POMDP + Information-Decay: Incorporating Defender's Behaviour in Autonomous Penetration Testing

Jonathon Schwartz, [1] Hanna Kurniawati, [1] and Edwin El-Mahassni[2]
{jonathon.Schwartz, hanna.kurniawati}@anu.edu.au, Edwin.El-Mahassni@dst.defence.gov.au
[1] Research School of Computer Science, Australian National University
[2] Defence Science and Technology, Australian Department of Defence

Abstract

Penetration testing (pen-testing) aims to assess vulnerabilities in a computer network by emulating possible attacks. Autonomous pen-testing allows frequent and regular pen-testing to be performed, which is increasingly necessary as networks become larger and more complex. Autonomous pen-testing is a planning under uncertainty problem, where the uncertainty is caused by partial observability of the network, lack of reliability of attack tools, and possible changes in the network that are triggered by the network administrator (the defender). Approaches that account for the first two causes of uncertainty have been developed based on the mathematically principled framework, Partially Observable Markov Decision Process (POMDP). However, they do not account for the third type of uncertainty. On the other hand, work that accounts for the defender's actions do not account for both partial observability and unreliability of the attack tools. ***This paper proposes a POMDP-based autonomous pen-testing framework that accounts for the defender's behaviour, thereby accounting for all of the above three causes of uncertainty.*** Key to our model is the observation that the defender's actions can be abstracted into two types: Network analysis, which does not alter the network, and active defence operations, which alter the network. This observation enables us to represent the defender's behaviour as a single variable: An information decay factor. This variable is based on the expected time the defender takes to move from analysing to actively defending the network, and therefore represents the decay of a pen-tester's knowledge about the network. We propose D-PenTesting, which assumes the decay factor is known prior to execution, and LD-PenTesting, which learns the decay factor as it attempts to break into the network. Simulation tests on two benchmark scenarios indicate that D-PenTesting and LD-PenTesting outperform existing POMDP-based pen-tester and is more robust than one that incorporates a POMDP-based defender.

Introduction

Penetration testing (pen-testing) aims to identify vulnerabilities in a computer network by emulating a real attack. It is essential to ensure network security. However, due to the highly skilled and expensive specialists required, pen-testing

is performed infrequently, if at all, which is not ideal as networks become larger and more complex. These difficulties have led to the proposal of autonomous pen-testing. In this paper, we focus on a component of autonomous pen-testing: The problem of deciding which tools to use when, assuming the pen-tester is a single agent.

The above problem is a planning under uncertainty problem, where uncertainty is caused by (*i*) partial observability of the network properties and its vulnerabilities, (*ii*) the lack of reliability of the tools used, and (*iii*) the possible changes to the system triggered by the network administrator (aka the defender). The first two causes of uncertainty have been modelled in various prior work since 2011 (Sarraute, Buffet, and Hoffmann 2011; Hoffmann 2015) based on the Partially Observable Markov Decision Process (POMDP) framework. A POMDP agent acknowledges he or she never knows its exact state, and therefore, it estimates its current state as a distribution over the state space, called beliefs, and decides the best action to perform with respect to beliefs, rather than single states. Such a framework fits well for modelling the first two types of uncertainty in pen-testing problem. However, incorporating the defender's behaviour is more difficult. To this end, autonomous pen-testing relies on the game theoretic framework (Liang and Xiao 2012). However, current game theoretic approaches do not account for partial observability of the network or unreliable attack tools.

This paper proposes a POMDP-based pen-tester that takes into account network changes triggered by the defender. To this end, we develop a compact defender model based on the observation that a pen-tester's knowledge about the defender's actions and strategies are only gained via observations about changes in the network. Therefore, we can abstract the defender's actions into two types: Network analysis, which does not change the network, and active defence operation, which alters the network properties. Based on this abstraction, we model the defender's behaviour as a Markovian Arrival Process (Asmussen 2010; Neuts 1979). In this paper, we adopt the simplest Markovian Arrival Process, the Bernoulli process, and model the defender's behaviour simply as the parameter for this Bernoulli process. From the defender's viewpoint, this parameter indi-

cates the expected time the defender takes to switch from analysing to actively defending the network. From the pen-tester viewpoint, it represents the probability that the defender alters the network and compromises information the pen-tester has about the network. Therefore, we call this Bernoulli parameter *information decay factor*.

Utilising the above model, we propose a POMDP-based pen-tester, called D-PenTesting, that incorporates the defender by augmenting the uncertainty in the effect of actions with the decay factor. Since the decay factor may differ according to a particular defender, its value is generally unknown in advance. Hence, we also propose a Bayesian Reinforcement Learning (BRL) pen-tester, called LD-PenTesting, that learns the decay factor as it attempts to break into the network. LD-PenTesting frames the learning problem as BRL and solves it as yet another POMDP. We tested both D-PenTesting and LD-PenTesting on two pen-testing scenarios that were introduced in previous work on POMDP-based pen-testing (Sarraute, Buffet, and Hoffmann 2012) and stochastic-game-based pen-testing (Lye and Wing 2005). In both scenarios, D-PenTesting and LD-PenTesting outperform existing POMDP-based pen-testing methods and are more robust than an autonomous pen-testing agent that incorporates a POMDP-based defender.

Background and Related Work

POMDP Formally, a POMDP is a tuple $\langle S, A, T, O, Z, R, \gamma \rangle$ (Kaelbling, Littman, and Cassandra 1998). At each time-step, a POMDP agent is in a state $s \in S$, executes an action $a \in A$, perceives an observation $o \in O$, and moves to the next state $s' \in S$. The next state is distributed according to $T(s, a, s')$, which is a conditional probability function $P(s'|s, a)$ that represents uncertainty in the effect of actions. The observation o perceived depends on the observation function Z, which is a conditional probability function $P(o|s, a)$ that represents uncertainty in sensing. The notation R is a state-action dependent reward function, from which the objective function is derived. The notations $\gamma \in (0, 1)$ is the discount factor to ensure that an infinite horizon POMDP problem remains a well-defined optimisation problem.

The solution to a POMDP problem is an optimal policy π^*, that maps beliefs to actions in order to maximise the expected total reward, i.e. $V^*(b) = \max_{a \in A} \left[R(b, a) + \gamma \sum_{o \in O} Pr(o|a, b) V^*(\tau(b, a, o)) \right]$, where $R(b, a) = \sum_{s \in S} R(s, a) b(s)$ and $Pr(o|a, b) = \sum_{s' \in S} Z(s', a, o) \sum_{s \in S} T(s, a, s')$. The function τ computes the updated belief after the agent executes a from b and perceives o.

Bayesian Reinforcement Learning (BRL) Formally, reinforcement Learning (RL) is the problem of finding a Markov Decision Processes (MDP) solution when the transition and/or reward function is unknown or partially known. MDP is a fully observed version of POMDP, meaning the effects of actions are non-deterministic but, once an action is performed, the agent's state is fully revealed. POMDP can be viewed as MDP in the belief space because, the subsequent belief of a POMDP agent after it performs an

action and perceives an observation is deterministic. However, the belief evolution for MDP in the belief space depends on pairs of action–observation rather than action alone. Therefore, finding a POMDP policy when the transition/observation/reward functions are unknown or partially known is an RL problem too.

Interestingly, RL can be solved as a POMDP, too. This approach for solving RL is called Bayesian RL (BRL)(Ghavamzadeh et al. 2015). It represents model uncertainty stochastically and leverages Bayesian inference to learn the model from observations, while finding a good policy to achieve the agent's goal. In BRL, the POMDP state space consists of a joint product between the state space of the system and the model parameters being learned. The POMDP beliefs then represent uncertainty on both the system's state and on the model, while the policy represents the best action to perform, so as to simultaneously optimise model learning and goal achievement. This then allows the RL agent to balance model learning and goal achievement, in the sense of learning the model just enough to accomplish the goal well, rather than learning the best model possible.

Markov Arrival Process (MAP) MAP models an arrival event as an absorption in an absorbing Markov Chain that evolves from a transient state. Once the Markov Chain arrives in an absorbing state, it resets to a transient state that may depend on the state where absorption occurs. A nice property of MAP is that it can approximate any stationary point process arbitrarily closely(Ibe 2013). In this paper, we apply the simplest type of MAP, the Bernoulli Process.

Pen-testing A good review of developments with respect to pen-testing and AI planning can be found in (Hoffmann 2015) and for game theory and cyber security in (Liang and Xiao 2012; Merrick et al. 2016).

Early research and success in autonomous pen-testing came with the development of attack graphs, which involve modelling individual attack actions in terms of preconditions and post conditions (Swiler and Phillips 1998; Lippmann and Ingols 2005; Ammann, Wijesekera, and Kaushik 2002). Combining attack graphs with AI planning made it possible to generate attack paths through a target system to identify vulnerabilities and has been applied in commercial systems (Boddy et al. 2005; Lucangeli, Sarraute, and Richarte 2010). However, the classical planning approach requires complete knowledge of the network topology and host configurations. Additionally, it assumes that host configurations are static and actions are monotonic; meaning once an attack asset is gained, it cannot be lost. These restrictions make classical planning limited when trying to model the uncertainty inherent in realistic pen-testing.

Framing the pen-testing problem as a POMDP models partial observability in autonomous pen-testing (Sarraute, Buffet, and Hoffmann 2011). This approach removed the requirement that each host's configuration be known at the start of an attack. Additionally, by incorporating this POMDP based pen-tester into a hierarchical model, it was shown that it could scale to large networks and handle actions that may cause a host to crash (Sarraute, Buffet, and Hoffmann 2012). Simpler models for planning under uncer-

tainty have also been proposed for autonomous pen-testing that have better scaling properties (Durkota and Lisỳ 2014; Shmaryahu et al. 2018). Although able to more realistically model pen-testing compared to classical planning, these approaches still assume monotonic actions, static host configurations and the absence of a defender.

To model both the attacker and defender, many studies have used game theoretic approaches to network security (Lye and Wing 2005; Alpcan and Basar 2006; Nguyen, Alpcan, and Basar 2009; Liang and Xiao 2012). However, these studies have either framed the problem as a fully observable stochastic game or have not accounted for both partial observability of the network and stochastic actions.

Taking a step back, pen-testing is essentially a Partially Observable Stochastic Games (POSG) problem(Hansen, Bernstein, and Zilberstein 2004). The pen-tester must find a strategy to beat the defender, despite partial information and observation about the state of the system and defender, and despite uncertainty in how its actions affect the system. POSG is a generalisation of POMDP to multi-agent with possibly conflicting reward. This is a very powerful framework but, it is even harder than POMDP to solve, and despite advances in POMDPs, finding a good POSG policy remains an open problem. Therefore, most autonomous pen-testing relaxes the problem by accounting for only partial-observability or defender's behaviour, as briefly discussed above. Simplified POSG model, such as Interactive-POMDP(Gmytrasiewicz and Doshi 2004), and their solutions(Doshi and Gmytrasiewicz 2009; Hoang and Low 2013) have been proposed. However, they require information about transition functions of the defender, which is difficult to know and learn. This paper proposes a further model simplification by combining POMDP and MAP.

Problem Formulation

We define the POMDP model of a D-PenTesting agent with decay factor $d \in [0,1]$ as $\mathcal{P}_d = \langle S, A, T_d, O, Z, R, \gamma \rangle$.

The state space is defined as $S = S_1 \times S_2 \times \cdots \times S_n$, where each state variable represents a property of the network and n is a finite natural number that represents the number of network properties the agent might alter. This network property can be relatively primitive such as, whether the ssh port of a machine is open, or relatively high level, such as whether the network has been compromised.

The action space A represents the set of actions available to the pen-tester. Each action $a \in A$ is associated with an affected set, denoted as $I(a)$. An affected set of an action consists of all the state variable indices that correspond to the network properties "directly affected" by the action. Here, "directly affected" means that either the particular network properties are post-conditions of the action or they are the properties observed by the action if the action is a pure information gathering action such as, scanning ssh port.

Although the action space in this model consists of only the pen-tester's actions, the transition function T_d considers both the pen-tester's and defender's actions. A defender's action is implicitly accounted by the decay factor d, which represents the probability that the value of a state variable is changed by the defender. Before describing the exact transition function, we will first describe the model and assumptions of the defender in D-PenTesting.

A D-PenTesting agent only considers the defender's actions that might alter network properties that the pen-tester agent might alter, i.e., properties represented by at least one of the state variables in S. More precisely, D-PenTesting assumes that the set of possible actions that the defender might perform is $A^{def} = A_1^{def} \times A_2^{def} \times \cdots \times A_n^{def}$, where each variable A_i^{def} for $i \in [1, n]$ has two values: An action to assess whether the network property represented by S_i has been altered by an attacker and a patching action that nullifies the attacker action(s) on S_i. The first action does not change the state of the D-PenTesting agent, while the second obviously, might.

D-PenTesting then assumes a typical defender's strategy, which starts by analysing the network leading to performing a patch that nullifies the attacker's actions. However for simplicity, D-PenTesting assumes that the effect of the defender's actions on a state variable is independent from its effect on different state variables. The defender's move from analysing to patching the network properties related to S_i ($i \in [1, n]$) can naturally be cast as a Markovian Arrival Process.

In this paper, we adopted the simplest Markovian Arrival Process, which is the Bernoulli Process. The success probability p_i of the Bernoulli Process represents the probability that at the current time-step, the defender nullifies the attacker's action related to S_i. D-PenTesting assumes the defender has the same capability in analysing attacks related to various different state variables, and therefore uses the same success probability for all $i \in [1, n]$, which is denoted as the decay factor d.

Now, we can describe the transition function T_d in more detail. Without loss of generality, suppose the action performed is $a \in A$ and the affected set is $I(a)$, then the transition function is: $T_d \left(\prod_{i=1}^{n} S_i' \mid \prod_{i=1}^{n} S_i, a \right) = \prod_{j \notin I(a)} T_d \left(S_j' \mid S_j, a \right) \cdot T_d \left(\prod_{i \in I(a)} S_i' \mid \prod_{k=1}^{n} S_k, a \right)$
The transition for S_j where $j \notin I(a)$ is then

$$T_d \left(S_j' = s_j' \mid S_j = s_j, a \right) = \begin{cases} d \cdot \frac{1}{|S_j'|-1} & s_j' \neq s_j \\ 1 - d & \text{Otherwise} \end{cases} \quad (1)$$

The first line in eq.(1) represents the case when the value of S_j is changed by the defender. The probability that the defender made the change is d. In the simplest case, the exact value the defender changes S_j into is assumed to be uniform. However, it is also simple enough to incorporate different distributions over the values of S_j if additional information is known. The transition $T_d \left(\prod_{i \in I(a)} S_i' \mid \prod_{k=1}^{n} S_k, a \right)$ will be affected only by the reliability (i.e. probability of success) of the action performed by the pen-tester, i.e., the defender's action is not counted. The reason is, D-PenTesting assumes that the defender cannot change the network properties affected by the pen-tester within the same time that the properties are being modified by the pen-tester. Note that for pure information gathering action, the action does not change the values of the state variables.

To perceive an observation about a property of the network, generally, a pen-tester must probe the network with the appropriate tools. The agent may perceive information about multiple state variables at once and the observation function itself may or may not be perfect.

A high reward is given whenever the network is compromised and small cost is given for each step. Rewards can also be used to reduce the chances of being caught.

Note that the decay factor d in T_d does not directly affect the observation and reward functions. Rather, it affects the state of the network and the pen-tester's understanding about the state of the network, which is reflected in the belief.

Now, the extremely simplified defender model may cast doubts on the ability of D-PenTesting to handle deceptions such as honeypots. Interestingly, this simple defender model does not prevent a D-PenTesting agent from avoiding deceptions, as long as the honeypots can be identified with non-zero probability. Avoiding deception, such as honeypots, is then a matter of setting the appropriate penalty to discourage D-PenTesting from entering.

Learning The Decay Factor

The decay factor d essentially represents how fast a defender is able to analyse and act to nullify the work of an attacker. This speed will likely differ from one defender to another, and therefore needs to be learned while the pen-tester assesses the network.

The above learning problem can naturally be framed as a BRL problem. Here, BRL balances the effort for learning a behaviour model of the defender (in this case, represented as the decay factor) and breaking into the network by learning a good enough model for making good decisions, rather than learning the most accurate model possible. We call this BRL-based pen-testing agent, LD-PenTesting. To compute a good strategy for an LD-PenTesting agent, we cast the BRL framing of the problem as yet another POMDP and solve the POMDP approximately using existing solvers. The policy of a LD-PenTesting agent is a strategy for the agent to break into the network, while learning the defender's behaviour.

Suppose $\mathcal{P}_d = \langle S, A, T_d, O, Z, R, \gamma \rangle$ is the POMDP model of the D-PenTesting agent. Then, the corresponding LD-PenTesting agent is defined as a POMDP $\mathcal{P} = \langle S, A, T, O, Z, R, \gamma \rangle$, where:

- $\mathcal{P}.S = \mathcal{P}_d.S \times D$, where D is the model parameter, which in this case is the set of possible values for the decay factor. To reduce computational complexity, we discretise D up to a certain resolution, denoted as δ. This state space definition means that the belief of the LD-PenTesting agent represents the uncertainty of the pen-tester in both the state of the network and the defender's behaviour, where the defender's behaviour refers to its speed in moving from analysing to patching the network.

- $\mathcal{P}.A = \mathcal{P}_d.A$. The action space is unchanged.

- The transition function differs slightly. Assuming the model parameter does not change over time, the transition

function of $\mathcal{P}$ can be computed as:

$$
\begin{aligned}
\mathcal{P}.T(\langle s, d \rangle, a, \langle s', d' \rangle) &= P(s', d' \mid s, d, a) \\
&= P(s' \mid d' s, d, a) \cdot P(d' \mid s, d, a) \\
&= P(s' \mid d', s, d, a) \cdot \Delta_{dd'} \\
&= \mathcal{P}_d.T_d(s, a, s') \cdot \Delta_{dd'}
\end{aligned}
$$

where $\Delta_{dd'}$ is the Kronecker Delta function, $\Delta_{dd'} = 1$ whenever $d = d'$ and 0 otherwise. Note that although we assume the parameter does not change, the pen-tester's understanding about the model parameter might change over time, which is reflected in the belief of the pen-tester.

- $\mathcal{P}.O = \mathcal{P}_d.O$, $\mathcal{P}.Z(\langle s, d \rangle, a, o) = \mathcal{P}_d.Z(s, a, o)$, $\mathcal{P}.R(\langle s, d \rangle, a) = \mathcal{P}_d.R(s, a)$. As described in the previous section, the observation space and the observation and reward functions are not affected directly by the decay factor. LD-PenTesting learns the model parameter indirectly, in the sense that the transition function under different decay factors results in different states, leading to different observations being perceived. Given the same action and a different decay factor, the probability of perceiving the particular observation differs and this differing probability will result in a different Bayesian estimate about the state of the network and the decay factor.

The simplified model of the defender enables us to construct a compact POMDP representation of the BRL problem. In this paper, for comparison purposes, we solve this BRL using an off-line POMDP solver. However, both D-PenTesting and LD-PenTesting are general enough to be used with any POMDP solver, including on-line solvers such as those found in (Silver and Veness 2010; Somani et al. 2013; Kurniawati and Yadav 2013), in a straightforward manner, which will improve scalability.

Note that since LD-PenTesting learns the defender's behaviour during execution, it relaxes the MAP (and Bernoulli Process) requirement that the defender's behaviour is stationary. Moreover, this is true even when we use off-line POMDP solver because during off-line computation, LD-PenTesting computes the attack strategies under various distributions over the decay factor. In terms of problem size, LD-PenTesting only increases the size of the state space of its corresponding D-PenTesting agent by $\frac{1}{\delta}$ fold.

Experiment Scenarios

We test our approach using two different scenarios in simulation. The first is based on (Sarraute, Buffet, and Hoffmann 2012), where pen-testing is modelled as a POMDP but did not incorporate defender's behaviour. The second is based on (Lye and Wing 2005), where cyber security operations are modelled as stochastic games, incorporating actions of both the pen-tester and the defender, but without incorporating uncertainty. We extend both scenarios so that each incorporates uncertainty and the defenders actions. Table 1 shows the POMDP size of D-PenTesting and LD-PenTesting for each scenario. The rest of this section presents the scenarios, with detailed components provided in the supplementary materials.

Table 1: POMDP problem size for each scenario and agent. *Min steps* is the minimum number of actions to reach the goal, and a lower bound on the minimum planning horizon.

Scenario, pen-testing agent	$|S|$	$|A|$	$|O|$	Min steps
1, D	3072	20	6	1
1, LD ($\delta = 0.1$)	30720	20	6	1
2, D	128	8	2	5
2, LD ($\delta = 0.1$)	1280	8	2	5

Experiment Scenario 1

The first scenario involves a single host where the goal of the pen-tester is to control the host (Sarraute, Buffet, and Hoffmann 2012).

State The state for this scenario is $S = C \times T \times K_1 \times \cdots \times K_n$, where C represents whether the host has been successfully controlled, T represents if the pen-tester has given up the attack, and K_i for $i \in [1, n]$ represents a component of the hosts configuration. Both C and T variables are fully observable and the scenario ends when either becomes *true*.

The host configuration components include the OS of the host and each of the ports targeted by the pen-testers available attack actions. The exact OS and ports considered are in supplementary materials.

Pen-tester

- **Actions:** Along with a *terminate* action which voluntarily ends the attack, we consider two types of actions: scans and attack actions. Scan actions allow the pen-tester to acquire information about the configuration of the host. The agent has a *scan-OS* action, for scanning the hosts OS, along with a *port-scan* action for each port included in the state, for detecting if a port is open or closed. A list of the attacks are in supplementary materials.

- **Transition:** Attack actions are deterministic and, given their preconditions are met, will always result in the host becoming *controlled*. For example, the *vsftpd-234* requires *os=linux* and *port 21=open*, if these state conditions are met this action will succeed. The uncertainty for the pen-tester comes from having no knowledge of the initial configuration of the host and uncertainty due to the actions of the defender. The latter, we represent by the decay factor. In D-PenTesting the decay factor is fixed, while in LD-PenTesting it is learned.

- **Observations and Observation function:** The attack and terminate actions affect the fully observable part of the state, namely the *controlled* and *terminated* state variables and thus receive their observations directly from fully observable changes in the state. The OS and port scan actions return an accurate observation of the state variable that they target.

- **Rewards:** In line with the original study we use a cost of 10 for both port scans and attack actions and a cost of 50 for an OS scan. The attacker receives zero reward for using the *terminate* action and moving into the *terminated*

state, while they receive a large reward of +9000 for successfully controlling the host.

- **Initial belief:** The pen-tester starts with no knowledge of the host configuration. The belief is uniformly distributed across the possible values for each state variable.

Defender To provide an opponent for evaluation purposes, we extend the original paper and incorporate a defender into our scenario. The strategy and actions of the defender are not incorporated into the planning of the pen-tester at all and are used only during evaluation.

- **Actions:** The defender has an action for controlling each aspect of the host configuration. Specifically, for each port the defender has an *open-port* and *close-port* action. For the OS the defender has an action to change the host to a specific OS: *change-os-linux*, *change-os-windows* and *change-os-openBSD*. We also include a *do-nothing*, that has no effect on the state.

- **Strategies:** We test our approach against two different defender strategies: no defender and random defender. The random defender simply chooses an action uniformly at random from all available actions at each time step. This strategy is used to test how well each pen-tester can handle random changes to the state.

Experiment Scenario 2

The second scenario is based on a stochastic-game involving a pen-tester and defender operating on a small network of two machines (Lye and Wing 2005). We specifically focus on the *stealing confidential data* scenario (scenario 3, section 4.3) from the original paper, where the goal of the pen-tester is to steal data from the workstation machine. Figure 1 shows a high-level graph for the pen-tester and defender. In the original scenario, the state was fully observed. We modify the scenario to make the state partially observable.

State The state represents the current state of an attack on the network. Each state variable is binary and corresponds to a different stage in the attack. The state variables for the scenario can be seen in figure 1, where each node in the graph, except the "Normal Operation" node, corresponds to a state variable. The "Normal Operation" node simply represents the case where all state variables are *false*. The scenario ends when the pen-tester steals the data from the workstation.

Pen-tester

- **Actions and Transition Function:** Figure 1 shows each of the possible pen-tester actions, their cost and the effect they have on the state (solid edges). The pen-tester also has a *do-nothing* action which has zero cost and has no effect on the state. Each action has associated preconditions, cost, and success probability. The preconditions define the state required for the action to have a chance to succeed. The success probability defines the chance the action will succeed, given its preconditions are satisfied. The cost and success probability values were chosen to match the original study.
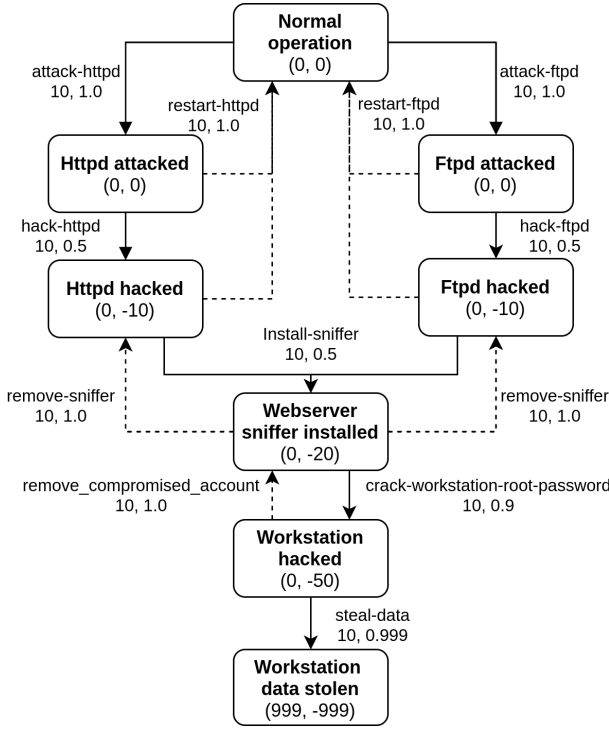
Figure 1: Graph representation for scenario 2. Each node represents a different stage of the attack. The "Normal operation" node represents when no attack is being performed. The values in brackets within each node are the reward for the pen-tester and defender, respectively. The solid line edges correspond to actions by the pen-tester, while dashed line edges are defender actions. Each action has the cost and success probability under the actions name.

- **Observations and Observation Function:** For each action performed, the pen-tester receives an observation; *success* if the action succeeded and *failure* if it failed.

- **Rewards:** In addition to the cost associated with performing each action the pen-tester receives a reward of +999 for successfully stealing the workstation data and 0 for any other state.

Defender As was the case for scenario 1, the strategy and actions of the defender are used only for evaluation but not for planning, with the exception of Oracle-PenTesting which we discuss in the next section.

- **Actions and Observations:** All defender actions, with the exception of *monitor*, are shown in figure 1, along with their costs and success probabilities. The *monitor* action has no effect on the state but allows the defender to make a noisy observation of the current state. This is similar to how an Intrusion Detection System (IDS) alerts system administrators of a potential attack. We chose a success probability of 0.9 for the *monitor* action to account for false positives and negatives common to an IDS. The *monitor* action has a cost of 1.

- **Rewards:** The goal of the defender is to prevent the pen-tester from stealing the data from the workstation. With that in mind, the defender receives different rewards for each state depending on how close the state is to the pen-tester stealing the data. Figure 1 shows the reward corresponding to each stage of the attack.

- **Strategies:** As with scenario 1, we test each pen-tester agent against no defender and random defender strategies. We also test against a defender policy generated by solving the defender POMDP (which we hereby refer to as the defender POMDP strategy).

Results and Discussion

We ran experiments on the two scenarios described above in order to: (*i*) find the parameters for D-PenTesting and LD-PenTesting agents and analyse the sensitivity of performance with respect to parameters, and (*ii*) compare performance of D-PenTesting and LD-PenTesting agents against other pen-testing agents.

Experimental Setup

Experiments were run on a machine with an Intel Xeon Silver 2.1 GHz CPU and 128 GB of RAM. We used the Approximate POMDP Planning toolkit (APPL) (Du et al. 2014), which provides an efficient C++ implementation of the SARSOP algorithm (Kurniawati, Hsu, and Lee 2008).

For planning, using a discount factor of $\gamma = 0.95$, we ran SARSOP until the planning time of 1 hour or the target precision of $\epsilon = 0.001$ was reached. The exact solving times of each agent is presented in supplementary materials.

We evaluated the performance of each agent by running simulations using APPL. We ran a total of 1000 simulations for each experiment with each simulation running for a maximum of 100 time steps. At each time step during the simulations, each player performed an action, with the pen-tester always going first, followed by the defender. For each simulation, the pen-tester won if they managed to reach the goal within the time step limit, otherwise the defender won.

Defenders For scenario 1, we evaluated each PenTesting agent against two defender strategies: no defender and random defender. For Scenario 2, we also evaluated agents against a POMDP defender.

POMDP-PenTesting As a baseline, we compare D-PenTesting and LD-PenTesting against the POMDP pen-tester approach with no decay factor (referred to as POMDP-PenTesting) (Sarraute, Buffet, and Hoffmann 2011; 2012).

Oracle-PenTesting Additionally, we compare D-PenTesting and LD-PenTesting against a stronger pen-tester agent: Oracle-PenTesting. We augment the POMDP-PenTesting agent by adding the defender's action as a fully observable variable to the agents state. During planning, the Oracle-PenTesting agent has access to the defender POMDP agent's policy and incorporates this into its transition function. Note, however, that Oracle-PenTesting remains a POMDP agent, and not a Partially Observable Stochastic Game agent. During each simulation, at each step
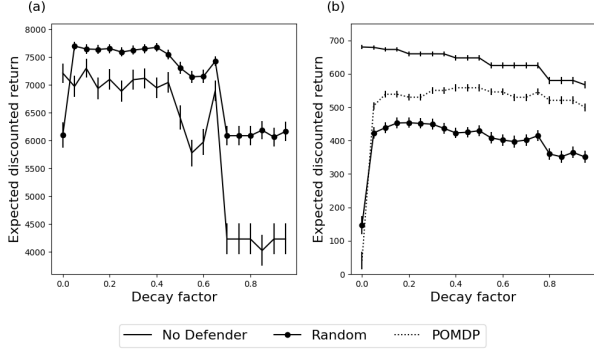
Figure 2: D-PenTesting performance, against each defender, with decay factor, at intervals of $0.05, \in [0, 0.95]$. (a) and (b) show results for scenario 1 and 2, respectively. Decay factor 0.0 is equivalent to POMDP-PenTesting.
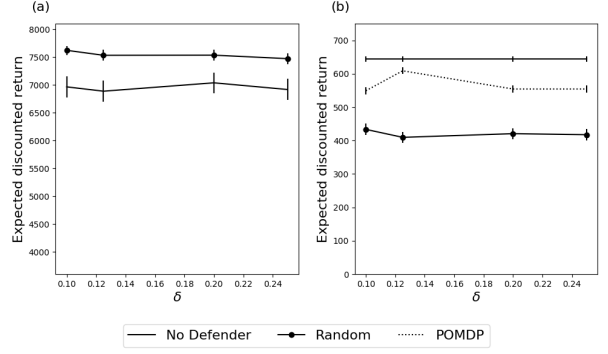

Figure 3: LD-PenTesting performance against each defender for different decay discretization resolution, δ, values (0.1, 0.125, 0.2 and 0.25). (a) and (b) show results for scenarios 1 and 2, respectively.

we set the fully observable defender action state variable of the Oracle-PenTesting agent to be the next action of the defender. This is done for all defender strategies (no defender, random and POMDP). In this way Oracle-PenTesting has full information about the defenders next action along with the effects of the action on the next state.

Unfortunately, we could only include the Oracle-PenTesting agent as a comparison for scenario 2. For scenario 1 we were unable to create a POMDP defender, as it would require significant modifications to the underlying problem formulation, which causes comparison with the original approaches to become infeasible.

Parameter Selection

D-PenTesting We evaluated the performance of D-PenTesting using different decay factors, against each defender strategy, on both scenarios. We ran experiments for multiple decay factors $\in [0, 0.95]$ at an interval of 0.05. For these experiments, POMDP-PenTesting is equivalent to when the decay factor is 0.0. Figure 2 shows the expected discounted return of D-PenTesting for both scenarios. Tables of full results are supplied in the supplementary materials.

When no defender is present, for scenario 1 performance of D-PenTesting is consistent up to a decay factor of 0.45, with performance declining significantly for decay factors above 0.45. For scenario 2 with no defender, the percentage of simulations where the pen-tester reached the goal (or win rate) is 100% for all tested decay factors. However, the performance in terms of expected discount return declines as the decay rate increases. The decline is quite slow when the decay factor is below 0.5, with decline occurring faster for higher decay factors.

Against the random defender, for both scenarios, we see a similar trend to when no defender is present. The expected discounted return is fairly constant up to a decay factor of around 0.5, before it begins to decline more rapidly.

This trend in performance can be attributed to overly conservative behaviour of D-PenTesting for high decay factors. We have observed that for larger decay factors ($>= 0.5$),

where the agent expects faster changes to the state, the D-PenTesting agent becomes increasingly conservative resulting in many repeated actions. This results in many wasted actions and a decline in performance. For scenario 1, for decay factors $>= 0.7$, this even resulted in agents repeatedly performing the same subset of actions leaving them unable to successfully control the host in the cases when the host was not vulnerable to the subset of actions performed.

LD-PenTesting To evaluate LD-PenTesting, we ran experiments using four different decay discretization resolution values of δ: 0.1, 0.125, 0.2 and 0.25. These were chosen based on the results from D-PenTesting, where we observed that performance for both scenarios tended to be consistent over decay factor intervals of 0.1 or larger.

Figure 3 shows the performance of LD-PenTesting for both scenarios. The expected discounted return was relatively consistent across the different δ values, against all defender strategies, across both scenarios. This indicates that at least for the scenarios tested, the δ has little effect on the possible performance.

Parameter Sensitivity Based on the results shown in figures 2 and 3, we can see that both D-PenTesting and LD-PenTesting are quite insensitive to parameter selection. For D-PenTesting, performance was reasonably consistent for decay factors $\in [0.05, 0.45]$. For LD-PenTesting, performance was consistent for each δ value tested.

Pen-Tester Comparison

Figure 4 shows a comparison of the performance of each pen-tester agent for both scenarios. For D-PenTesting we chose the best decay factor for each defender strategy (scenario 1: no defender = 0.1, random = 0.05, and scenario 2: no defender = 0.05, random = 0.2, POMDP = 0.45). For LD-PenTesting we show results for the agent with $\delta = 0.1$.

Against no defender, performance of LD-PenTesting is slightly worse than that of the best D-PenTesting agent, which can be explained by the cost of learning in LD-PenTesting. When there is no defender the correct decay fac-

tor is 0.0. For the LD-PenTesting agent it cannot observe the decay factor directly and instead infers it through the transition dynamics, hence it will always have some uncertainty over the correct value. This uncertainty incurs a learning cost in the form of more repeated actions. This cost is emphasised when the planning horizon is longer; as it is in scenario 2 where the difference in performance is most notable. However, even with this extra cost, against no defender, the expected discounted return of LD-PenTesting is only slightly lower than that of D-PenTesting and POMDP-PenTesting and win rate is comparable across all three agents for both scenarios.

For the random and POMDP defenders, performance was comparable between D-PenTesting and LD-PenTesting. The consistency of performance between these two approaches can be explained by their insensitivity to parameter selection. This meant that learning a good decay factor would be relatively easy for the LD-PenTesting agent since the range of good decay factors is quite large.

When compared with POMDP-PenTesting, when there is no defender, we see comparable performance with D-PenTesting and LD-PenTesting, except for scenario 2 where the performance of LD-PenTesting is slightly worse (discussed above). Against the random and POMDP defenders D-PenTesting and LD-PenTesting significantly outperform POMDP-PenTesting. This illustrates the superior advantage D-PenTesting and LD-PenTesting have over POMDP-PenTesting in the presence of a defender and dynamic changes to the state.

When compared with Oracle-PenTesting we see that, in terms of expected discounted return, D-PenTesting and LD-PenTesting outperform Oracle-PenTesting against the random defender. When we incorporate the defender policy into the Oracle-PenTesting agent we essentially fit the pen-tester agent to a specific defender policy, hence why the Oracle-PenTesting agent performs significantly worse against the random defender than against the POMDP defender (i.e. the policy it had planned with is wrong). The better performance of D-PenTesting and LD-PenTesting versus Oracle-PenTesting, against a random defender, suggests that D-PenTesting and LD-PenTesting agents can be more robust to different defender behaviour than an agent designed to counter a specific defender strategy.

As expected, Oracle-PenTesting outperforms D-PenTesting and LD-PenTesting against the POMDP defender. However, both D-PenTesting and LD-PenTesting are still able to achieve a 100% win rate and a good expected discounted return that is significantly higher than that of POMDP-PenTesting. This indicate D-PenTesting and LD-PenTesting are able to perform well against a reasonable defender strategy. This further supports that incorporating decay into the pen-tester model can help improve agent robustness against different defender strategies.

Summary

This paper presented POMDP-based pen-testing models, D-PenTesting and LD-PenTesting. They have been designed to handle the three main sources of uncertainty in autonomous pen-testing: partial observability of the network and its vulnerabilities, lack of reliability of pen-testing tools, and possible changes triggered by the defender. Based on an observation that a pen-tester's knowledge about the defender's actions and strategies are gained in an indirect manner, via the network, we abstract the defender's actions into two types: Network analysis (which does not change the network) and defensive cyber operations (which alter the network properties). This abstraction enables us to model the defender's behaviour as a single variable, namely the decay factor. D-PenTesting incorporates this variable in its transition function, assuming the decay factor is known a priori. LD-PenTesting learns the decay factor as it attempts to break into the network, by framing the problem as BRL, solved as yet another POMDP problem. Experiments on two benchmark scenarios indicate D-PenTesting and LD-PenTesting outperform the POMDP-based pen-testing approach that does not account for the defender's behaviour and is more robust than autonomous pen-testing that assumes a POMDP-based defender.
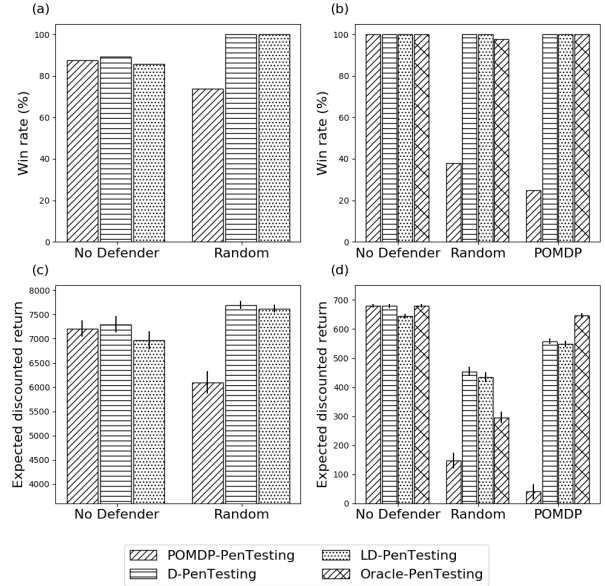
Future work abounds. This work indicates that even a simple defender's model can substantially improve the capability of autonomous pen-testing. We are interested in better understanding the limits of this approach by testing it against more powerful defenders (Ahmadi et al. 2018). Another avenue is to apply the proposed concept to on-line POMDP solvers for scalability. Last but not least is to remove assumptions, such as the independence and identical capability of the defender in assessing different network properties.

Figure 4: Comparison between the different pen-testing agents for scenarios 1 (a and c) and 2 (b and d). (a) and (b) show the win rate, and (c) and (d) show expected discounted return ($\pm$ 95% CI), for each pen-tester agent against each defender strategy.

Supplementary Materials

Acknowledgement

References

Ahmadi, M.; Cubuktepe, M.; Jansen, N.; Junges, S.; Katoen, J.-P.; and Topcu, U. 2018. The partially observable games we play for cyber deception.

Alpcan, T., and Basar, T. 2006. An intrusion detection game with limited observations. In *12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis, France*, volume 26. Citeseer.

Ammann, P.; Wijesekera, D.; and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 217–224. ACM.

Asmussen, S. 2010. Applied probability and queues (stochastic modeling and applied probability series vol. 51).

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21.

Doshi, P., and Gmytrasiewicz, P. J. 2009. Monte carlo sampling methods for approximating interactive pomdps. *Journal of Artificial Intelligence Research* 34:297–337.

Du, Y.; S.W.Png; H.Kurniawati; S.C.W.Ong; W.S.Lee; and D.Hsu. 2014. Approximate POMDP planning toolkit (APPL). http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/.

Durkota, K., and Lisỳ, V. 2014. Computing optimal policies for attack graphs with action failures and costs. In *STAIRS*, 101–110.

Ghavamzadeh, M.; Mannor, S.; Pineau, J.; Tamar, A.; et al. 2015. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning* 8(5-6):359–483.

Gmytrasiewicz, P. J., and Doshi, P. 2004. Interactive pomdps: Properties and preliminary results. In *International Conference on Autonomous Agents: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-*, volume 3, 1374–1375.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, 709–715.

Hoang, T. N., and Low, K. H. 2013. Interactive pomdp lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Hoffmann, J. 2015. Simulated penetration testing: From "dijkstra" to "turing test++". In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

Ibe, O. 2013. *Markov processes for stochastic modeling*. Newnes.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1-2):99–134.

Kurniawati, H., and Yadav, V. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *Proc. Int. Symp. on Robotics Research*.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland.

Liang, X., and Xiao, Y. 2012. Game theory for network security. *IEEE Communications Surveys & Tutorials* 15(1):472–486.

Lippmann, R. P., and Ingols, K. W. 2005. An annotated review of past papers on attack graphs. Technical report, Massachusetts Ins. of Tech. Lexington Lincoln Lab.

Lucangeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *Workshop on Intelligent Security (SecArt 2010)*.

Lye, K.-w., and Wing, J. M. 2005. Game strategies in network security. *International Journal of Information Security* 4(1-2):71–86.

Merrick, K.; Hardhienata, M.; Shafi, K.; and Hu, J. 2016. A survey of game theoretic approaches to modelling decision-making in information warfare scenarios. *Future Internet* 8(3):34.

Neuts, M. F. 1979. A versatile markovian point process. *Journal of Applied Probability* 16(4):764–779.

Nguyen, K. C.; Alpcan, T.; and Basar, T. 2009. Security games with incomplete information. In *2009 IEEE International Conference on Communications*, 1–6. IEEE.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2011. Penetration testing== pomdp solving? In *Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)*, 66.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. Pomdps make better hackers: Accounting for uncertainty in penetration testing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Shmaryahu, D.; Shani, G.; Hoffmann, J.; and Steinmetz, M. 2018. Simulated penetration testing as contingent planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, 2164–2172.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. Despot: Online pomdp planning with regularization. In *Advances in neural information processing systems*, 1772–1780.

Swiler, L. P., and Phillips, C. 1998. A graph-based system for network-vulnerability analysis. Technical report, Sandia National Labs., Albuquerque, NM (United States).

Supplementary Materials

Experiment Scenario 1 State and Action Details

Table 2: Pen-tester attack actions for scenario 1. The target columns show the OS and port number that each attack action targets (i.e. the attack action preconditions).

Attack Action	Target OS	Target Port
vsftpd-234	linux	21
IIS-ftp	windows	21
openBDSD-ssh	openBSD	22
IIS-http	windows	80
apache-openBSD	openBSD	80
rsh	linux	512
unreal-ircd-3281	linux	3632
manageEngine9	windows	8020
tomcat	windows	8282
elastic-search	windows	9200

Experiment Scenario Planning Time

Table 3: Planning times for pen-testing agents. The Decay column shows the decay factor used for each D-PenTesting agent and the decay discretization resolution, δ, for each LD-PenTesting agent. The solver time limit was 3600 seconds and target precision was, $\epsilon = 0.001$.

Agent	Decay, δ	Scenario 1		Scenario 2	
		Time (s)	Precision	Time (s)	Precision
POMDP-PenTesting	0.0	555.03	< 0.001	0.02	< 0.001
Oracle-PenTesting	0.0	NA	NA	0.18	< 0.001
D-PenTesting	0.05	3614.52	87.109	3.58	< 0.001
D-PenTesting	0.1	3609.87	82.141	8.30	< 0.001
D-PenTesting	0.15	3617.7	74.307	6.00	< 0.001
D-PenTesting	0.2	3631	62.045	3.68	< 0.001
D-PenTesting	0.25	3616.78	35.675	2.34	< 0.001
D-PenTesting	0.3	3620.12	4.521	1.95	< 0.001
D-PenTesting	0.35	3619.38	0.682	2.62	< 0.001
D-PenTesting	0.4	3619.43	0.031	1.34	< 0.001
D-PenTesting	0.45	717.83	< 0.001	0.80	< 0.001
D-PenTesting	0.5	311.89	< 0.001	0.61	< 0.001
D-PenTesting	0.55	367.95	< 0.001	0.51	< 0.001
D-PenTesting	0.6	3601.54	0.002	0.23	< 0.001
D-PenTesting	0.65	3606.03	0.960	0.19	< 0.001
D-PenTesting	0.7	3601.64	0.003	0.23	< 0.001
D-PenTesting	0.75	912.37	< 0.001	0.18	< 0.001
D-PenTesting	0.8	3007.85	< 0.001	0.08	< 0.001
D-PenTesting	0.85	3669.95	13.386	0.11	< 0.001
D-PenTesting	0.9	3620.07	43.992	0.06	< 0.001
D-PenTesting	0.95	3608.78	52.290	0.07	< 0.001
LD-PenTesting	0.1	3600.72	400.171	3684.03	0.803
LD-PenTesting	0.125	3624.62	396.710	3753.75	0.757
LD-PenTesting	0.2	3688.45	359.444	3688.32	0.595
LD-PenTesting	0.25	3603.53	345.460	3697.02	0.516

Detailed Results for D-PenTesting and LD-PenTesting with Different Parameters

Table 4: Full performance results for pen-testing agents for experiment scenario 1. The Decay column shows the decay factor used for each D-PenTesting agent and the decay discretization resolution, δ, for each LD-PenTesting agent. Results are from running 1000 simulations, each for a max length of 100 steps. We show the expected discounted return, with $\gamma = 0.95$

Agent	Decay, δ	No Defender		Random Defender	
		Expected Return $\pm$ 95% CI	Win %	Expected Return $\pm$ 95% CI	Win %
POMDP-PenTesting	0.0	7211.85 $\pm$ 171.71	87.50	6097.78 $\pm$ 228.47	73.90
D-PenTesting	0.05	6972.84 $\pm$ 192.42	85.30	7698.92 $\pm$ 75.94	100.00
D-PenTesting	0.1	7299.51 $\pm$ 168.94	89.30	7646.51 $\pm$ 83.66	100.00
D-PenTesting	0.15	6936.61 $\pm$ 192.69	85.20	7638.61 $\pm$ 81.33	100.00
D-PenTesting	0.2	7101.24 $\pm$ 182.04	87.10	7654.13 $\pm$ 81.69	100.00
D-PenTesting	0.25	6886.62 $\pm$ 192.93	85.30	7591.75 $\pm$ 81.50	100.00
D-PenTesting	0.3	7093.86 $\pm$ 182.19	87.30	7625.24 $\pm$ 80.33	100.00
D-PenTesting	0.35	7119.47 $\pm$ 176.54	88.10	7650.80 $\pm$ 83.94	100.00
D-PenTesting	0.4	6948.46 $\pm$ 189.81	85.90	7675.21 $\pm$ 76.69	100.00
D-PenTesting	0.45	7045.95 $\pm$ 185.65	86.70	7548.12 $\pm$ 86.35	100.00
D-PenTesting	0.5	6416.93 $\pm$ 216.28	80.50	7307.58 $\pm$ 104.12	100.00
D-PenTesting	0.55	5778.01 $\pm$ 237.54	73.10	7146.50 $\pm$ 106.20	100.00
D-PenTesting	0.6	5976.65 $\pm$ 229.85	75.60	7155.58 $\pm$ 115.28	100.00
D-PenTesting	0.65	6895.90 $\pm$ 184.12	86.90	7424.49 $\pm$ 88.17	100.00
D-PenTesting	0.7	4231.06 $\pm$ 277.61	52.50	6088.01 $\pm$ 171.96	100.00
D-PenTesting	0.75	4231.06 $\pm$ 277.61	52.50	6088.01 $\pm$ 171.96	100.00
D-PenTesting	0.8	4231.06 $\pm$ 277.61	52.50	6088.01 $\pm$ 171.96	100.00
D-PenTesting	0.85	4024.77 $\pm$ 277.93	50.20	6188.24 $\pm$ 165.77	100.00
D-PenTesting	0.9	4231.06 $\pm$ 277.61	52.50	6063.02 $\pm$ 170.51	100.00
D-PenTesting	0.95	4231.06 $\pm$ 277.61	52.50	6164.43 $\pm$ 171.53	99.90
LD-PenTesting	0.1	6965.32 $\pm$ 186.32	85.70	7622.19 $\pm$ 81.35	99.90
LD-PenTesting	0.125	6887.69 $\pm$ 191.91	84.70	7532.83 $\pm$ 97.58	98.90
LD-PenTesting	0.2	7038.24 $\pm$ 185.49	86.10	7535.30 $\pm$ 98.46	98.80
LD-PenTesting	0.25	6918.08 $\pm$ 190.81	84.90	7472.56 $\pm$ 98.56	98.60

Table 5: Full performance results for pen-testing agents for experiment scenario 2. The Decay column shows the decay factor used for each D-PenTesting agent and the decay discretization resolution, δ, for each LD-PenTesting agent. Results are from running 1000 simulations, each for a max length of 100 steps. We show the expected discounted return, with $\gamma = 0.95$

| Agent | Decay, δ | No Defender | | Random Defender | | POMDP Defender | |
		Expected Return (95% CI)	Win rate (%)	Expected Return (95% CI)	Win rate (%)	Expected Return (95% CI)	Win rate (%)
POMDP-PenTesting	0.0	680.27 ± 11.62	100.00	146.57 ± 55.04	37.90	40.30 ± 51.68	24.70
Oracle-PenTesting	0.0	680.27 ± 11.62	100.00	295.45 ± 39.64	97.60	646.94 ± 15.37	100.00
D-PenTesting	0.05	679.10 ± 12.66	100.00	422.81 ± 33.65	99.90	504.43 ± 21.68	100.00
D-PenTesting	0.1	673.01 ± 13.16	100.00	438.89 ± 32.53	100.00	538.84 ± 20.03	100.00
D-PenTesting	0.15	673.01 ± 13.16	100.00	452.73 ± 31.72	100.00	538.84 ± 20.03	100.00
D-PenTesting	0.2	659.92 ± 15.25	100.00	453.54 ± 31.46	99.90	529.98 ± 20.50	100.00
D-PenTesting	0.25	659.92 ± 15.25	100.00	451.37 ± 30.89	100.00	529.98 ± 20.50	100.00
D-PenTesting	0.3	659.92 ± 15.25	100.00	449.13 ± 31.38	100.00	550.62 ± 19.86	100.00
D-PenTesting	0.35	659.66 ± 15.14	100.00	436.52 ± 32.59	100.00	550.01 ± 19.80	100.00
D-PenTesting	0.4	647.66 ± 16.83	100.00	423.03 ± 32.21	99.80	558.15 ± 20.15	100.00
D-PenTesting	0.45	647.66 ± 16.83	100.00	425.08 ± 31.94	100.00	558.15 ± 20.15	100.00
D-PenTesting	0.5	647.66 ± 16.83	100.00	429.71 ± 31.93	99.70	558.15 ± 20.15	100.00
D-PenTesting	0.55	624.82 ± 17.35	100.00	408.23 ± 33.74	100.00	545.68 ± 20.33	100.00
D-PenTesting	0.6	624.82 ± 17.35	100.00	401.45 ± 33.87	100.00	545.68 ± 20.33	100.00
D-PenTesting	0.65	624.82 ± 17.35	100.00	397.10 ± 34.96	99.90	529.77 ± 21.70	100.00
D-PenTesting	0.7	624.82 ± 17.35	100.00	401.58 ± 34.14	99.90	529.77 ± 21.70	100.00
D-PenTesting	0.75	624.82 ± 17.35	100.00	414.68 ± 33.68	100.00	545.68 ± 20.33	100.00
D-PenTesting	0.8	580.10 ± 21.87	100.00	359.93 ± 34.77	99.60	520.68 ± 23.25	100.00
D-PenTesting	0.85	580.10 ± 21.87	100.00	351.52 ± 34.99	99.70	520.68 ± 23.25	100.00
D-PenTesting	0.9	580.10 ± 21.87	100.00	364.62 ± 34.66	99.70	520.68 ± 23.25	100.00
D-PenTesting	0.95	567.31 ± 22.18	100.00	351.12 ± 35.91	99.80	499.40 ± 24.04	100.00
LD-PenTesting	0.1	644.49 ± 8.50	100.00	433.55 ± 16.80	100.00	549.27 ± 10.45	100.00
LD-PenTesting	0.125	644.49 ± 8.50	100.00	409.74 ± 16.71	99.70	608.99 ± 9.99	100.00
LD-PenTesting	0.2	644.54 ± 8.51	100.00	420.76 ± 16.58	100.00	554.49 ± 10.28	100.00
LD-PenTesting	0.25	644.54 ± 8.51	100.00	417.61 ± 17.16	99.90	554.51 ± 10.28	100.00