Journal of Intelligent Information Systems
https://doi.org/10.1007/s10844-022-00738-0

Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks

Mohamed C. Ghanem[1] · **Thomas M. Chen**[1] · **Erivelton G. Nepomuceno**[2]

Received: 23 May 2022 / Revised: 14 August 2022 / Accepted: 22 August 2022

Abstract
Penetration testing (PT) is a method for assessing and evaluating the security of digital assets by planning, generating, and executing possible attacks that aim to discover and exploit vulnerabilities. In large networks, penetration testing becomes repetitive, complex and resource consuming despite the use of automated tools. This paper investigates reinforcement learning (RL) to make penetration testing more intelligent, targeted, and efficient. The proposed approach called Intelligent Automated Penetration Testing Framework (IAPTF) utilizes model-based RL to automate sequential decision making. Penetration testing tasks are treated as a partially observed Markov decision process (POMDP) which is solved with an external POMDP-solver using different algorithms to identify the most efficient options. A major difficulty encountered was solving large POMDPs resulting from large networks. This was overcome by representing networks hierarchically as a group of clusters and treating each cluster separately. This approach is tested through simulations of networks of various sizes. The results show that IAPTF with hierarchical network modeling outperforms previous approaches as well as human performance in terms of time, number of tested vectors and accuracy, and the advantage increases with the network size. Another advantage of IAPTF is the ease of repetition for retesting similar networks, which is often encountered in real PT. The results suggest that IAPTF is a promising approach to offload work from and ultimately replace human pen testing.

Keywords Penetration testing · Artificial intelligence · Machine learning · Reinforcement learning · Hierarchical reinforcement learning · Markov decision process · Vulnerability assessment

✉ Mohamed C. Ghanem
mohamed.ghanem@city.ac.uk

Thomas M. Chen
tom.chen.1@city.ac.uk

Erivelton G. Nepomuceno
erivelton.nepomuceno@mu.ie

[1] Department of Electrical and Electronic Engineering, City, University of London, Street, London EC1V 0HB, UK

[2] Department of Electronic Engineering, Maynooth University, Street, Maynooth W23 F2K8, Ireland

Published online: 12 September 2022

1 Introduction

Modern networks are generally becoming larger and more complex due to increasingly sophisticated applications. At the same time, cyber security threats are more frequent and commonplace, prompting security professionals to add more security layers and policies (He & Bode, 2006). Unfortunately, networks continue to have vulnerabilities due to human errors, misconfigurations, and systems weaknesses (Bacudio et al., 2011). Penetration testing (PT) is a proactive approach to protect networks by identifying their exploitable vulnerabilities. PT is a central, often mandatory component of cyber security audits. It constitutes all standard auditing and testing tasks starting from information gathering and analysis, planning, identifying vulnerabilities, and executing relevant exploits (Backes et al., 2017).

In the beginning, most PT processes were done manually with tests run against a limited number of hosts, which allowed manual PT to be effective. Then the proliferation of computer networks forced the automation of PT tools to cover more ground in a short time (Phong & Yan, 2014). Nonetheless, automation was not enough in light of highly technological organizations with hundreds or more IP addresses and increasingly complex applications and virtualization; PT experts found it difficult to assess the security of every component in a reasonable time. To meet this challenge, machine learning (ML) has become indispensable to further improve efficiency, accuracy and coverage. In addition, PT is a repetitive process where organizations perform the testing periodically or whenever a major network upgrade occurs (e.g., new infrastructure added) (Abu-Dabaseh & Alshammari, 2018). Intelligent PT tools capable of learning from experience are well suited for repetitive similar tasks.

Metasploit, Core Impact, Nessus and other tools that come with Kali and Parrot paved the way for more automated PT but always required a portfolio of scripts and tools commanded and orchestrated by an experienced human tester. Core Impact was the first automated vulnerability scanner that initiated the use of AI leading to popularity of this tool with early stage testers. To achieve more efficiency, other research has focused on using AI and ML for penetration testing which can be more efficient and more effective, saving time and resources compared to manual testing (Abu-Dabaseh & Alshammari, 2018).

This paper focuses on reinforcement learning (RL), an artificial intelligence technique that enables a system to learn and adapt without following explicit instructions but instead by interacting with its environment (Yaqoob et al., 2017). The introduction of RL in pen testing, especially with current size and complexity of network infrastructure, is considered as a valuable addition to PT practice. In fact, a higher number of online services mean a larger exposition surface with attacks that can range in scale from massive state-sponsored attacks to simple attacks on individuals and small businesses in the hopes of gaining credentials or financial details (Ghanem & Chen, 2019).

In our experiments with embedding RL into PT tools, notably Metasploit, a scalability problem was encountered arising from exponential growth in computational power demand as the size of the RL environment increased (Ghanem & Chen, 2019). Moderate and large size networks led to very large POMDP environments in terms of number of states, transitions and observations. After considering different approaches, this paper proposes to tackle the scalability problem using hierarchical reinforcement learning (HRL) which reduces dimensionality by decomposing the RL problem into several sub-problems.

In this paper, Section 2 reviews the current challenges in automating PT. Section 3 presents RL principles in the context of PT. In Section 4, HRL is proposed and

described for automating PT. Section 5 describes the new IAPTF framework. Solution of POMDP is discussed in Section 6. Experimental results are given in Section 7.

2 Research background

In this section we will briefly review PT and current challenges in its regular automation (also called blind), highlighting the existing tools and systems to address the efficiency issues in medium and large LANs. This section will establish the motivation behind our research which aims to address the scalability problem experienced in our previous paper (Ghanem & Chen, 2019).

2.1 Penetration testing

PT has been a cornerstone in cyber security practice during the last decade. It implies planning and performing a real and controlled attack on a digital asset (machine, Web server, software or network) with the aim of evaluating its security, PT is becoming as key methods employed by organizations for strengthening their defenses against cyber threats (Sarraute et al., 2012). The process of PT is often divided into a sequence of tasks in order to methodically and comprehensively assess the security of the system and often include actively identifying vulnerabilities and perform a set of actions to test if the target could be compromised by running exploits against those vulnerabilities (Yaqoob et al., 2017).

In practice, PT tasks shown in Fig. 1 vary from case to case but generally start with an information gathering phase, where the expert explores the web using OSINT (open source intelligence) techniques to gather information about the target system. After completing the first phase, the PT expert will engage in a discovery phase to actively harvest security related information such as the system architecture, configurations and security measures in place. Next the PT expert will analyze the gathered information to look for potential vulnerabilities for exploitation in order to gain control (full or partial) or escalate the privilege held. Lastly, the newly gained access will be used to penetrate further, repeating this process until the desired target is achieved or no more attack vectors are possible. The findings of PT are reported along with expert recommendations.

A key element about PT practice is it remains non-standard in term of methods and approaches and also involves the use of a versatile tools, systems and frameworks to accomplish different activities and tasks. For instance, the information gathering phase

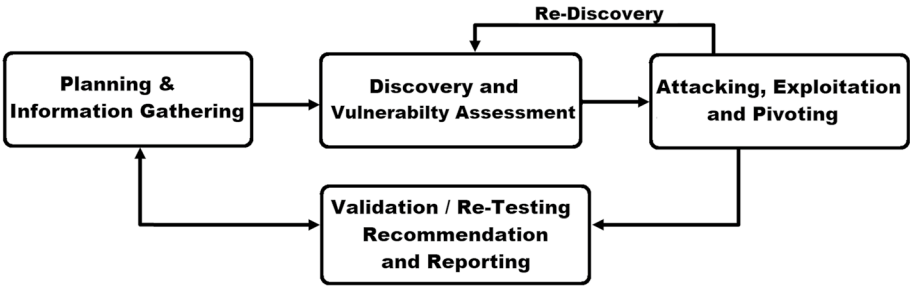

Fig. 1 Simplified network PT workflow and activities

typically involves utilizing tools such as traffic monitoring, port scanning and OS fingerprinting in order to gather relevant information that can be used to dress target system security profile and therefore determine if it contains a vulnerability that can be exploited. On the other hand, the exploitation phase requires the use of different framework and eventually customized payload modules and scripts in order to execute the relevant exploits aiming to take advantage of the identified vulnerabilities with the aim of compromising the target or gaining additional privilege or control over it. The post-exploitation tasks also are not standardized and additional tools such as rootkits are used to maintain the breach and work toward further penetration also called pivoting. Finally, PT also involves testing scenarios and attack vectors that differ per asset (Ghanem & Chen, 2019).

2.2 PT automation and optimization

The first attempt to address PT automation was attack planning as part of the AI Planning in Cyber Security domain (Boddy et al., 2005). Independently, the approach was put forward by Core Security researchers (Sarraute et al., 2013) and implemented in their PT tool Core Impact. Their approach captures most security variables and solves scenarios based on the probability of success of actions, execution time, and generated traffic.

In related work, (Backes et al., 2017) attempted to automate PT by mitigating vulnerabilities and countermeasures by conducting comprehensive what-if analyses. A conceptual framework is provided to reason about mitigation actions applied to a network model. The approach determines optimal combinations that minimize attacker success following a holistic mitigation strategy (Backes et al., 2017).

In a recent work, (Zennaro & Erdodi, 2020) attempted to apply RL to solve capture the flag (CTF) scenarios. Fundamentally, CTF competitions are very specific scenarios which do not account for many variables in typical PT, but the study was significant for investigating the relevance of different RL techniques.

A noteworthy proposal aimed to automate exploitation and post-exploitation by combining deep RL and the PowerShell empire post-exploitation framework (Maeda & Mimura, 2021). RL agents pick a PowerShell module and use its internal features as action states and then compare the learning progress of three RL models; A2C, Q-Learning, and SARSA. The results showed that A2C is the most efficient and trained agent can eventually obtain the admin privileges of the domain controller system.

2.3 Research motivations and contribution

During the last decade, several researchers attempted to tackle PT automation and optimization using AI techniques. In our previous research (Ghanem & Chen, 2019) we proposed the first versatile, fully automated and intelligent framework for network PT called IAPTF (Intelligent Automated Penetration Testing Framework). However, the approach used in modelling the PT domain as RL problem for medium and large networks resulted into enormous POMDP environments that were difficult to solve and we experienced the well-known curse of dimensionality problem (Sarraute et al., 2012). Furthermore, the previous work faced two major challenges:

- *Exploitation* the re-use of the knowledge already learned in the future to determine automatically the best action in a given state.

- *Exploration* the number of explored attack vectors resulting from continuous policies improve in parallel with agent exploring new states.

The current work aims to overcome the aforementioned challenges. For the scalability issue, we propose an innovative hierarchical RL representation which is embedded within IAPTF and tackles large networks efficiently and effectively by dividing large networks into clusters following a security-oriented logic, and then tackling each cluster separately as small network at first stage, then the network of head of clusters. In terms of capturing expertise for reuse, we propose a novel knowledge extraction, generalization using scripts and an expert system which is dedicated to expertise processing, storing and direct future re-use (Bertoglio & Zorzo, 2017), while the use of hierarchical RL enabled IAPTF to achieve an accepted balance between efficiency and exploration notably by guarantee (Al-Emran, 2015).

3 Reinforcement learning

RL provides a conceptual framework to address a fundamental problem in artificial intelligence, the development of situated agents that learn how to behave while interacting with the environment. RL is increasingly seen as a general framework for learning in a human way in both model-free and model-based approaches (Bacudio et al., 2011). In this section, RL fundamentals will be reviewed, as well as POMDP modeling and solving.

3.1 POMDP fundamentals

RL came from the notion that a human expert can be assisted or ultimately replaced by automated systems. RL provides a conceptual framework to address a fundamental problem in Ai which is how situated agents can learn how to behave by interacting with the environment. In RL, this problem is formulated as an agent-centric optimization in which the goal is to select actions to gain as much reward as possible in the long run (Yaqoob et al., 2017; Ghanem & Chen, 2019). An overview of RL is shown in Fig. 2.

Sequential decision making is usually formalized as a Markov decision process (MDP) when fully observed with no uncertainties, or otherwise as a partially observed Markov decision process. There are two basic approaches to solve MDP and POMDP which are planning and reinforcement learning.

A partially observable Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, O, \mathscr{B})$ where $\mathcal{S}$ is the set of possible states; $\mathcal{A}$ is the set of possible actions; $\mathcal{T}$ is the set of possible transitions; $\mathcal{R} : S|\mathcal{A}$ is the reward function; $\mathcal{O}$ is the set of possible observations; $\mathscr{B}$ is the belief distribution calculated from Prob $(\mathcal{O} \mid \mathcal{S})$; γ is the discount factor with preset value in the [0, 1] domain; and π is the set of learned policies π_i. Belief state space denoted as $\mathscr{B}$ is a $|S|-1$ dimensional simplex whose elements are probability distributions over states. T is a state transition and observation probability matrix, defined by the following equations:

$$T^{a}_{ss'} = P[S_{t+1} = s' | S(t) = s, A(t) = a] \tag{1}$$

$$O^{a}_{so'} = P[S_{t+1} = o' | S(t) = s, A(t) = a] \tag{2}$$

R is a reward function, defined by the equation:

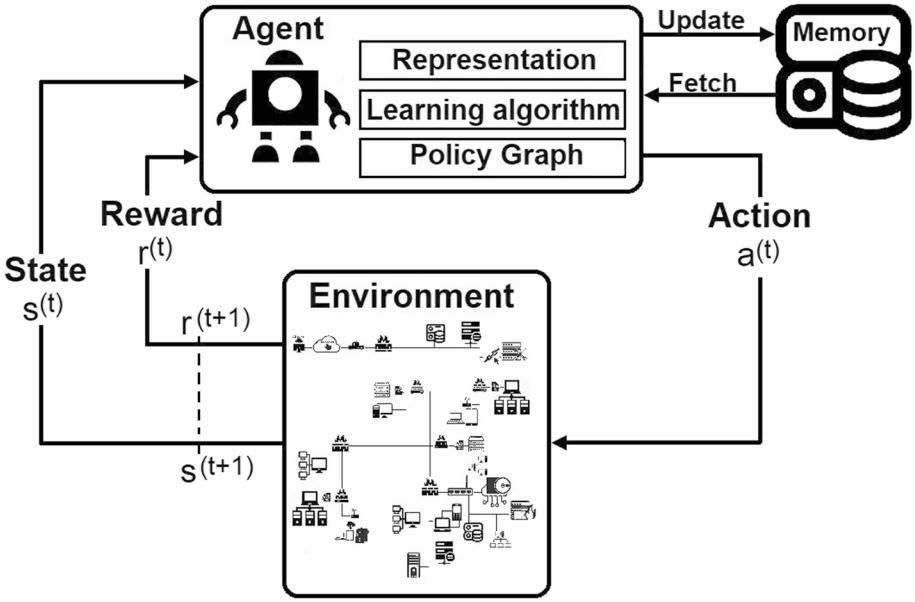

Fig. 2 The POMDP conceptual framework in context of PT (Moerland et al., 2020)

$$R_s^a = \sum_{t=0}^{t=n} [R(t+1)|S(t) = s, A(t) = a] \tag{3}$$

The overall reward G_t and state value functions are:

$$G_t = \sum_{k=0}^{k=m} \gamma^k R_{t+k+1} \tag{4}$$

$$V_\pi(s) = \sum_{\pi} [G(t)|S(t) = s] \tag{5}$$

and the action value function $Q_\pi(s,a)$ is:

$$Q_\pi(s, a) = \sum_{\pi} [G(t)|S(t) = s, A(t) = a] \tag{6}$$

Policy function is defined as:

$$\Pi(a|s) = \sum_{\pi} [A(t) = a|S(t) = s] \tag{7}$$

Belief probabilities are calculated as:

$$b \in \mathcal{B}|b(S) = Prob(S|currentstate) \tag{8}$$

Transition model describes transitions between beliefs, rather than transitions between states, where b_a^o is the belief state reached when starting in belief state b, taking action a, and observing o. It is calculated as:

$$b_a^o(s') = \frac{P(o|s')\sum_s P(s'|s,a)b(s)}{P(o|b,a)} \tag{9}$$

Then the belief states iterate through time and get updated as follows:

$$P(b_a^o \mid b,a) = \sum_{s'} P(o_{t+1} = o \mid s_{t+1} = s') \sum_s P(s' \mid s,a)b(s) \tag{10}$$

Policies map beliefs to actions:

$$\Pi : \mathcal{B} \rightarrow \mathcal{A} \tag{11}$$

Finally, the Bellman equation for POMDP backups is defined as:

$$V(b) \leftarrow \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} R(s,a)b(s) + \gamma \sum_o P(b_a^o|b,a)V(b(a,o)) \right] \tag{12}$$

In the next subsection we will describe the representation of PT practice as a POMDP environment.

3.2 RL for intelligent automated PT

RL is often presented as the most efficient ML technique for sequential decision-making in control problems. In the quest for AI-led PT practice and unlike previously proposed approaches involving supervised learning where it is difficult to provide supervision, RL is suited because usually there is incomplete knowledge about the correct or optimal action in a given stage. In recent years, RL has achieved impressive advances in AI exceeding human expert performance in many domains notably in sequential-based problems (Roijers et al., 2013).

Modern RL involves a combination of planning and learning resulting in the field of models (initially known or learned) and learning to approximate a global value function or policy graph. While RL has shown great success in defensive cyber security, notably intrusion detection and cyber threat intelligence, it usually faces challenges like randomness, uncertainty and partial observability in offensive security. Nonetheless, it comes with important benefits such as data efficiency, targeted exploration, stability and explainability as discussed in (Moerland et al., 2020) and its suitability for PT has been demonstrated (Ghanem & Chen, 2019).

3.3 Representing PT as RL problem

In RL, we enumerate three approaches as illustrated in Fig. 3. In model-free RL, the learning occurs without access to any pre-representation or pre-structuring of the environment. Rather than building such an internal model, the RL agent instead stores estimates for the expected values of the actions available in each state or context, shaped by a history of direct interaction with the environment. By contrast, in model-based RL, the agent does possess an internal model, one that both predicts action outcomes and estimates the immediate reward associated with specific situations. Decisions are made through planning instead of on the basis of stored action values (Sarraute et al., 2012).

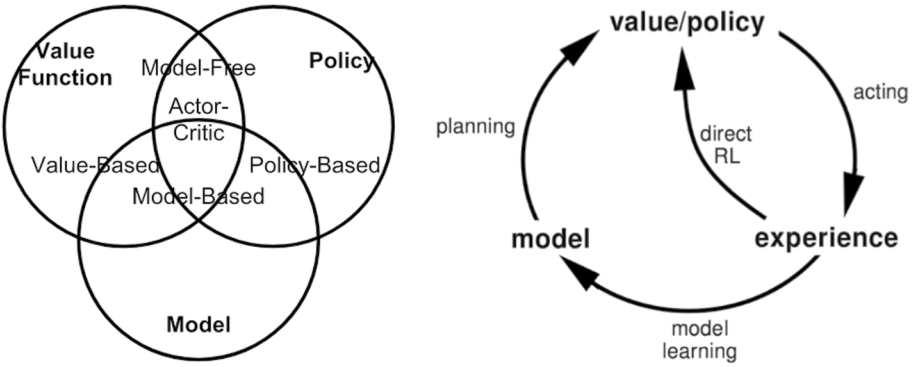

Fig. 3 A comparison of planning, model-based and model-free RL approaches and implication on policy and value function

In short, the distinction between model-free and model-based RL lies fundamentally in what information the agent stores in memory. In model-free RL, the agent stores a representation of the value associated with individual actions which is an estimate of the cumulative reward the agent expects to gain over the course of future behavior, when beginning in a particular situation with a particular action. Updates to these values, known collectively as the value function, are driven by reward prediction error which is generated based on action outcomes experienced during direct interaction with the environment (Spaan, 2012).

HRL expands the set of actions available to the RL agent to include a set of temporally extended sub-tasks or subroutines. In the implementation of HRL that we will take as our focus — the options framework3 introduced by (Sarraute et al., 2013), these temporally abstract actions are referred to as options. Options can be selected for execution, just like low-level (primitive) actions. Once this happens, behavior is guided by an option-specific policy, which dictates the action to be selected in each possible situation or state. Each option is additionally associated with an initiation set, defining the situations in which the option can be selected or launched; a termination function, which dictates when execution of the option ends; and an option-specific reward function, a pseudo-reward function, which attaches a special form of reward to specific outcomes, effectively defining the goals being pursued during execution of the option.

4 Hierarchical RL for large networks

The obvious approach to address the scalability issue in POMDP solving is by solving smaller environments which arise from dividing large environments into many smaller ones by either splitting a large network into a number of sub-networks or by considering each phase activities and tasks separately. We detail here the two approaches considered and implemented initially within IAPTF, then justify the choice of the clustering approach to achieve an efficient hierarchical RL representation of PT (Jain & Niekum, 2018).

4.1 Network security-based clustering

In this subsection, we will present the methodology adopted to address the scaling issue encountered in solving the RL problem for medium and large size networks (Ghanem & Chen, 2019). Initially, we will re-introduce the proposed POMDP model which will serve as a starting point to the new hierarchical RL model for representing large network PT. To achieve this goal we had to consider two options: the PT phases separation and security cluster separation. The latter was the most adequate in terms of efficiency and relevance. The obvious approach to address the scalability issue in POMDP solving is by solving smaller environments which arise from dividing large environments into many smaller ones by either splitting the large network into an number of sub-networks or by considering each phase activities and tasks separately.

Initially we considered a task-oriented clustering approach which aimed to divide large POMDP into smaller environments covering each one or two tasks or activities. This approach will certainly reduce the size and thus the solving time but it will produce inconsistent results as PT practice is highly interactive and some activities are repetitive and dependent on each other making tasks and activities separation irrelevant. As the task-based approach turned out to be inadequate, we changed to a more realistic and logical approach based on dividing a large network into several clusters. Each cluster is tackled separately as a small network and then the network of clusters is dealt with in the final stage. We refer to this approach as security clustering which mimics real-world PT practice by considering each part of network with the same security protection and defense at the time. Figure 4 details large LANs often found in corporate context security clustering which will be detailed later in the next subsection.

The proposed approach involves the adoption of a two-tier hierarchical POMDP representation of PT practice which is fundamentally different from sub-networking basic decomposition of POMDP as it relies on security isolation aspect and not on the network addressing mechanisms. The output is a set of small clusters (number is at least twice as superior than the subnets number) and a network of clusters. Clustering will therefore generate many small size POMDP environments and their solving will be time-efficient as we expect to avoid the scalability issue (Ghanem & Chen, 2019).

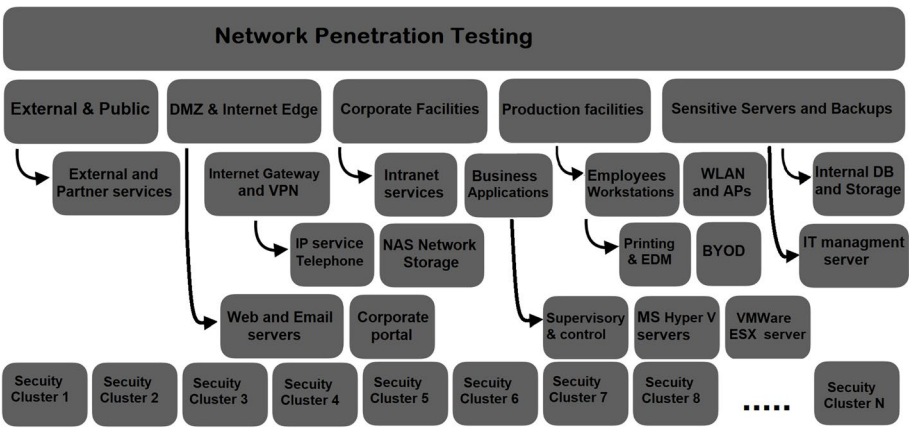

Fig. 4 Two-level hierarchical representation of PT practice using security clustering

4.2 Hierarchical POMDP representation for PT

The most important component of IAPTF-Prep is the security clustering scripts. This module output is twofold. Firstly, the cluster constitutions in term of name and IP address of each machine and the belonging cluster. secondly the head (or heads) of each cluster which is in theory the most vulnerable machine with root/admin control.

The proposed approach focuses on a key characteristic of network PT practice namely security isolation overview. This is a common approach for hackers and cyber attackers as they oversee the target network from a security point of view and not simply networking functioning in the aim of extracting key information about the security isolation and reachability (Bacudio et al., 2011). It will be noticed in the proposed clustering approach that some fundamental networking features and aspects were neglected in the task of dividing large LANs and WANs and thus modeling them into smaller POMDP environments. The eliminated networking data are indeed irrelevant from the PT point of view and doing so in IAPTF comes for the benefit of scalability as removing such useless details which results in reducing the associated POMDP environment size and therefore contribute hugely into improving the efficiency. Finally, it is important to highlight that the one-way filtering and blocking workflow adopted in security systems (firewalls, routers and intrusion detection) is a key element in our proposed clustering approach making the clustering approach fundamentally different from the regular sub-netting approach which is by default symmetric (reciprocity of reachability between subnets) as illustrated in Fig. 5 where SN0 and SN1 are considered the only subnets in the network following the symmetric reachability approach (Stock, 2017).

In term of adapting the proposed POMDP representation, we opted for a two-tier (levels) approach; first is to consider each security cluster as a separate network and only represent the data about machines in that cluster in the low-level POMDP environments (Joglekar, 2008). Then we represent the network of the head of each cluster and including all possible connections including machine-to-machine and cluster-to-cluster connections information to fully reflect the real-world inter-cluster connectivity. In the Clustering module we implemented three scripts detailed below.

The first script defines the cluster composition and named Cluster-composition.py which the output is the full security clustering of the network. In other words, the result is many clusters (number will depend on the size of the network, configurations and security

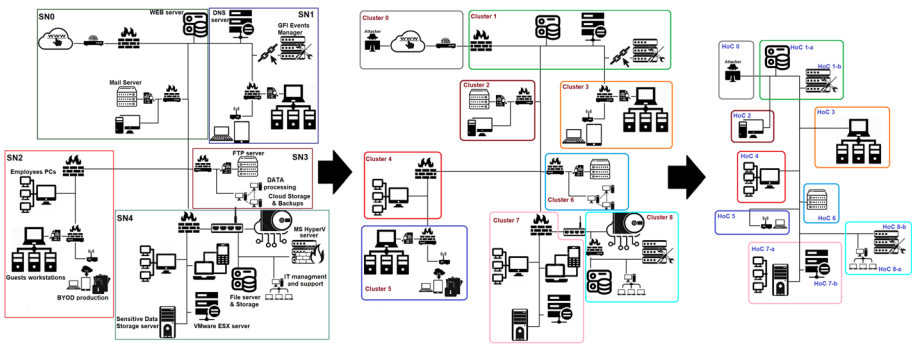

Fig. 5 Security clustering approach output from regular sub-netting in context of medium LAN, election of HoCs and constitution of intra-clusters network

setting DiD but will be at least three) which contain each many machines belonging to the same cluster will be treated similarly. The idea behind this assumption is the nature of the networking environment, the machine belonging to the same sub-net often has a similar defense and protection. In fact, an attacker who gains control of a host will easily progress to the rest of the machine on the same sub-net or cluster as a trust relation (in some applications) exists between the infected host and the remaining allowing the attacker to take advantage to execute the exploit against those machines.

The second script named Clusters-Connectivity.py oversees capturing and processing information regarding the type of connections (active and available) along with the type of the used protocol, security, and other relevant information such as the number of hopes. The number of hops is purely related to the existing security mechanism separating two different machines (cluster) such as firewalls, IDSs, IPSs, routers, and so on. The full details about the representation are in the previous section.

The third script named HoC.py is in charge of identifying the head of clusters HoC (in some occasion more than one machine is designated as HoC). The idea starts by reducing the low-risk machine first and only focus on machine with a large attack surface. Attack surface stands for the number of open ports, running services and associated vulnerabilities in the machine making it more likely to be targeted and exploited by hackers. We exclude a honeypot machine (or even a honey-net) from being HoC. An illustrative example with the output of HoC.py on the test-bed network basing on IAPTF-Prep vulnerability assessment output data is in Fig. 5.

Finally, it is worth noting that the proposed approach does include the attacker machine which is often connected through the Internet (external) but will be considered as an independent cluster (C0) for the purpose of modeling.

4.3 Representing PT in form of POMDP environment

In this section, we reintroduce the proposed representation of the PT problem in form of POMDP environment. This has been detailed in past paper (Ghanem & Chen, 2019) where the proposed representation which was introduced through a series of illustrative examples. In this paper, we will not re-introduce fully the proposed representation but simply build upon it and only focus on the features and aspects that are relevant for the HRL representation. Information about security and the network topology and security architecture are therefore included RL state space in addition to representing configuration and built of each machine in the network.

State space The most important part of the representation deals with states which include machines, networking and security information. First we consider machines configuration as separate entities. A computer machine or networking device, either physical or virtual, is represented by the character M, R or S depending on its functionality in the network, respectively, Computer, Routing device, or Security Device. Then, each character is followed an assigned ID number such as $M1$ or $R2$. Then we represent available and active connection of each machine and the the cluster to which it belongs as character C followed by ID number. The main information represented about a machine are the OS, version, Service-Pack or Knowledge-Base, open TCP or UDP ports, running services. In addition as the Vulnerability Assessment data is merged into the POMDP representation we add the relevant CVE number and details about the status which is either Secured, Vulnerable, Compromised, Untested or simply Unknown. An example of machine representation is

provided in Fig. 6. The second phase of representation covers the networking data about connection and reachability. The example of M13 and M20 connected using a TCP on SMTP and HTTP over SSL via a router and belonging to the same security cluster is represented as M13-M20-TCP-SMTP-C5. The third and final phase covers networking functionality and security consideration. We extract information from Trace-Route function and add information about hops number at the end which will represent the number of networking equipment separating the two machines with only security and routing mechanism and system being considered as hops between the machine. Thus, machines belonging to the same cluster should have a direct connection or at worst only one hop and this is reflected in the model by the number 0 or 1 meaning zero and one hops. Fig. 6 is an example of POMDP state space representation.

Action space the actions representation is meant to mimic real world PT actions performed by testers and thus encompass all PT tasks and sub-tasks following a certain notation. As with any RL problem, the number of action is known, static and limited and PT does not fall out of this logic and we include in this space a variety of PT related actions such as Probe, Detect, Connect, Scan, Fingerprint, VulnAssess, Exploit, PrivEscal, Pivot in addition to some generic action that will be used for control purpose by RL agent. The number of actions that the expert can perform is huge and cannot be totally represented within the RL action space such as Terminate, Repeat and Give-Up and others as detailed previously in [20]. Furthermore, as in PT domain successful or failed action might require further or repeating actions we defined some additional actions in order to differentiate between the original action and the others action. In practice, the purpose of such representation is to deal with the special and complex scenarios notably:

```
States:
Internet
M0
M0-win7sp1
M0-win7sp1-p3389
M0-win7sp1-p3389-RDP_ECO
M0-win7sp1-p3389-RDP_ECO-C1
M0-win7sp1-p3389-RDP_ECO-vulnerable-CVE-2018-8550
M0-win7sp1-p3389-RDP_ECO-compromised-CVE-2018-8550
M0-win7sp1-p3389-RDP_ECO-secured-CVE-2018-8550
...
R2-CISCO-XEN-p80-EC-Priv-secured-CVE-2019-17147
...
M9-Ubuntu16-p6881-BitTorrent-compromised-CVE-2018-5702
...
M7-M12-TCP-FTP-C4-1
M7-M12-TCP-RPC-C4-0
M13-M20-TCP-SMTP-C7-1
M13-M20-TCP-HTTPoSSL-C11-0
...
Terminal

actions:
...
SVCDetect
VulAssess
Exploit
Pivot
PrivEscalation
Terminate
Give_Up
...
observations:
M0-Off
M0-On
M0-OSDetectedWin7sp1
...
M0-Exploited-SMBv1
M0-Exploited-RDP_ECO
M0-Secure-SMBv1
M0-Secure-RDP_ECO
...
```

```
\Initial-Belief
start: 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0

Transitions: \probabilities
T: * : * : * 0.01
...
T: Probe : M5-Solaris5_11 : * 0.0
T: Probe : M5-Solaris5_11 : M5-Solaris5_11-p3260 0.45
T: Probe : M5-Solaris5_11 : M5-Solaris5_11-p23 0.25
T: Probe : M5-Solaris5_11 : M5-Solaris5_11-p3020 0.25
T: Probe : M5-Solaris5_11 : M5-Solaris5_11 0.05
...
T: Give_Up : * : Terminal 0.01
T: Terminate : M5 : Terminal 1.0
...
Observations: \probabilities
O: * : * : * 0.005714286
...
O: PrivEscalation : M0 : * 0.000574713
O: PrivEscalation : M0 : M0-Escal-User 0.9
O: PrivEscalation : M0-Root : * 0.000574713
O: PrivEscalation : M0-Root : M0-Escal-Root 0.9
...
Rewards:
R : PrivEscalation : M2...SMBv1-compromised : M2-Root : M2...Root 10.00
R : PrivEscalation : M2...RDP_ECO-compromised : M2 : M2...User 5.00
R : PrivEscalation : M2...RDP_ECO-compromised : M2-Root : M2...Root 10.0
...
R : Terminate : M5-Root : Terminal : Test-Achieved 100.00
R : Terminate : * : Terminal : Test-Stopped -1.00
R : Terminate : * : Terminal : Test-Overtime -1.00
R : Give_Up : * : * : *  -10.00
```

Fig. 6 Example of partial POMDP representation of PT practice including states, actions, observation, initial-belief, transition and observation probabilities and rewards

- Actions failed to fully (root) control a machine that leads to further action attempting user session or escalates privileges or switching to other attack paths.
- Action partially failed due to uncertain or incomplete information and which might succeed when additional information becomes available.
- Actions prevented or stopped by security defense (Firewalls or Intrusions Detection Systems) which may be re-attempted under different circumstances.

In terms of transition and observation probabilities as well as the rewarding scheme they remain unchanged from our previous work (Ghanem & Chen, 2019).

5 Intelligent automated PT framework

In this section, we will gradually introduce the different components and modules constituting the proposed IAPTF framework.

5.1 Pre-processing and memory building

In our proposed framework, the main challenge we noticed is that PT is a highly repetitive practice which led us to take advantage of this repetitively, re-usability and similarity rather than having it as counterweight to our framework performances (Babenko & Kirillov, 2022). We came up with idea of extracting the knowledge output during any testing and make it general (perform generalization processing) then store it into an expert system for future use. The first part of this activity is the generalization tasks which are done through python scripts directly on the output XML files of the POMDP PT solving results. Once done, we progress into storing this precious knowledge in a basic expert system (ES) using CLIPS (Zhou et al., 2008). The diagram of IAPTF-Prep system e constituted from all modules of IAPTF responsible of preparing, processing and constructing the memory is illustrated in Fig. 7. There are several different modules and features in the proposes architecture, but we only focus in this section on functions which offload the POMDP solving XML files and extract from the policy graph (PG), basing on standard formula and input regarding network configuration the decision (acting) policy made in each situation which is extracted in their original context to avoid irrelevant generalization. Then, a python script named ES-Generalization.py is applied into this data to produce a general format from which specific data is removed such as IP addresses, machine name, non-generic data.

The next step in building the IAPTF memory is the implementation of the expert system which will oversee storing and reusing of decision policies. Since the aim of this research is mainly applying RL in PT practice, the ES was a second priority, and we decided that we will not implement a heavy weight ES within IAPTF and only relying on CLIPS which is a public domain software tool for building expert systems. We will briefly describe how the general production system tool CLIPS is used to extract, process, store and reuse expertise for network penetration testing purposes using previous testing captured experiences. The proposed system can also be applied to real-time time decision making assist in terms of replying to PT tasks.

In IAPTF we opted for a direct application of CLIPS expert system to achieve our objective of capturing and replaying human CEH expertise and knowledge. CLIPS is a complete environment for developing IAPTF expert systems which includes an integrated

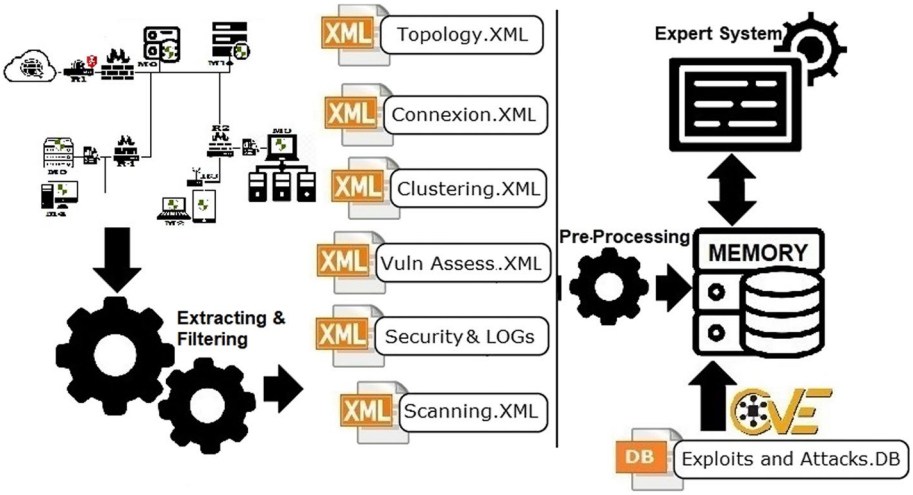

Fig. 7 IAPTF-Prep modules ensuing data collection, processing and storing functioning along with populating and interacting with the Expert System

editor and a debugging tool and enable inferences or reasoning. CLIPS provides the three key elements of: memory for data, knowledge-base, and rule-base. The written program consists of rules, facts, and objects with the inference engine to select rules (action) to be executed for a given object. In IAPTF, we built a PT expert system by performing some modification into the default CLIPS code by introducing features such as single and multiple string pattern matching, certainty factors and timestamp with uses of MSF plugins adapted for pre-processing. The complexity of MSF in terms of data handling and storing add nonetheless more complexity and challenges for our proposed expert system. To overcome these shortcomings, we proposed an integration of our developed CLIPS expert system with MSF-POSTGRESQL database. Thus, IAPTF allows the simplification of the complex data workflow by considering complete testing and attacking scenarios instead of atomic actions. Finally, a python script will ensure CVE and NVD databases import and store within the IAPTF-Memory and it will oversee the following:

- Storing and structuring in a specific format enabling CVE use and search by the IAPTF-Core directly basing on customized research criteria
- Enabling the usage of a lighter version of the large database both in terms of number of CVEs and the description information stored within the original databases CVE and NVD. Only relevant information from PT point of view is kept in IAPTF-Memory.
- Direct interaction with Metasploit MSF console to enable real-time search narrowing and prioritization based on NVD score as calculated per CVSSv3.

5.2 IAPTF-core module

Figure 8 illustrates IAPTF-Core module functioning, the main component is the RL system and the memory.In terms of solving PT generated POMDP problems, we opted for a rigorous approach aiming to scientifically elect the best solving approach following several metrics (Ghanem & Chen, 2019).

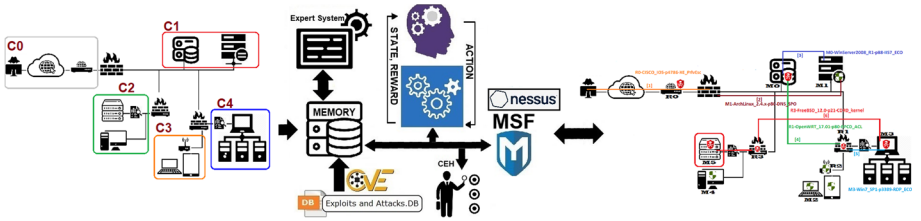

Fig. 8 IAPTF-Core module diagram with RL functioning and memory handling

The first step towards the solving is to select the appropriate solving approach. As we already demonstrated that PT practice versatility required a model-free RL and thus pre-eliminated solving approaches related to the model-based RL namely learning a model and using a model, we then considered the two available options: policy search and value iteration. PT practice is by default decision-making oriented tasks and this nature give more credit to the policy search approach as the quality of the solving is more related to the relevance of the decision policy (PG) rather than the accumulated value form repeating the task during many iterations. Fig. reffig:choices summarizes the solving choices and directions made in IAPTF.

As described earlier, our choice of embedding RL within the PT practice comes from its suitability, relevance, and pertinence to sequential decision-making problems of which PT is one of them. The second challenge we faced was to decide on the modeling approach with two major candidates namely model-free and model-based approaches being considered and investigated. Here again the nature of PT activities and the versatility of possible tasks that vary with the tested asset have heavily influenced the choice of model-free approach. In fact, despite limiting the scope of IAPTF to computer network PT, the environment remains complex and hard to capture fully which made impossible the adoption of model-based RL approach. Finally, the solving techniques were discussed, and the choice of policy search technique was made over the value iteration. This obvious choice is backed by our initial aim of fully automating and optimizing the PT practice and thus offloading the human expert whether CEH or CPT of decision making duties and thus having the software agent as replacement require decision policy tree which we construct from the POMDP solving output of policy graphs (PG).

Once the modeling approach and solving technique choices were made, the second major step was electing the appropriate solving modes; approximate or exact. The aim of this work is not only to embed RL within the PT practice to enable an intelligent automation of the PT tasks but also enhance accuracy and efficiency. Therefore, we opted for a more comprehensive framework which offers flexibility at an early stage of development. The idea is to implement and test both solving mode starting with the approximate one because of its flexibility. Many approximate solving algorithms were shortlisted to finally select PERSEUS which was the first solving algorithm implemented during early modeling stage to test and evaluate the proposed representation of PT practice as POMDP problems. Nonetheless, as IAPTF aims a high efficiency and accuracy other exact solving algorithms were candidates to be integrated and early assessment confirmed that GIP is the most relevant and efficient candidate. Thus, PERSEUS and GIP (both original and modified version) were embedded within IAPTF through the external solver POMDPsolve, and many modifications were introduced to allow a more flexible solving and input handling.

6 Solving POMDP problem

When it comes to the solving method, we considered, implemented and tested both methods of solving POMDP problem: approximate solving and exact solving. Most exact solving of POMDPs algorithms operate by optimizing the value function over all possible beliefs states but exact solutions are typically computationally intractable for all but the smallest problems. As the exact solving involving value iteration for determining the value function of POMDPs, the optimal action can be read from the value function for any belief state (Pineau & Gordon, 2003). But the exact solution comes at a cost of time and computational power which is exponential in actions and observations dimensionality of the belief space grows with number of states.

Figure 9 summarizes the choices made in the journey to model PT as RL problem along with the solving approach we opted for in our proposed framework.

6.1 POMDP approximate solving

The approximate solving approach is efficient but inexact because it relies on a discrete choice of belief states. In this work we will use it for guidance and comparison purpose. We will utilize Point-Based Value Iteration (PBVI) which begin at some initial belief then pick belief points by forward simulation and prune by distance. It approximates an exact value iteration solution by selecting a small set of representative belief points and then tracking the value and its derivative for those points only which ensures the value function increases for every belief point in every iteration. In practice, PBVI relies on a stochastic approach in choosing belief points, and by maintaining only one value hyperplane per point, PBVI is reputed for successfully solving large problems which network PT in one of them. In IAPTS, we used PERSEUS (Spaan & Vlassis, 2005); Randomized Point-based Value Iteration for POMDPs which performs approximate value backup stages, ensuring that in each backup stage the value of each point in the belief set is improved. Perseus performs backup stages until some convergence criterion is met, and the convergence criteria can be by tracking the number of policy changes (Walraven & Spaan, 2017; Spaan & Vlassis, 2005).

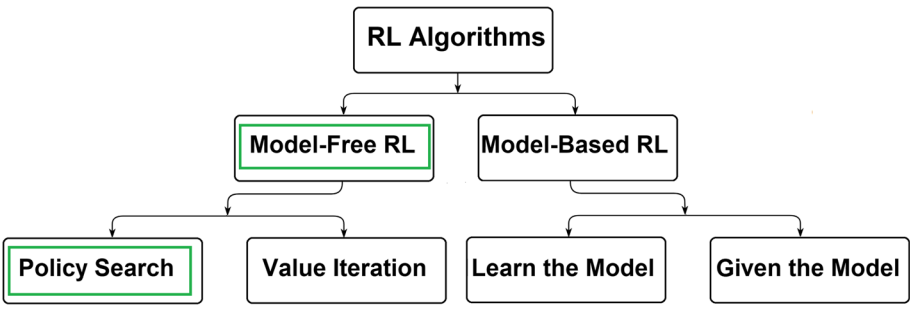

Fig. 9 IAPTF RL modeling and solving choices

6.2 POMDP exact solving

The second possible solving method of POMDP is the exact. Exact solving POMDP problem and computing the optimal solutions is challenging because of the high computational requirements of POMDP solution algorithms. We opted for several algorithms use a subroutine to prune dominated vectors in value functions, which requires a large number of linear programs (LPs) to be solved and it represents a large part of the total running time. In this paper we show how the LPs in POMDP pruning subroutines can be decomposed using a Benders (Sarraute et al., 2012). The resulting algorithm incrementally adds LP constraints and uses only a small fraction of the constraints. Our algorithm significantly improves the performance of existing pruning methods and the commonly used incremental pruning algorithm. Our new variant of incremental pruning is the fastest optimal pruning-based POMDP algorithm. Full algorithm function was detailed in (Zhang et al., 2003) and (Walraven & Spaan, 2017) proposed and implemented an enhanced version which we utilized in this research.

6.3 External solvePOMDP

In IAPTF, we opted for the option of relying on a state-of-the-art external POMDP solver called SolvePOMDP rather than implementing our internal one including LP solver and solving algorithms and associated libraries and functions from the scratch. SolvePOMDP is an open-source Java program for solving POMDPs. This solver includes an exact solving called generalized value-iteration pruning algorithm (GIP) (Walraven & Spaan, 2017) and an approximate solving called randomized point-based value iteration algorithm (PERSEUS) (Spaan & Vlassis, 2005). It comes with two built-in linear programming solvers that can be used without any additional configuration namely LPSolve and JOptimizer. The latter solver runs faster but remains less reliable as some minor numerical instability problems often result in producing unreliable output. In this research, JOptimizer was only used during early research phase as we tested the initial POMDP representation of PT practice. The input for SolvePOMDP is a set of *.POMDP files defined using Cassandra's POMDP file format (Cassandra et al., 2013) and the output resulting solutions are represented by alpha vectors and policy graphs (Spaan, 2012).

7 Test results

7.1 Test-bed and testing environment

The experiments were run on a HP Z2 tower with CPU Intel Xeon processor E7-2176, 8 core, 20MB cache and 3.70GHz, an unbuffered memory of 64GB DDR4 2666 DIMM ECC, graphical Nvidia QUADRO P4000 8GB GFX. This machine runs Linux Calculate 20 kernel version 5.4.6 which is the fastest and most resource-efficient Linux distribution based on Gentoo. This built maintains an optimal balance between state-of-the-art processing libraries and a renowned stability.

The SolvePOMDP runs without time horizon limit until a target precision $\epsilon = 0.001$ is reached. A discount rate (factor) of 0.95 was used to improve performance. We initially tested several discount rates varying from 0.5 to 0.99 and we settled on a 0.95

value which guaranteed a better balance between efficiency (time required for solving) and testing coverage (number of generated attack vectors). The POMDP environments generator is implemented in a Python script which import processed data directly from IAPTF memory and rely on the following parameters: number of machines (physical and virtual) *N*, number of identified vulnerabilities *V*, number of pre-fetched (relevant) exploits *E*, number of security clusters *C*, and number of machines with unchanged configuration *I* (since last testing).

Once POMDP environment files are created, they will be parsed into a buffer file which serves as input for the SolvePOMDP solver. At the end of each round, output PGs are translated by the attack-vector-gen.py script into attack vectors then transmitted to Metasploit framework to act upon. On the other hand, output PGs are processed in parallel using a Generalization.py scripts to make them general and are then fed to the CLIPS expert system for future direct application. In terms of SolvePOMDP parameters, the table in Fig. 10 summarizes the POMDP solving experiences set-up and used parameters.

Parameter	Value	Description
Discount Rate γ	Variable	The choices of this value (value is between 0 and 1) comes after testing many values (1, 0.99, 0.95, 0.9 and 0.8)
Max steps	500	Maximum number of steps allowed per episode, before environment is reset to start state
Value Function Tolerance	0.01	Algorithm terminates if the absolute value difference in two successive iterations is below the tolerance
Epsilon	0.001	Vectors are only added if the value improvement exceeds epsilon
Accelerated LP Threshold	100	Accelerated LP method is only used if the number of vectors in the LP exceeds the threshold. If a 0 threshold 0 is used, then it uses default pruning without the accelerated LP
Accelerated Tolerance	0.0001	Accelerated LP terminates if the absolute difference in two successive iterations is below the tolerance
Coefficient Threshold	0.000001	coefficients in LPs are discarded if their absolute value is below the threshold (to prevent numerical stability issues
Belief Sampling Runs	100	define how many runs the belief sampling executes (Approximate PERSEUS only)
Belief Sampling Steps	50	define how many steps for each belief sampling run (Approximate PERSEUS only)

Fig. 10 IAPTF-Core module diagram with RL functioning and Memory handling

7.2 Results and discussion

To evaluate the new IAPTF and notably the HRL representation as well as the different new modules, we planned a series of tests on real-world data captured for real corporate network and reconstructed and implemented on a Virtual Box environment. Although the tests cover different network size varying from 2-200 machines, our main focus are medium (30-100) and large (100-200) networks. Previous attempts to solve POMDP environment generated from a medium size LAN required a large amount of time of 149.5 hours (6.2 days) for a network of 100 machines which is an unreasonable amount of time. The poor performances in medium networks of 30-100 machines was expected as the exact POMDP solving is a P-SPACE complete problem compared with NP-complete in approximate solving, thus the time required for solving became computationally intractable. We tested the new hierarchical representation of PT which meant solving several small size POMDP problem for each cluster then solving the inter-clusters POMDP problem. We accounted for the overall time required and we repeated the test 20 times for small networks, 10 times for medium networks, and 5 times for large networks. The obtained results for five solving approaches, namely PERSEUS, RL-GIP-LPSolve, RL-GIP-LPSolve+Initial Belief HRL-GIP-LPSolve, and HRL-GIP-LPSolve+Initial Belief are plotted in Fig. 11 showing the mean values and standard deviations.

The results show some loss of performance for HRL (compared with regular RL for very small networks) with number of machines up to 10 machines (4 clusters, 33 vulnerabilities, 24 exploits). This issue is completely justified by the fact that clustering and cluster processing is useless and only slow down IAPTF. In small networks, security clustering produce often a big number of security clusters and thus many very small (2-3 machines) POMDPs on the top of the POMDP representing the Heads of clusters. This will result into forcing IAPTF in executing a big amount of data manipulation and

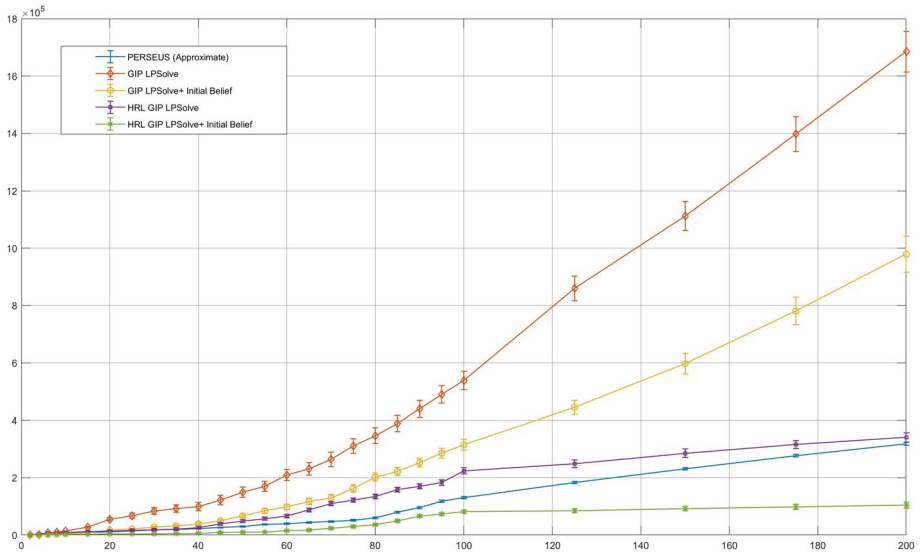

Fig. 11 Solving different size POMDP problem using different algorithm and initial belief handling approaches. X-axis represents the number of machines and Y-axis represents time in seconds necessary to solve the problem

POMDPs' solving which are in fact unnecessary gorging the fact that Regular RL solving of entire POMDP is faster

However, HRL-GIP effect is largely appreciated in larger networks and reach a very good rate in 100-machine network (25 clusters, 102 vulnerabilities, 80 exploits). HRL approach requires 224087.118 ± 12564.7 (2.6 days) compared with 538318.624 ± 31964.2 (6.2 days) in regular RL-GIP. Going beyond the 100-machine size, HRL is at least 4 times more efficient and reaching 200 machine size (52 clusters, 153 vulnerabilities, 115 exploits), HRL-GIP performed almost as well as approximate PERSEUS and required 340582.592 ± 16297.8 (3.9 days) compared with 1685011.539 ± 71160.5 (19.5 days) for RL-GIP and 278369.056 ± 5236.9 (3.2 days). When we repeat the tests using the output of previous testing as initial belief (after processing), GIP-HRL surpass PERSEUS performance and only required 1.2 days compared with 3.2 for approximate.

In terms of re-testing, we ran networks PT for each network 4 times each while introducing changes in the assessed network, the amount of change is different each time and represent 10%, 30%, 50% and 75%. The tests were carried out for algorithm variant with customized initial belief, namely RL-GIP-LPSolve+Initial-Belief and HRL-GIP-LPSolve+Initial-Belief. The obtained results were better than expected in 10% and 30% context which reflect the PT real-world domain. Figure 12 provides a comparative illustration for each of the tests comparing with the initial testing.

The results confirm the hypothesis on the crucial impact of prior knowledge and initial belief on the algorithm performances as it accelerate the convergence toward optimal value. The obtained results in the context of 100- to 200-machine LANs were extremely encouraging and nearly halved the consumed time as shown in Fig. 12.

Finally, we evaluated the overall effectiveness of IAPTF compared with regular semi-automated MSF and human CEH manual testing. The efficiency was calculated based on two metrics: the global testing time and the number of covered attack vectors, and an overall efficiency ratio was calculated. In terms of consumed time and as shown in Fig. 13, IAPTF over-performs CEH expert in medium and large networks despite the heavyweight pre-processing and post-processing. IAPTF performs twice better than human CEH in 200-machine network and 5 to 6 times better than blind automation.

In terms of coverage, we calculated the number of valid (either successful or failed) attack vector covered which we measured and compared as summarized in Fig. 13(b).

The number of attack vectors covered by all variant of IAPTF exceed by far any test performed by CEH. The number even doubled in large networks making IAPTF more reliable in terms of PT output confidence.

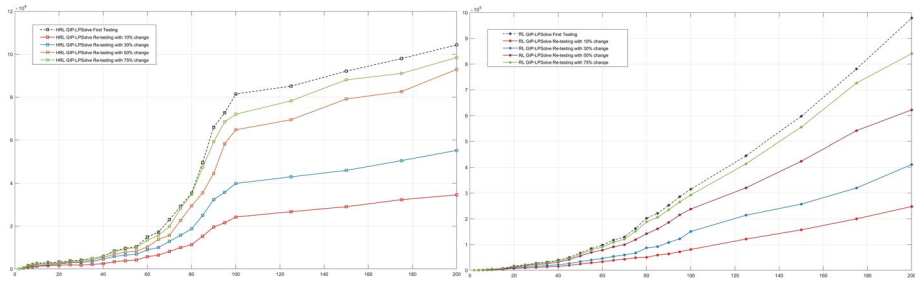

Fig. 12 Re-testing the same network after introducing a percentage of changes in machines configurations

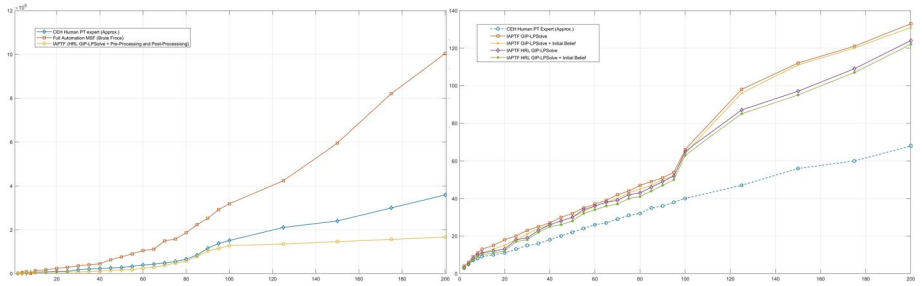

Fig. 13 Re-testing the same network after introducing a percentage of changes in machines configurations

8 Conclusion

This paper explores a novel approach of embedding RL techniques to the offensive cyber security domain. By adopting a hierarchical RL representing of the complex PT domain we overcame the huge scaling-up challenges in solving large POMDP encountered with regular RL representation of PT on medium and large networks. The proposed approach divides the network into security clusters and enables IAPTF to deal with each cluster separately as small networks (intra-clusters), then proceed to the processing of the network of clusters heads which results into covering all possible basic and most of complex (multi-steps) attacking vectors and thus matching and even exceeding the effectiveness of Certified Ethical Hackers. The proposed IAPTF is a versatile and comprehensive framework which relieves human experts from time-consuming repetitive tasks and unveil special and complex situations such as unusual vulnerabilities or combined non-obvious combinations which are often ignored in manual testing.

In terms of output, and as this research focused mainly on improving efficiency and effectiveness of IAPTF in large networks context, the obtained result are by far better in terms of time efficiency and covered tests. Although, human CEH would spend the same time in completing PT tasks on networks smaller than 100 machines, the number of covered attack vector and thus validated vulnerabilities is far superior in IAPTF. In larger network up to 200 machines, neither CEH nor automated system can compete against IAPTF when HRL is adopted as this approach addressed two major issues we faced with previous RL representation: the performance enhancement and expertise capturing.

The first is illustrated by the result of solving several small POMDP problems rather than dealing with one large and complex environment. The second is effectively addressed by the HRL approach which facilitated and simplified the process of expertise capturing and generalization which allow the re-usability in the case of retesting the same network when only few changes were introduced which is often the real-world context in PT. The results obtained confirm the efficiency, accuracy, and effectiveness of the proposed framework IAPTF designed to offload and ultimately replace the slow, costly and unreliable human PT experts. As draw back of the proposed approach, we can identify the slight decrease in the covered attack vectors which would results in missing some complex attack vectors that human hacker might adopt. This challenge will constitute a research vector for future research works.

Acknowledgements This work is part of an ongoing PhD research project currently in writing-up stage.

Author contributions Authors contributed equally to this work.

Data availability Data and Codes are available upon request.

Declarations

Consent for publication Authors give full consent for publication.

Competing interests The authors declare that they have no known competing interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abu-Dabaseh, F., & Alshammari, E. (2018). Automated penetration testing : an overview computer science and information technology.

Al-Emran, M. (2015). Hierarchical reinforcement learning: a survey. *International Journal of Computing and Digital Systems*, *4*, 2210–142. https://doi.org/10.12785/ijcds/040207.

Babenko, L., & Kirillov, A. (2022). Development of automated malware detection system. izvestiya SFedu. Engineering Sciences:153–167. https://doi.org/10.18522/2311-3103-2021-7-153-167.

Backes, M., Hoffmann, J., Künnemann, R, Speicher, P., & Steinmetz, M. (2017). Simulated penetration testing and mitigation analysis. arXiv:1705.05088.

Bacudio, A., Yuan, X., Chu, B., & Jones, M. (2011). An overview of penetration testing. *International Journal of Network Security & Its Applications*, *3* (1-2), 19–38. https://doi.org/10.5121/ijnsa.2011.3602.

Bertoglio, D.D., & Zorzo, A.F. (2017). Overview and open issues on penetration test. *Journal of the Brazilian Computer Society*, *23*(1), 1–16.

Boddy, M., Gohde, J., Haigh, T., & Harp, S. (2005). Course of action generation for cyber security using classical planning. Proceedings of the 15th international conference on automated planning and scheduling. ICAPS'05:12–21.

Cassandra, A.R., Littman, M.L., & Zhang, N.L. (2013). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. arXiv preprint arXiv:1302.1525

Ghanem, M., & Chen, T. (2019). Reinforcement learning for efficient network penetration testing. *Information*, *11*, 6. https://doi.org/10.3390/info11010006.

He, L., & Bode, N. (2006). Network penetration testing. In A Blyth (Ed.) *EC2ND 2005. Springer, pp 3–12*.

Jain, A., & Niekum, S. (2018). Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics.

Joglekar, N. (2008). Hierarchical planning under uncertainty: Real options and heuristics, pp 291–313. https://doi.org/10.1016/B978-0-7506-8552-8.50014-1.

Maeda, R., & Mimura, M. (2021). Automating post-exploitation with deep reinforcement learning. *Computers & Security*, *102108*, 100. https://doi.org/10.1016/j.cose.2020.102108.

Moerland, T., Broekens, J., & Jonker, C. (2020). Model-based reinforcement learning: a survey. https://doi.org/10.48550/arXiv.2006.16712.

Pineau, J., & Gordon, G. (2003). Point-based value iteration: an anytime algorithm for pomdps. In *Proceedings international joint conference of artificial intelligence,* pp. 1025–1032.

Phong, C., & Yan, W. (2014). An overview of penetration testing. *International Journal of Digital Crime and Forensics (IJDCF)*, *6*, 50–74. https://doi.org/10.4018/ijdcf.2014100104.

Roijers, D.M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, *48*, 67–113.

Spaan, M., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for pomdps. *Journal Artificial Intelligent Research (JAIR)*, *24*, 195–220.

Spaan, M.T. (2012). Partially observable markov decision processes. In *Reinforcement learning,* pp 387–414.

Sarraute, C., Buffet, O., & Hoffmann, J. (2012). Pomdps make better hackers: accounting for uncertainty in penetration testing. Proceedings of the twenty-sixth aaai conference on artificial intelligence, pp 1816–1824.

Sarraute, C., Richarte, G., & Lucángeli Obes, J. (2013). An algorithm to find optimal attack paths in nondeterministic scenarios. Proceedings of the acm conference on computer and communications security. https://doi.org/10.1145/2046684.2046695.

Stock, S. (2017). Hierarchical hybrid planning for mobile robots. *KI-Künstliche Intelligenz*, *31*(4), 373–376.

Walraven, E., & Spaan, M.T.J. (2017). Accelerated vector pruning for optimal pomdp solvers. In *AAAI*.

Walraven, E., & Spaan, M. (2017). Accelerated vector pruning for optimal pomdp solvers. Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31(1).

Yaqoob, I., Hussain, S., Mamoon, S., Naseer, N., & Akram, J. (2017). Penetration testing and vulnerability assessment. *Journal of Network Communications and Emerging Technologies*, *7*, 12–21.

Zennaro, F.M., & Erdodi, L. (2020). Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning. arXiv:2005.12632.

Zhang, W., Nevin, D., Zhang, N., Supervisor, T., & Golin, G. (2003). Algorithms for partially observable markov decision processes. https://doi.org/10.14288/1.0098252.

Zhou, R., Pan, J., Tan, X., & Xi, H. (2008). Application of clips expert system to malware detection system, vol. 1, pp. 309–314. https://doi.org/10.1109/CIS.2008.100.