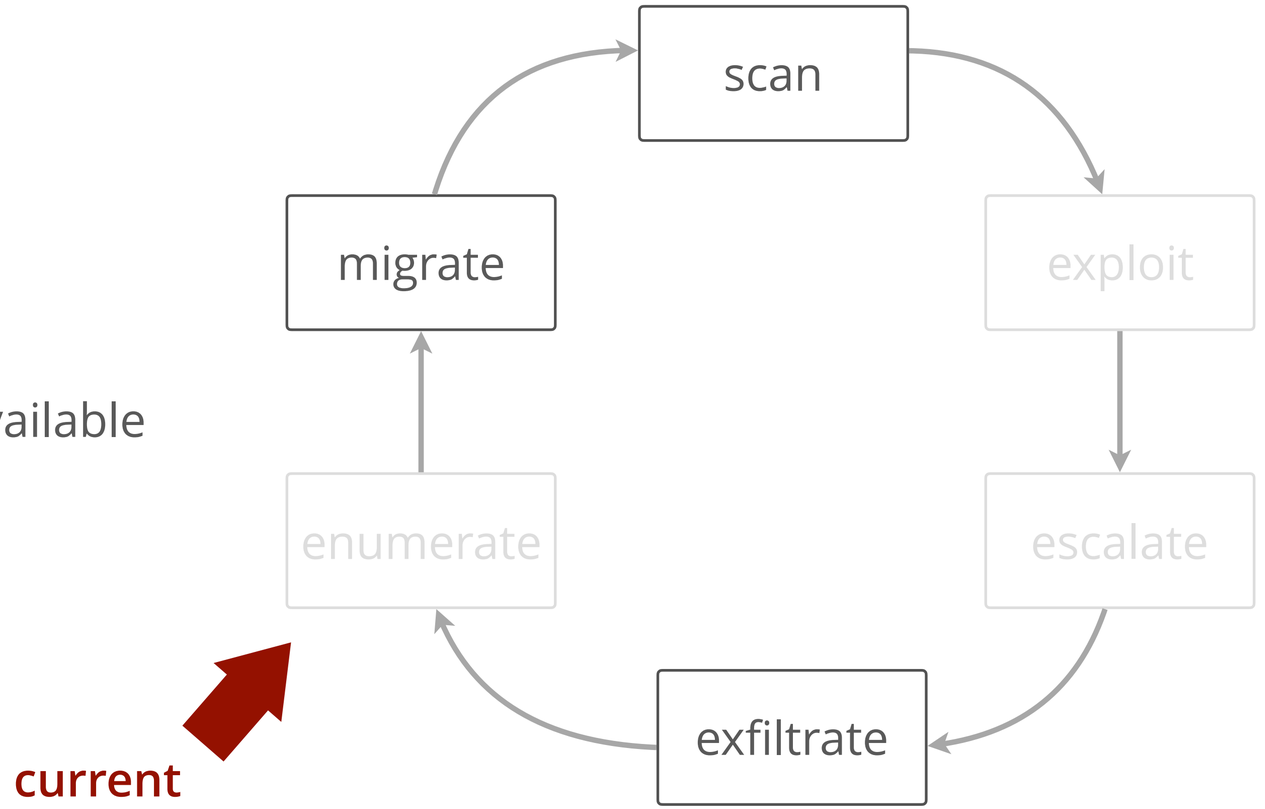
IMMEDIATE: FSM

- Finite State Machine
- Has an order of actions
- Cycles through them



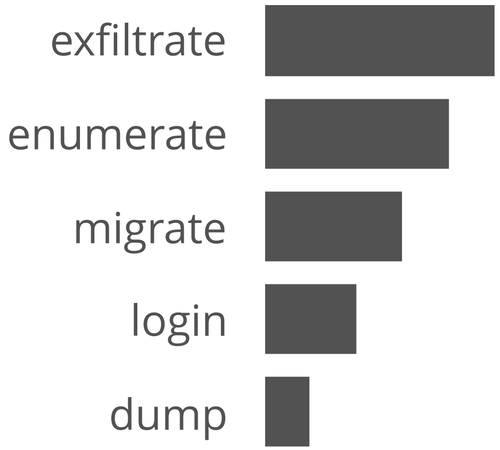
IMMEDIATE: FSM

- Finite State Machine
- Has an order of actions
- Cycles through them
- Acts randomly if next one is unavailable



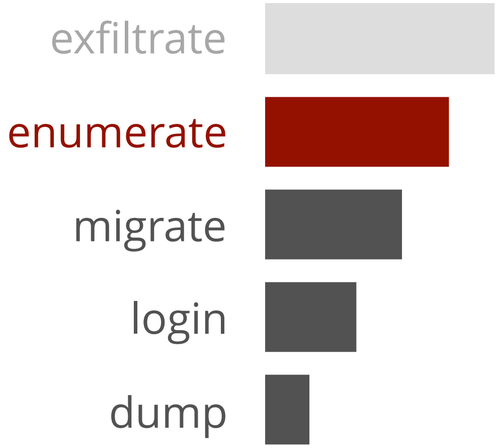
IMMEDIATE: GREEDY

- Has a list of preferences



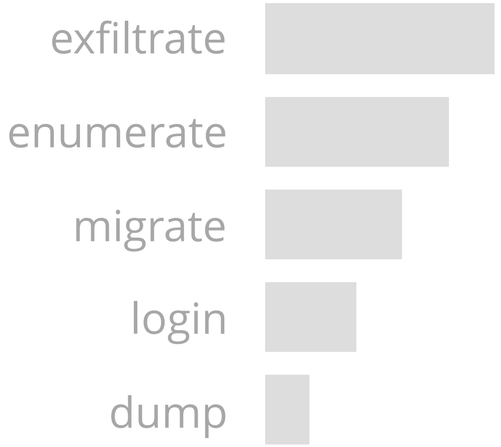
IMMEDIATE: GREEDY

- Has a list of preferences
- Always picks highest available



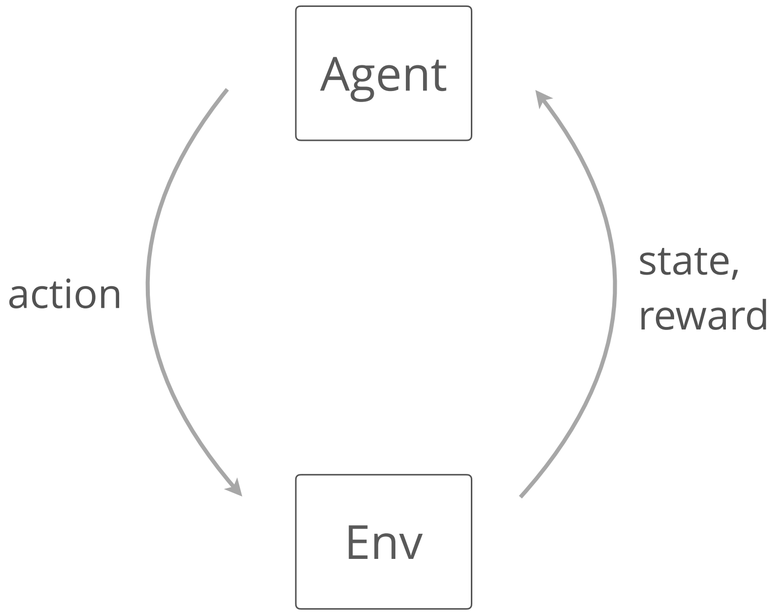
IMMEDIATE: GREEDY

- Has a list of preferences
- Always picks highest available
- Acts randomly when none available

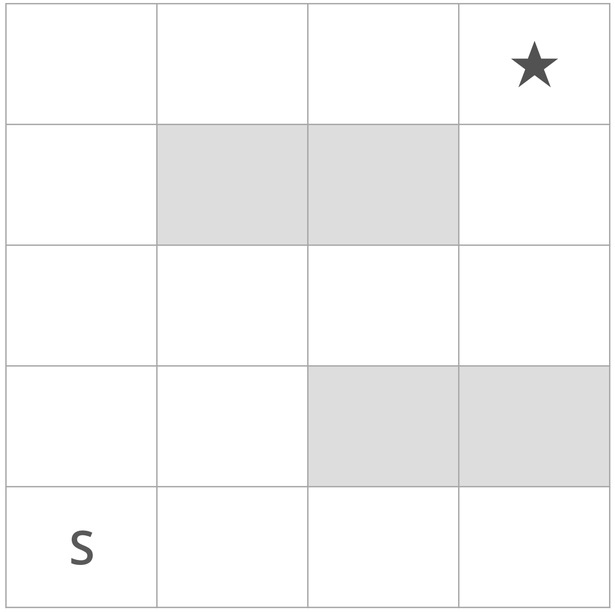


REINFORCEMENT LEARNING

- Find best strategy for:
  - maximizing cumulative reward
  - in a sequential decision-making problem
- Agent in charge of:
  - learn from environment observations
  - steer the way new experience flows in

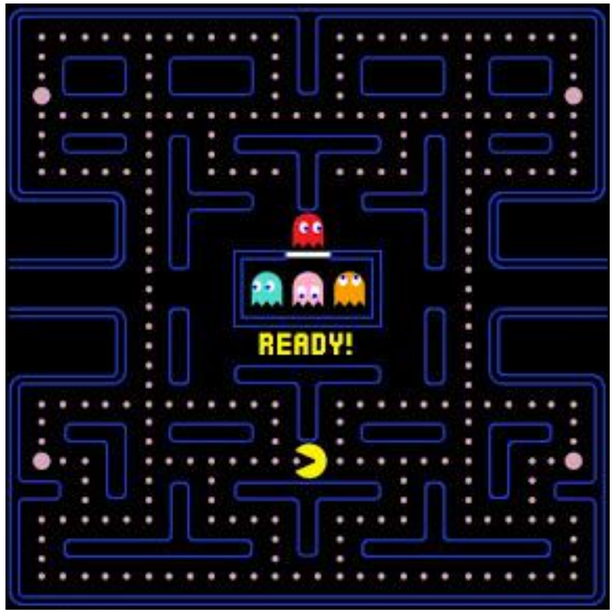


CLASSICAL SETTINGS



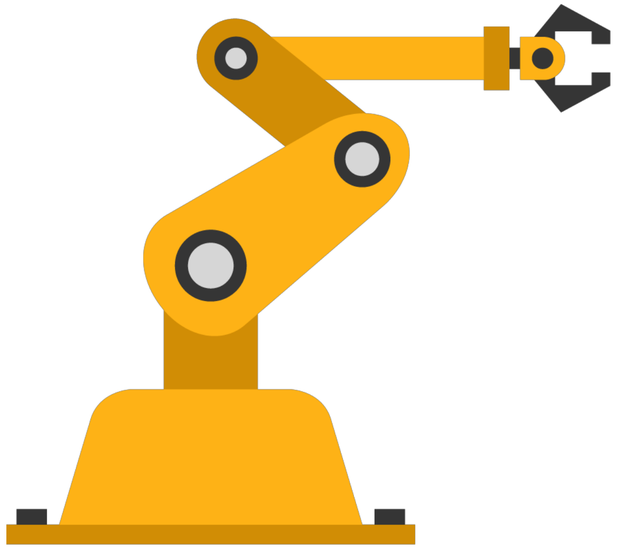
Toy: Grid-world

Chess



Modern: Pac-man

MORE SETTINGS



Robot Arm Control

Preventive Maintenance

Stock Exchange

SETTINGS FORMALIZATION

Setting	State	Actions	Rewards
Grid-world	x, y	← → ↑ ↓	+1 on goal
Chess	pieces location	move pieces	+1 for victory, –1 for defeat
Pacman	player & enemies location	← → ↑ ↓	+1 per pellet

SETTINGS FORMALIZATION

Setting	State	Actions	Rewards
Robot	x, y, z, velocity	activate motors	proximity to target
Maintenance	sensor measurements	continue / repair	–1 on accident
Stocks	companies info	buy / sell	profit

SETTINGS FORMALIZATION

Setting	State	Actions	Rewards
Pentesting	machines state	attacker actions	objective based

PENTESTING STATE

	enumer	scan	migrate	login	escalate	persist	dump	exfiltrate	cleanup	$exploit_1$	$exploit_2$	...
m_1	✓	✓			✓	o	o	o	✓	o	✓	
m_2	✓	o	o		o				o			
m_3	✓	✓			✓	✓	✓	o	o	✓	o	
m_4	o	o	✓	o	✓	o	o	✓	✓			
...												

	evade	wait	abandon
n/a	✓	o	o

✓ performed

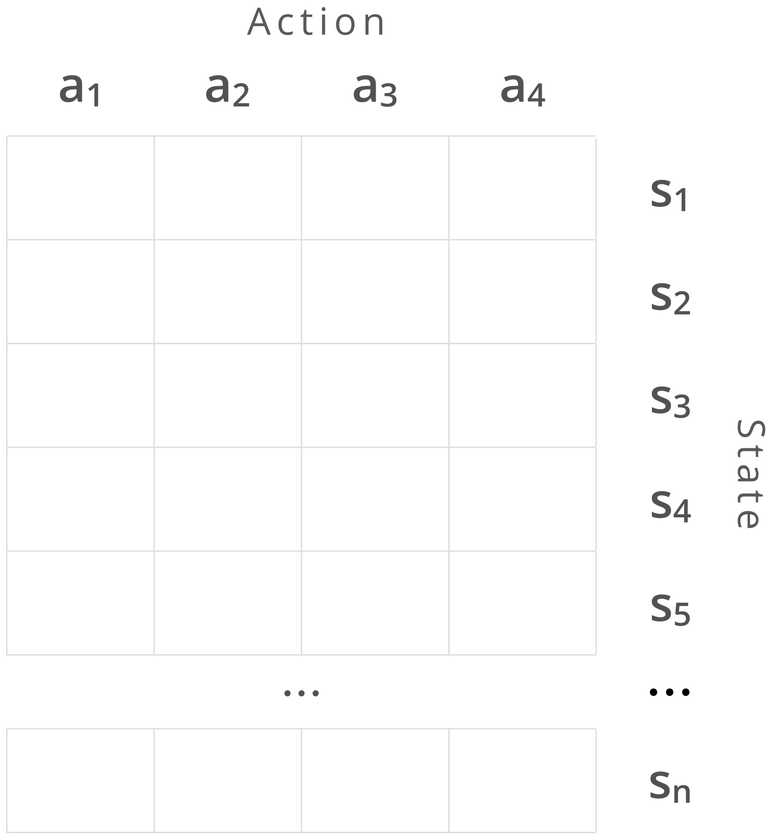
o available

RL: TABULAR

- If you know how valuable each action is, in each possible state
- Then you have solved the problem: always select most valuable action

RL: TABULAR

- Number each possible state from 0 to n
- Q table of value for each action, in each state



source: [1]

RL: TABULAR

- Number each possible state from 0 to n
- Q table of value for each action, in each state
- Initial values: arbitrary
  - zero

	Action				
	a_1	a_2	a_3	a_4	
	0	0	0	0	s_1
	0	0	0	0	s_2
	0	0	0	0	s_3
	0	0	0	0	s_4
	0	0	0	0	s_5
		...			...
	0	0	0	0	s_n

State

RL: TABULAR

- Number each possible state from 0 to n
- Q table of value for each action, in each state
- Initial values: arbitrary
  - zero
  - non-zero to incentivize exploration

	Action			
	a_1	a_2	a_3	a_4
s_1	1	0	3	0
s_2	1	1	2	2
s_3	3	3	1	1
s_4	2	3	1	3
s_5	3	1	2	3
...	...	...	...	...
s_n	0	1	3	1

State

RL: TABULAR

- Number each possible state from 0 to n
- Q table of value for each action, in each state
- Initial values: arbitrary
  - zero
  - non-zero to incentivize exploration
  - can imbue apriori knowledge

	Action			
	a_1	a_2	a_3	a_4	
s_1	0	7	0	3	
s_2	0	7	0	3	
s_3	0	7	0	3	State
s_4	0	7	0	3	
s_5	0	7	0	3	
...	...				
s_n	0	7	0	3	

RL: TABULAR — LEARN

- Based on observation s, a, r, s'
  - After taking action a in state s
  - Receiving reward r and landing on state s'
  - eg: $s_2, a_3, +8, s_4$

State	Action			
	a_1	a_2	a_3	a_4
s_1	0	7	2	2
s_2	6	0	5	4
s_3	3	3	9	6
s_4	8	3	3	4
s_5	6	2	1	0
...		...		
s_n	4	5	2	8

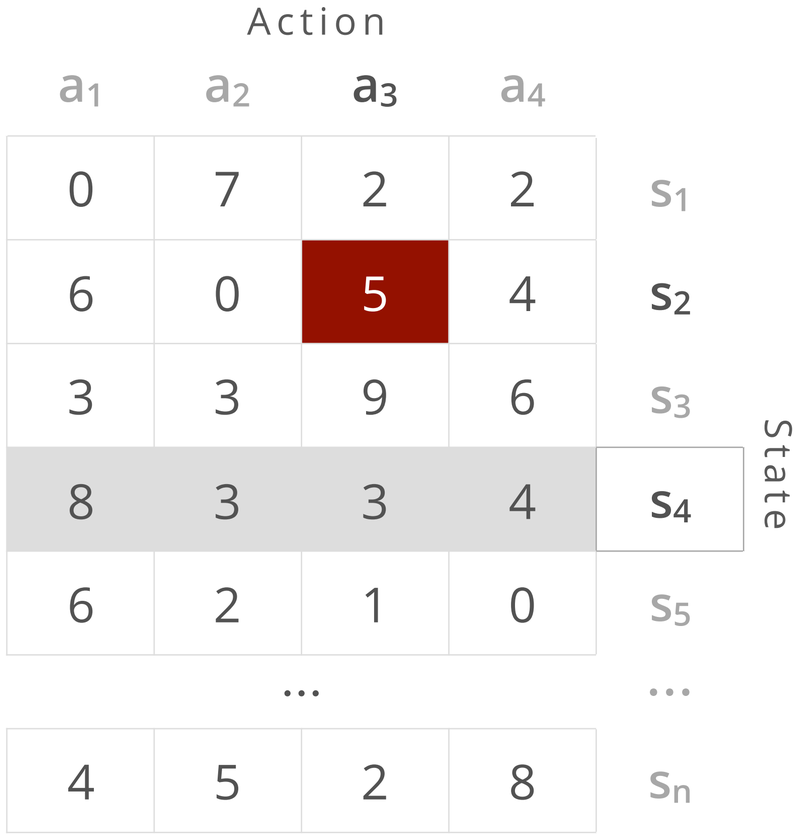
RL: TABULAR — LEARN

- Based on observation s, a, r, s'
  - After taking action a in state s
  - Receiving reward r and landing on state s'
  - eg: $s_2, a_3, +8, s_4$
- Update: $Q(s, a) \rightsquigarrow r$

State	a_1	a_2	a_3	a_4
	Action			
s_1	0	7	2	2
s_2	6	0	5	4
s_3	3	3	9	6
s_4	8	3	3	4
s_5	6	2	1	0
...		...		
s_n	4	5	2	8

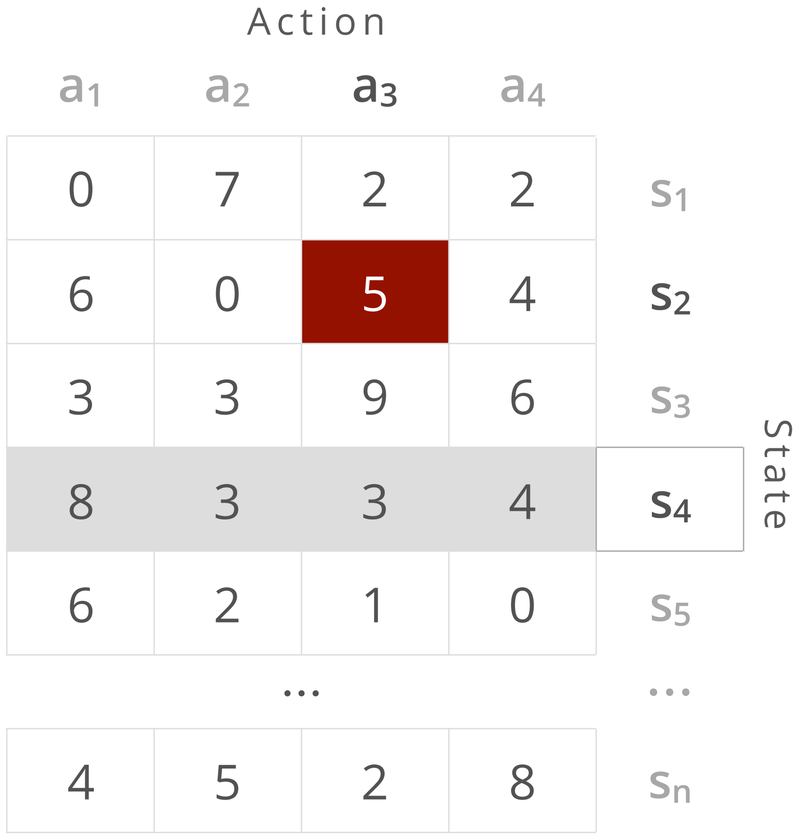
RL: TABULAR — LEARN

- Based on observation s, a, r, s'
  - After taking action a in state s
  - Receiving reward r and landing on state s'
  - eg: $s_2, a_3, +8, s_4$
- Update: $Q(s, a) \rightsquigarrow r + maxQ(s', a_i)$
- Approximate future value $Q(s')$
  - max ⇒ brave: judge itself by best case
  - avg ⇒ cautious: judge itself by most probable
  - linear combination of the two, controlled by η



RL: TABULAR — LEARN

- Based on observation s, a, r, s'
  - After taking action a in state s
  - Receiving reward r and landing on state s'
  - eg: $s_2, a_3, +8, s_4$
- Update: $Q(s, a) \rightsquigarrow r + max Q(s', a_i) \times \gamma$
- Approximate future value $Q(s')$
  - max ⇒ brave: judge itself by best case
  - avg ⇒ cautious: judge itself by most probable
- Discount future rewards by γ
  - small ⇒ visionary: care about future returns
  - large ⇒ hedonistic: prefer immediate reward



RL: TABULAR — ACT

- Based on the current state s
- Pick an action a

	Action			
	a_1	a_2	a_3	a_4	
s_1	0	7	2	2	
s_2	6	0	5	4	
s_3	3	3	9	6	
s_4	8	3	3	4	
s_5	6	2	1	0	
...			...		
s_n	4	5	2	8	

State

RL: TABULAR — ACT

- Based on the current state s
- Pick an action a
- Select the most valuable one:

$$a = argmax\ Q(s, a_i)$$

	Action			
	a_1	a_2	a_3	a_4	State
s_1	0	7	2	2	
s_2	6	0	5	4	
s_3	3	3	9	6	
s_4	8	3	3	4	
s_5	6	2	1	0	
...			...		
s_n	4	5	2	8	

RL: TABULAR — ACT

- Based on the current state s
- Pick an action a
- Select the most valuable one:

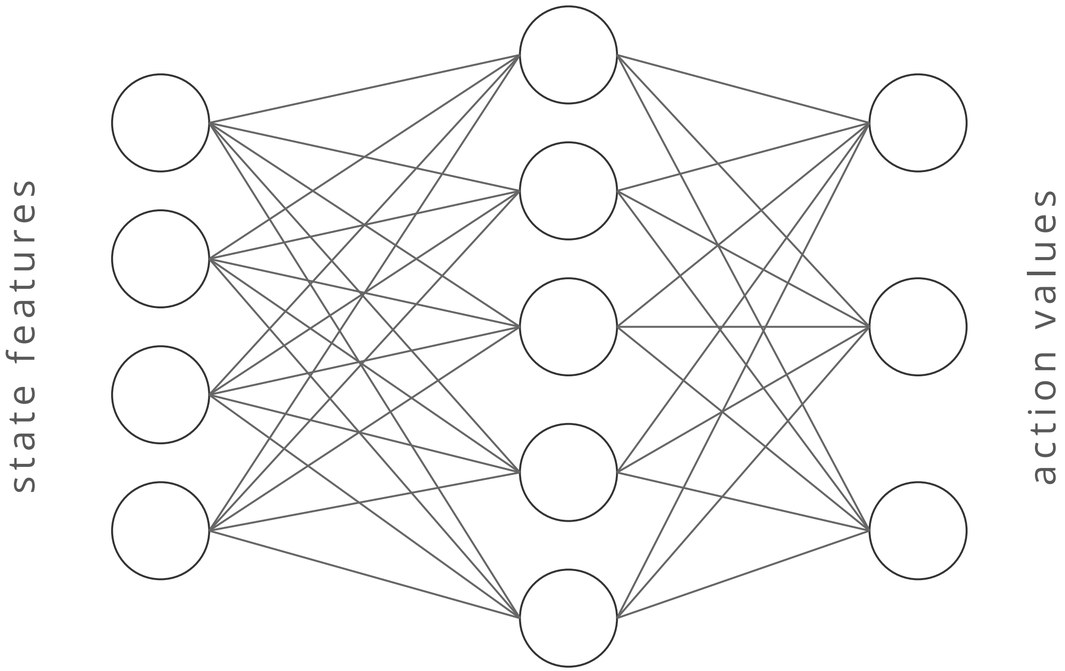
$$a = argmax\ Q(s, a_i)$$

- Sometimes, explore other options:
  - pick randomly, with probability ε
  - allows discovery of better paths

State	Action a_1	a_2	a_3	a_4
s_1	0	7	2	2
s_2	6	0	5	4
s_3	3	3	9	6
s_4	8	3	3	4
s_5	6	2	1	0
...		...		
s_n	4	5	2	8

RL: APPROXIMATE

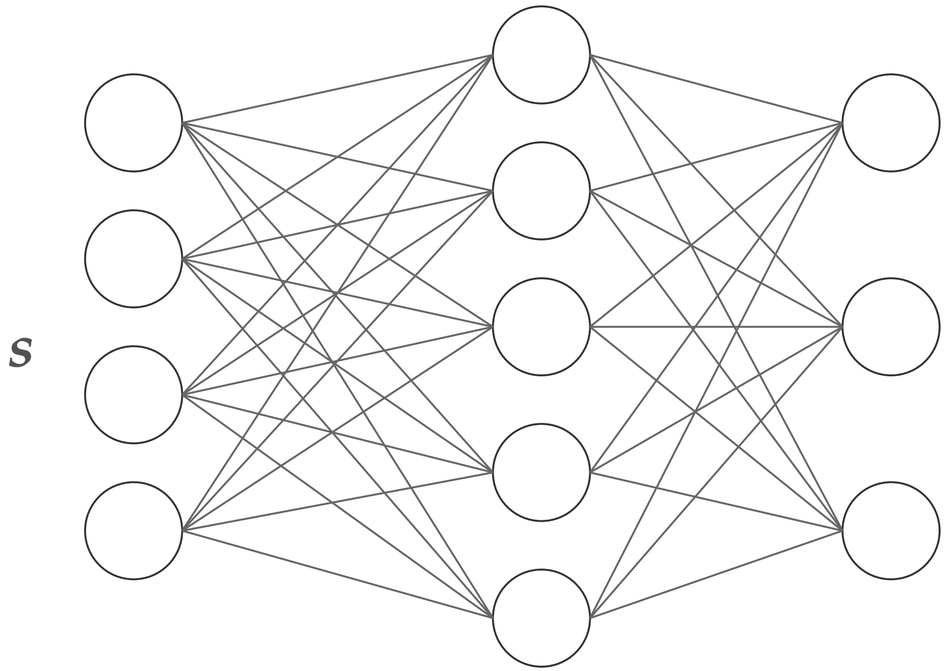
- Replace lookup table with a predictive model
- Works with state features
  - leverage state similarities
  - enables generalization



source: [1]

RL: APPROXIMATE — LEARN

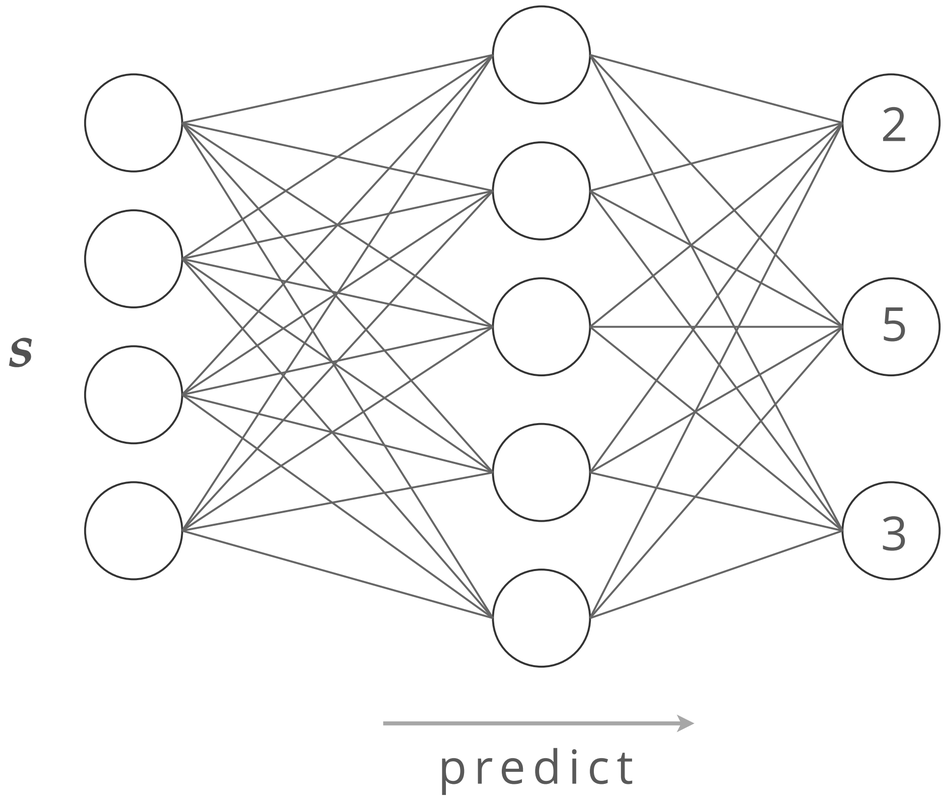
- Based on observation $\boldsymbol{s}$, a, r, s'



RL: APPROXIMATE — LEARN

- Based on observation s, a, r, s'

1. Predict value of each action $y = net(s)$



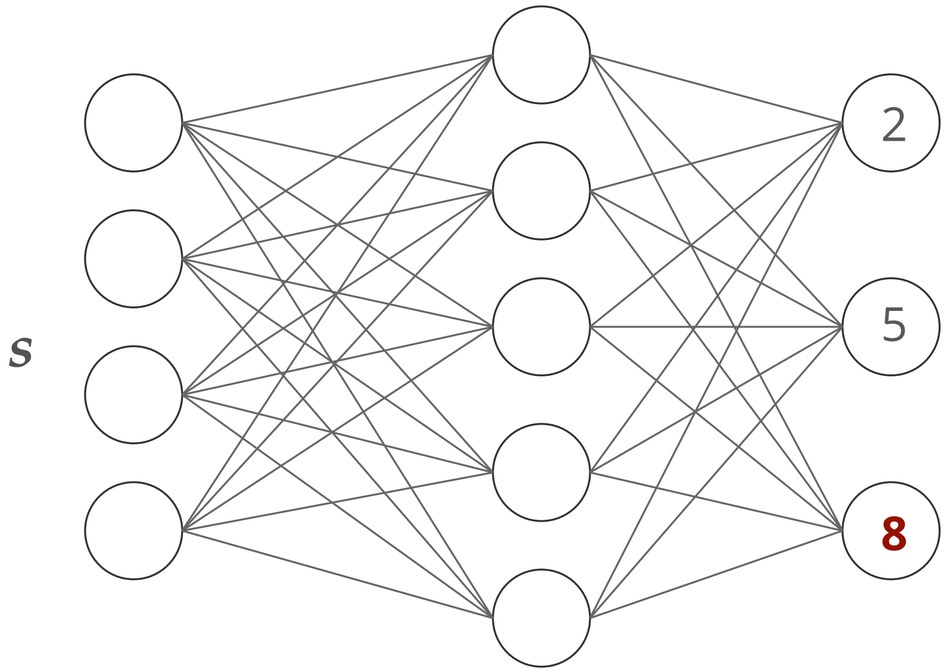
RL: APPROXIMATE — LEARN

- Based on observation s, $\boldsymbol{a}$, r, s'

1. Predict value of each action $y = net(s)$
2. Compute target g (same as Q update)

$$g = r + max\ net(s') \times \gamma$$

3. Set observed action's value $y_a = g$



update

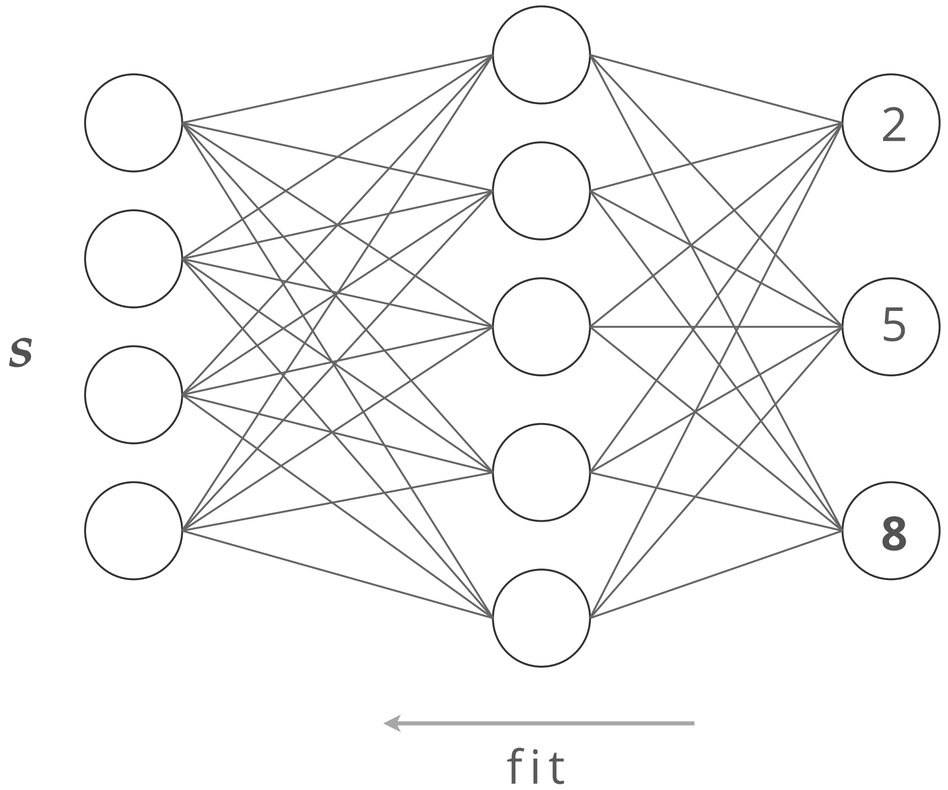
RL: APPROXIMATE — LEARN

- Based on observation s, a, r, s'

1. Predict value of each action $y = net(s)$
2. Compute target g (same as Q update)

$$g = r + max\ net(s') \times \gamma$$

3. Set observed action's value $y_a = g$
4. Train model on new pair (s, y)

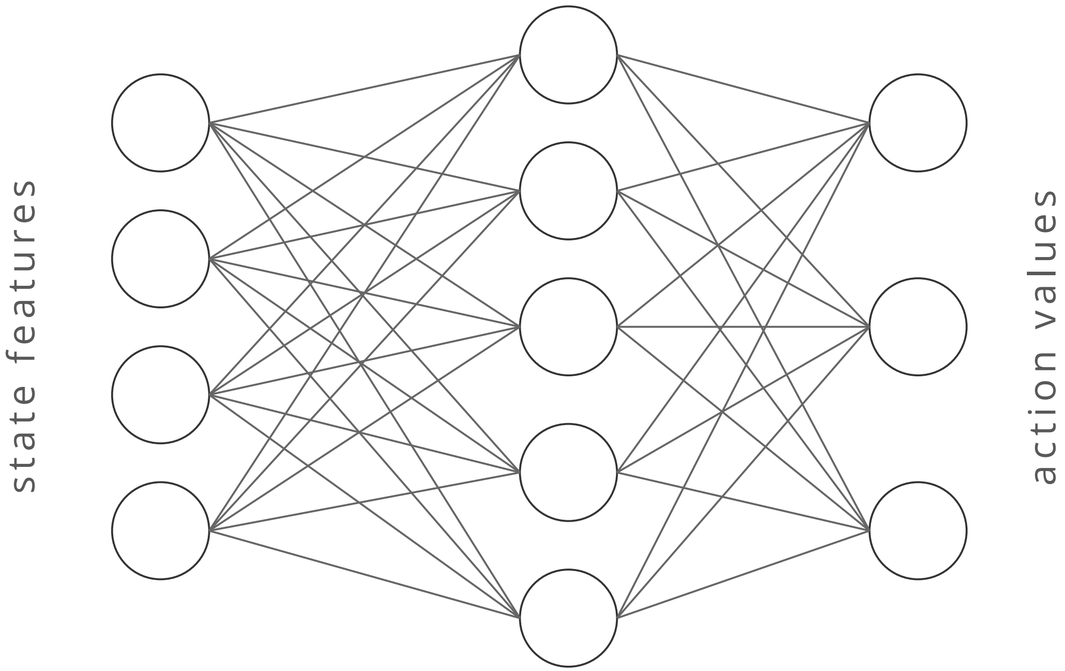


RL: APPROXIMATE — ACT

- Based on the current state s:

1. Predict value of each action $y = net(s)$
2. Pick most valuable action $a = argmax\ y$

- Sometimes (probability ε) pick randomly



RL: APPROXIMATE — SOME EXTENSIONS

- **Multi-step Returns**: look ahead multiple time steps when estimating action value
- **Double** (2015): use a second network for predictions, updated slowly towards main network
- **Dueling** (2015): decouple $Q(s, a)$ into state value $V(s)$ and action advantage $A(a)$
- **Prioritized Experience Replay** (2015): favor transitions the network can learn the most from
- **Bayesian Networks** (2015): handle uncertainty by approximating a GP using dropout
- **Bolzmann Exploration** (2017): sample actions proportional to estimated values
- **Distributional** (2017) : learn a distribution, instead of approximating a single q value
- **Noisy Nets** (2018): add parametric noise to weights
- **Asynchronicity** (2018): multiple independently interacting agents

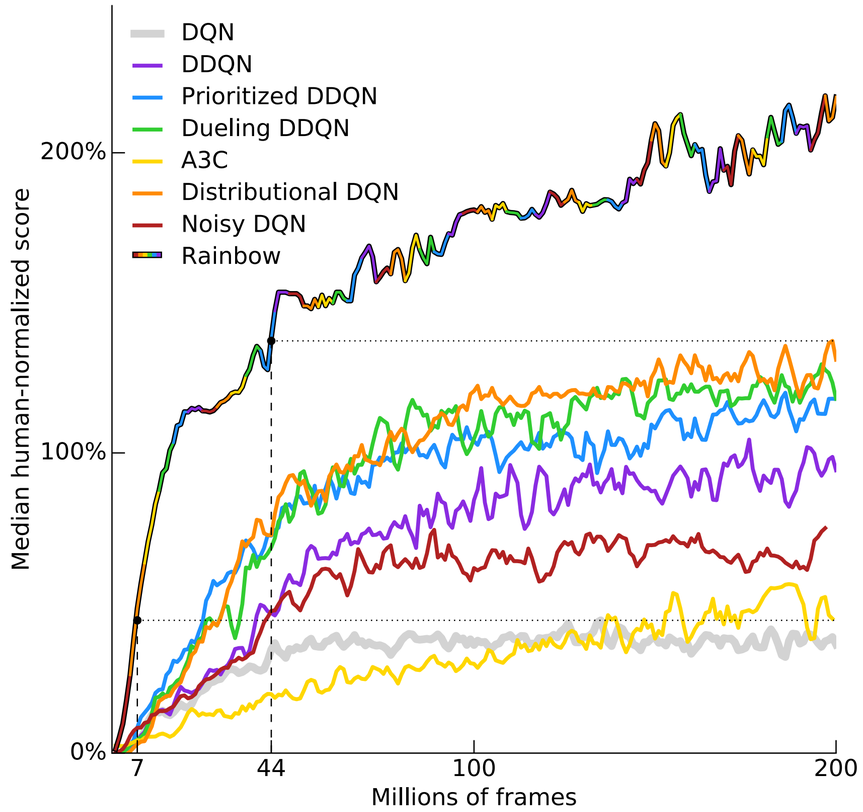
RL: APPROXIMATE — SPINOFFS

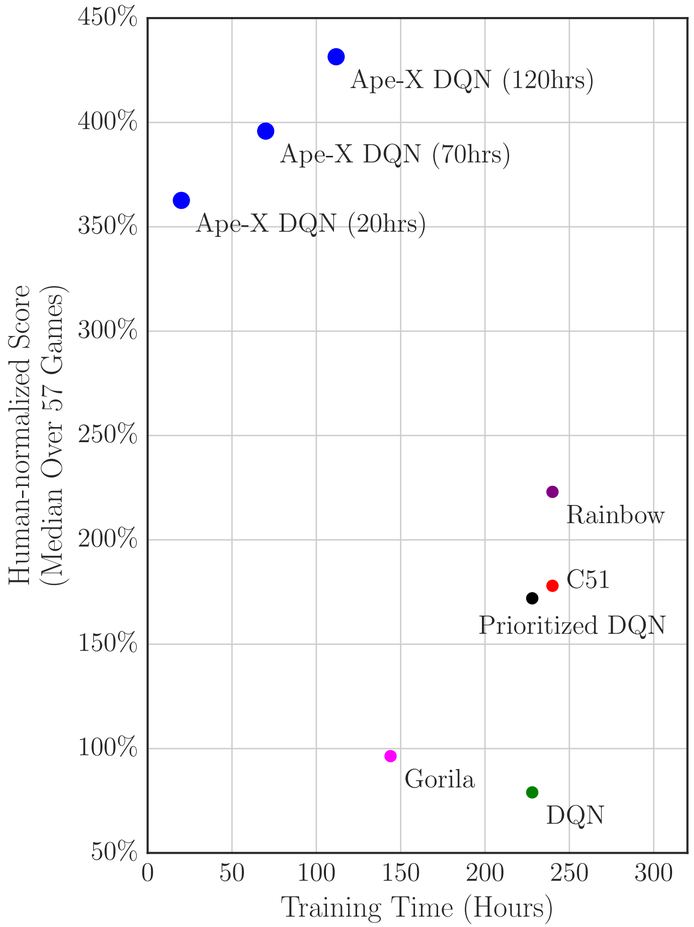
- **Policy Gradients**: learn policy directly, as a distribution over actions
  added stochasticity is essential in partially observable scenarios

- **Actor-Critic**: combine value iteration (QL) with policy iteration (policy gradient)
  also compute how much better actions turned out to be than expected

- Many more: Hierarchical RL, Recurrent DQN, Intrinsic Motivation, Trust-Region Optimization, etc

EXTENSIONS IMPACT





source: [4]

GENETIC ALGORITHMS

- Use GA as the discovery mechanism: many (guided) random individuals
- And the update mechanism: evolve fittest using genetic operators

source: [2]

GA: LINEAR CLASSIFIER SYSTEM

- Rule-Based ML

source: [7]

GA: LINEAR CLASSIFIER SYSTEM

- Rule-Based ML
- Binary encode states and actions

GA: LINEAR CLASSIFIER SYSTEM

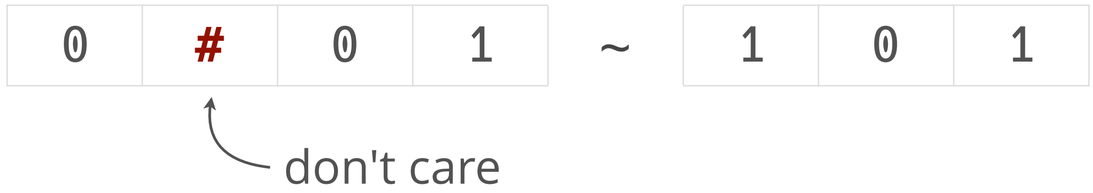
- Rule-Based ML
- Binary encode states and actions
- Creates mapping: state ~ action

GA: LINEAR CLASSIFIER SYSTEM

- Rule-Based ML
- Binary encode states and actions
- Creates mapping: state ~ action
- Simple *if-then* rules

GA: LINEAR CLASSIFIER SYSTEM

- Rule-Based ML
- Binary encode states and actions
- Creates mapping: state ~ action
- Simple *if-then* rules
- Wildcard character **#**



GA: LINEAR CLASSIFIER SYSTEM

- Individual ≔ single classifier
- Population ≔ multiple classifiers
- Solution ≔ entire population
- Rule ≔ context dependent relationship state ~ action
- Classifier ≔ a rule and its properties:
  - fitness
  - age
  - reward-prediction accuracy
  - descriptive statistics: avg/max/etc
  - numerosity

GA: LCS — LEARN

- Based on observation s, a, r, s'
  - Classifier A acted in state s
  - Classifier B acted in state s'
- Update A's fitness (same as Q update)

$$F_A \rightsquigarrow r + F_B \times \gamma$$

- Update classifier's age & statistics

State				Action		Fitness
0	1	0	~	0	0	2
#	0	#	~	1	1	4
0	1	1	~	1	1	3
0	1	0	~	0	1	3
#	1	1	~	1	0	1

GA: LCS — EVOLVE

- Initial population empty
- Apply genetic operators:
  - Highly elitist (most of population retained)
  - Binary *crossover* & *mutation*
- Cleanup
  - Subsumption: merge redundant classifiers
  - Deletion: *select* inversely proportional to fitness

GA: LCS — ACT

- Based on the current state s
  - eg: `0 1 1`

State				Action		Fitness
0	1	0	~	0	0	2
#	0	#	~	1	1	4
0	1	1	~	1	1	3
0	1	0	~	0	1	3
#	1	1	~	1	0	1

GA: LCS — ACT

- Based on the current state s
  - eg: `0 1 1`
- Find matching actions

State				Action		Fitness
0	1	0	~	0	0	2
#	0	#	~	1	1	4
0	1	1	~	1	1	3
0	1	0	~	0	1	3
#	1	1	~	1	0	1

GA: LCS — ACT

- Based on the current state s
  - eg: `0 1 1`
- Find matching actions
- *Select* classifier proportional to fitness

State				Action		Fitness
0	1	0	~	0	0	2
#	0	#	~	1	1	4
0	1	1	~	1	1	3
0	1	0	~	0	1	3
#	1	1	~	1	0	1

GA: LCS — ACT

- Based on the current state s
  - eg: `0 1 1`
- Find matching actions
- *Select* classifier proportional to fitness
- If no rule matches, cover:
  - introduce randomly generated rule

State				Action		Fitness
0	1	0	~	0	0	2
#	0	#	~	1	1	4
0	1	1	~	1	1	3
0	1	0	~	0	1	3
#	1	1	~	1	0	1

IMPORTANT CONSIDERATIONS

- Feature selection
  - immediate agents (zero features) do better than learning agents with all features
  - but learning agents overcome with a subset of features
- Exploratory starts
  - don't learn from scratch
  - have the immediate agents' experience as a starting point
- Rewards range
  - some agents deal best in the $(-1, +1)$ range

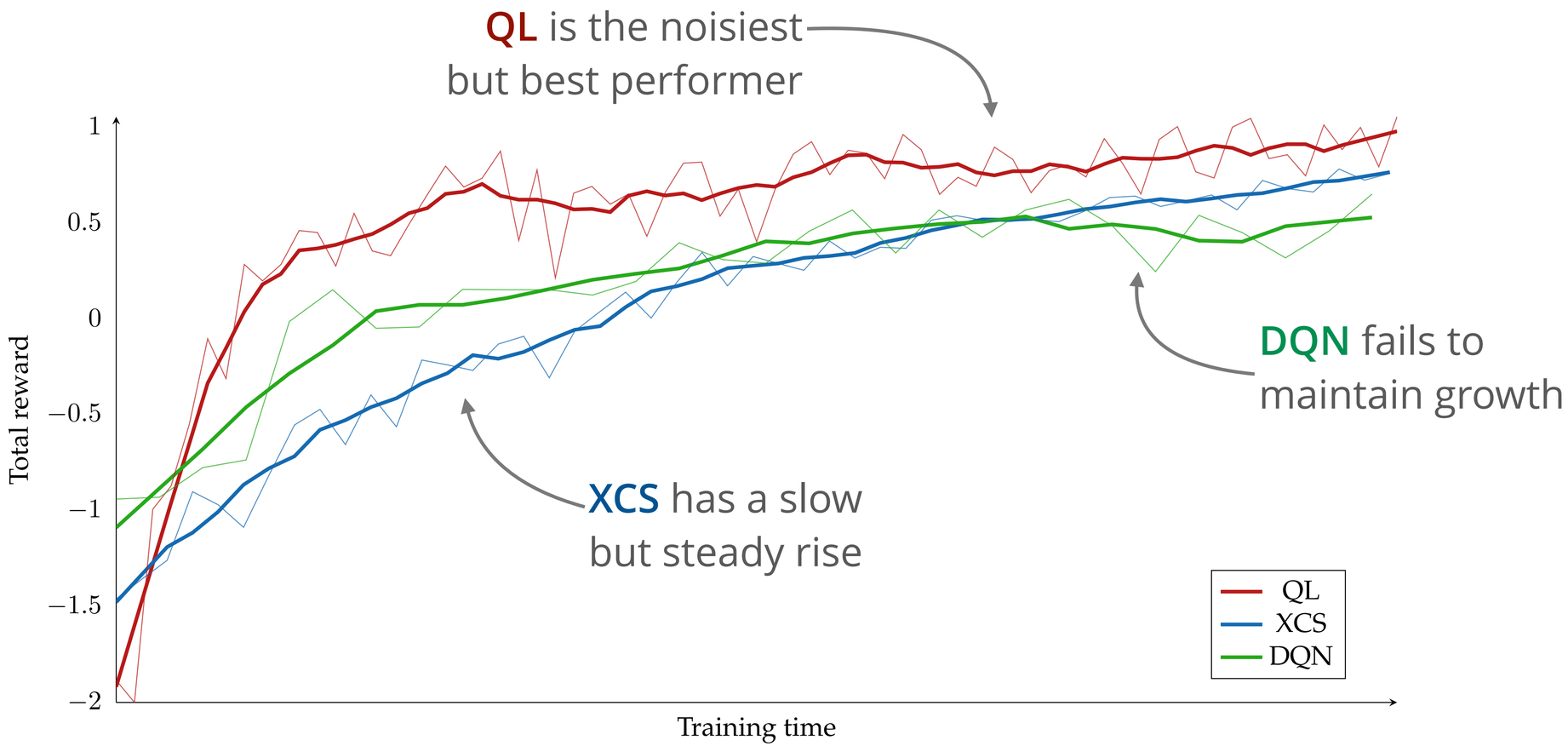
IGNORED TECHNIQUES

- Classical planning
- Perfect knowledge simulation

5

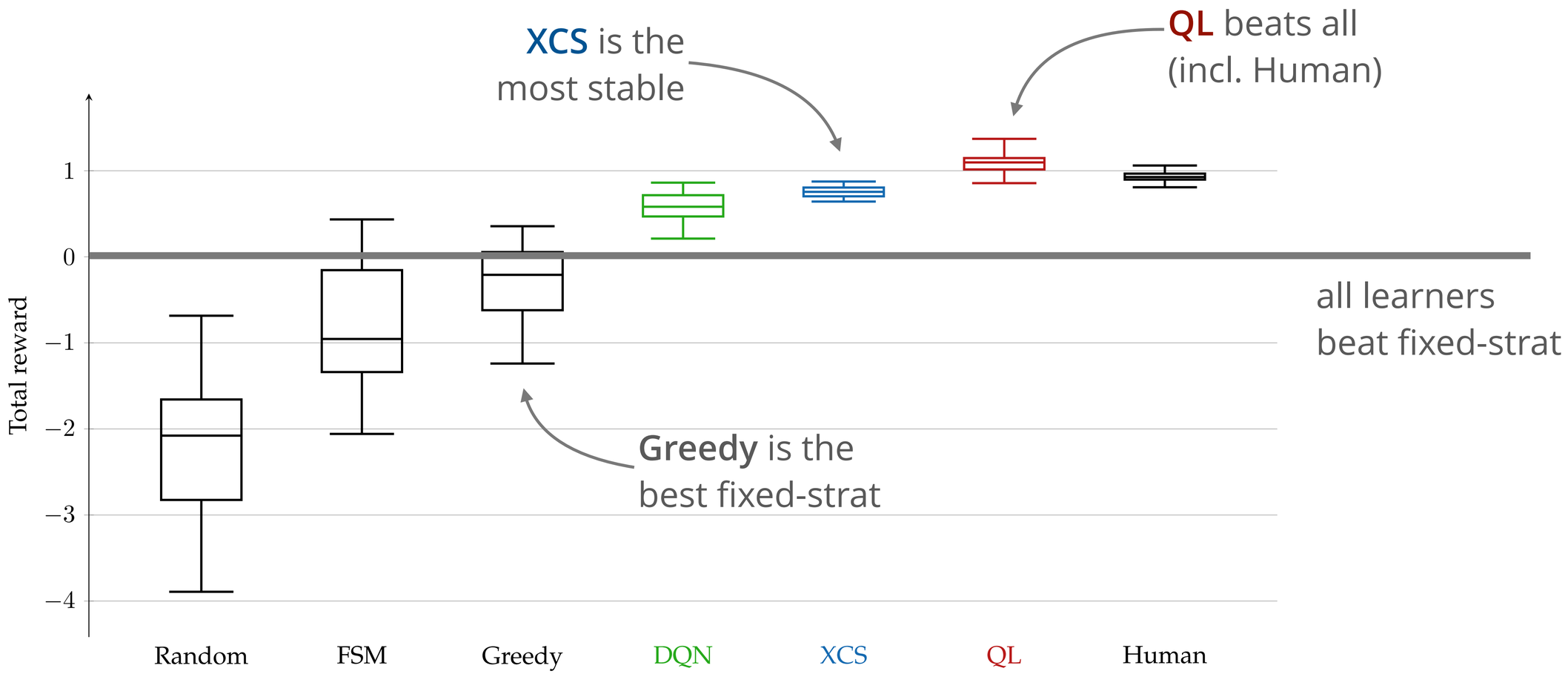
RESULTS

LEARNING BEHAVIOR

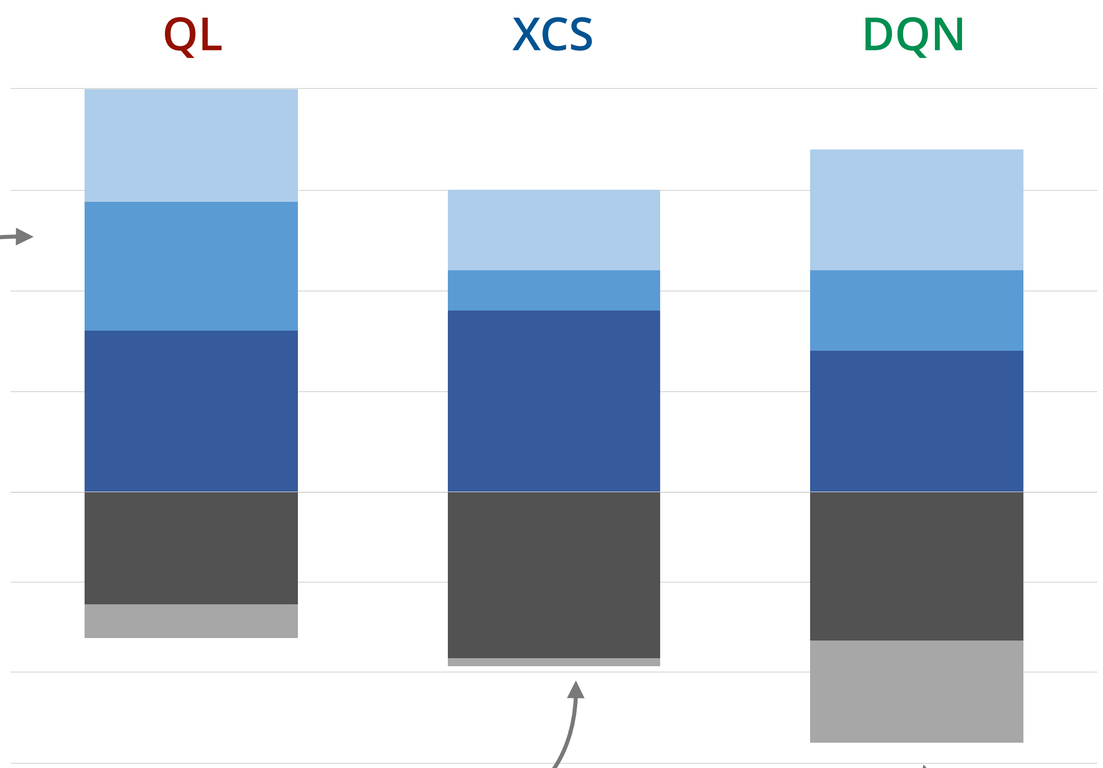


AXIS VALUES!

AGENTS PERFORMANCE



OBJECTIVES FOCUS



Positive reward:
- exfiltration
- credentials
- foothold

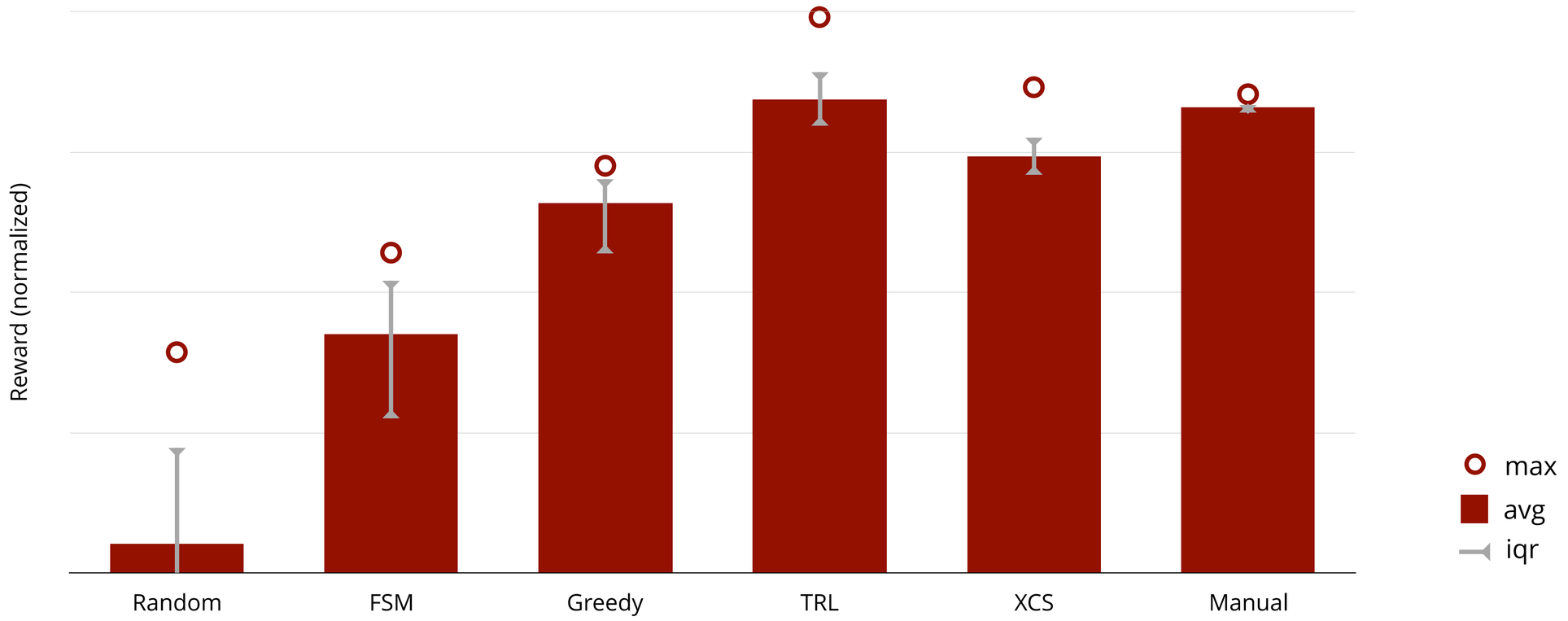
Negative reward:
- detection
- time

learned to leverage diverse sources; fast

focused only on target; careful, stealthy

gets many things right; not afraid to get caught

ALGORITHMS COMPARISON



6 CONCLUSIONS

CHALLENGES

- Game definition
  - Very few relevant papers
  - Hard to balance real-world closeness / implementation feasibility
- Algorithms fit:
  - Dissimilarity from classical RL application
  - Not evident state definition
  - Sparsity of rewards
  - Computational (in)efficiency of simulating the environment
- Algorithms training:
  - Large number of hyper-parameters (up to 34); very high training times (over 2 days)
  - No mature model-selection frameworks; not straightforward feature selection
  - Lack of generally-applicable advanced RL / efficient GA methods implementations

CONCLUSIONS

- Learning-based methods surpass fixed-strategy ones
- QL manages to overcome human performance
- Diverse strategies could be valuable in emulating various attacker types:
  - QL is fast and makes use of many techniques
  - XCS is methodical and stealthy
  - DQN is effective but also reckless
- Introduced hyper-parameter η proved useful for all algorithms, balancing caution / aggression

CONFERENCE CRITIQUE

Venue: ACM Computer and Communications Security 2018 (A* CORE ranking)

+ from real world data (Metasploit, NVD)
+ RL could be very impactful on attack design
+ easy to understand for non-expert RL
+ good insight on framework design
+ comparison shows advantages of RL agent

– simplistic pentest model
– incl. no social engineering
– model hard to scale to large networks
– no in-depth analysis of RL
– no actionable security advice

Bottom line: better suited for a RL rather than a security venue

FUTURE WORK

ENVIRONMENT — LONGER TIMESPANS

- Add missing *ATT&CK* actions:
  - collection: key-logger, webcam, etc
  - command & control: periodically communicate externally (in/out)
- Can measure actions impact in the long run
- Some modern attacks focus on of stealth, long-term infiltrations

ENVIRONMENT — LARGER NETWORKS

- Leverage the clustering nature of enterprise networks
- Generalize same approach to a larger environment
- Employ abstraction layers:
    1. choose a connected component
    2. choose a machine
    3. choose an action

DEFENDER — MORE SOPHISTICATED

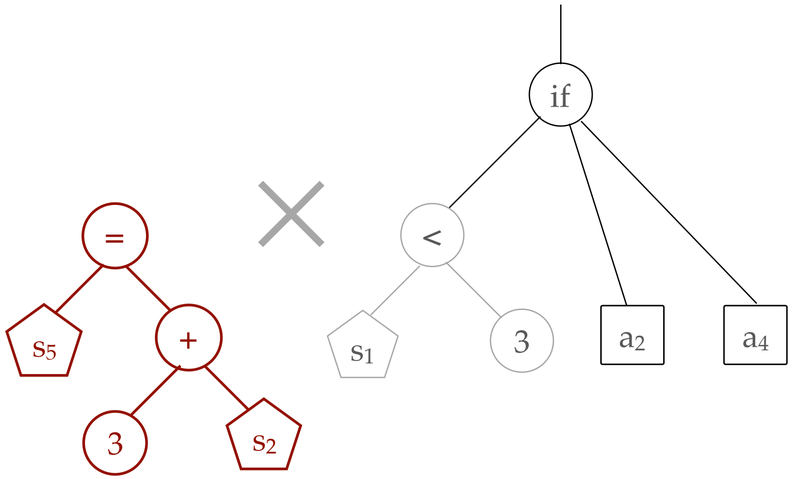
- Stackelberg Equilibrium
  - from Game Theory, Economics
  - protect those places that are valuable & often targeted
  - real-life eg: allocate policemen to airport gates
- Other defense strategies
- Measure agents performance against multiple levels of defender strength

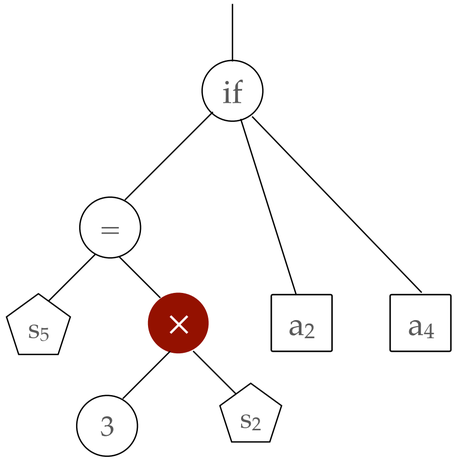
ATTACKER — PRESS FURTHER

- Finer DQN hyper-parameter tuning
- Other DQN flavors
- Let them run for longer

ATTACKER — ADVANCED GA METHOD

- Evolve data structures which are executable programs
- Generalized Decision Trees
- Nodes operations:
  - branching: *if-then*
  - arithmetic: +, -, ×, /
  - comparison: >, ≤, ≠
  - logical: *and, or, not*
- Input (root): state features
- Output (leaves): action to take





APPLY TO REAL WORLD

- Map actions to actual commands
- They are already loosely based on *Metasploit* (one of the most popular pentest tools)
- Run and benchmark against real anti-virus solutions

Thank you,

QUESTIONS?

SELECTED BIBLIOGRAPHY

- Techniques fundaments
    1. Reinforcement Learning: An Introduction, Sutton & Barto (1998)
    2. Handbook of Evolutionary Computation, Fogel & Bäck (1997)
    3. Towards a Science of Security Games, USC (2016)
    4. Rainbow: Combining Improvements in Deep Reinforcement Learning, DeepMind (2017)
- Problem formalization
    5. Adversarial Reinforcement Learning in a Cyber Security Simulation, U Groningen (2017)
    6. Analysis of Automated Adversary Emulation Techniques, MITRE (2017)
    7. POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing, Core (2012)