Using Reinforcement Learning to Conceal Honeypot Functionality

Seamus Dowling[1(✉)], Michael Schukat[2], and Enda Barrett[2]

[1] Galway Mayo Institute of Technology, Castlebar, Mayo, Ireland
seamus.dowling@gmit.ie
[2] National University of Ireland Galway, Galway, Ireland

Abstract. Automated malware employ honeypot detecting mechanisms within its code. Once honeypot functionality has been exposed, malware such as botnets will cease the attempted compromise. Subsequent malware variants employ similar techniques to evade detection by known honeypots. This reduces the potential size of a captured dataset and subsequent analysis. This paper presents findings on the deployment of a honeypot using reinforcement learning, to conceal functionality. The adaptive honeypot learns the best responses to overcome initial detection attempts by implementing a reward function with the goal of maximising attacker command transitions. The paper demonstrates that the honeypot quickly identifies the best response to overcome initial detection and subsequently increases attack command transitions. It also examines the structure of a captured botnet and charts the learning evolution of the honeypot for repetitive automated malware. Finally it suggests changes to an existing taxonomy governing honeypot development, based on the learning evolution of the adaptive honeypot. Code related to this paper is available at: https://github.com/sosdow/RLHPot.

Keywords: Reinforcement learning · Honeypot · Adaptive

1 Introduction

Honeypots have evolved to match emerging threats. From "packets found on the internet" in 1993 [1] to targeting and capturing IoT attacks [2], honeypot development has become a cyclic process. Malware captured on a honeypot is retrospectively analysed. This analysis informs defence hardening and subsequent honeypot redevelopment. Because of this, honeypot contribution to security is considered a reactive process. The value of honeypot deployment comes from the captured dataset. The longer an attack interaction can be maintained, the larger the dataset and subsequent analysis. Global honeypot projects track emerging threats [3]. Virtualisation technologies provide honeypot operators with the means of abstracting deployments from production networks and bare metal infrastructure [4]. To counter honeypot popularity, honeypot detection tools were developed and detection techniques were incorporated into malware deployments [5]. These have the effect of ending an attempted compromise

U. Brefeld et al. (Eds.): ECML PKDD 2018, LNAI 11053, pp. 341–355, 2019.
https://doi.org/10.1007/978-3-030-10997-4_21

once it is discovered to be a honeypot. The only solution is to modify the honeypot or incorporate a fix into a new release or patch. This is a cyclic process. The rapid mutation of malware variants often makes this process redundant. For example, the recent Mirai bot [6] exploited vulnerabilities in Internet of Things (IoT) deployments. IoT devices are often constrained reduced function devices (RFD). Security measures can be restricted and provide an opportunity for compromise. When the Mirai bot source code was made public, it spawned multiple variants targeting different attack vectors. Bots and their variants are highly automated. Human social engineering may play a part in how the end devices are compromised. Botmasters communicate with command and control (C&C) to send instructions to the compromised hosts. But to generate and communicate with a global botnet of compromised hosts, malware employs highly automated methods. Coupled with honeypot detection techniques, it becomes an impossible task for honeypot developers to release newer versions and patches to cater for automated malware and their variants. This paper describes a reinforcement learning approach to creating adaptive honeypots that can learn the best responses to attack commands. In doing so it overcomes honeypot detection techniques employed by malware. The adaptive honeypot learns from repetitive, automated compromise attempts. We implement a state action space formalism designed to reward the learner for prolonging attack interaction. We present findings on a live deployment of the adaptive honeypot on the Internet. We demonstrate that after an initial learning period, the honeypot captures a larger dataset with four times more attack command transitions when compared with a standard high interaction honeypot. Thus we propose the following contributions over existing work:

- Whilst the application of reinforcement learning to this domain is not new, previous developments facilitated human attack interactions which is not relevant for automated malware. Our proposed state action space formalism is new, targeting automated malware and forms a core contribution of this work.
- Presents findings on a live deployment with a state action space formalism demonstrating intelligent decision making, overcoming initial honeypot detection techniques.
- Demonstrates that using the proposed state action space formalism for automated malware produces a larger dataset with four times more command transitions compared to a standard high interaction honeypot.
- Demonstrates that our proposed state action space formalism is more effective than similar deployments. Previous adaptive honeypots using reinforcement learning collected three times more command transitions when compared to a high interaction honeypot.
- Demonstrates that the collected dataset can be used in a controlled environment for policy evaluation, which results in a more effective honeypot deployment.

This article is organised as follows:

Section 2 presents the evolution of honeypots and the taxonomy governing their development. This section also addresses honeypot detection measures used by malware and the use of reinforcement learning to create adaptive honeypots capable of learning from attack interactions.

Section 3 outlines the implementation of reinforcement learning within a honeypot, creating an adaptive honeypot. It rewards the learning agent for overcoming detection attempts and prolonging interaction with automated malware.

Section 4 presents findings from two experiments. The first experiment is a live deployment of the adaptive honeypot on the Internet, which captures automated attack interactions. The second is a controlled experiment, which evaluates the learning policies used by the adaptive honeypot.

Sections 5 and 6 discuss the limitations of the adaptive honeypot model that provides for future work in this area and our conclusions.

2 Related Work

2.1 Honeypot Evolution

In 2003, Spitzner defined a honeypot as a "security resource whose value lies in being probed, attacked or compromised" [7]. After a decade of monitoring malicious traffic [1], honeypots were seen as a key component in cybersecurity. Capturing a successful compromise and analysing their methods, security companies can harden subsequent defences. This cycle of capture-analyse-harden-capture, is seen as a reactive process. Virtualisation allowed for the easy deployment of honeypots and gave rise to popular deployments such as Honeyd [4]. This increase in honeypot popularity raised the question of their role. To address this, Zhang [8] introduced honeypot taxonomy. This taxonomy identifies security as the role or class of a honeypot, and could have prevention, detection, reaction or research as values. Deflecting an attacker away from a production network, by luring them into a honeypot, has a prevention value. Unauthorized activity on a honeypot is red flagged immediately providing a detection value. Designing a honeypot to maintain an attackers interest, by offering choices or tokens [9] has a reaction value. Finally the research value gives insights into the behaviour and motivation of an attacker. As more devices connected and threats evolved on the Internet, Seifert [10] introduced a new taxonomy, a summery of which is displayed in Table 1. The possible combinations for class/value from Table 1 informed the development of complex honeypots. The purpose of honeypots developed under this taxonomy is to collect data for the retrospective analysis of malware methods. However, firewalls and intrusion prevention and detection systems (IPS, IDS) became standard IT infrastructure for proactively limiting malware infections. Therefore honeypots were briefly considered as alternatives to IDS and IPS [11,12].

Table 1. Seifert's taxonomy for honeypot development [10]

Class	Value	Note
Interaction level	High	High degree of functionality
	Low	Low degree of functionality
Data capture	Events	Collect data about changes in state
	Attacks	Collect data about malicious activity
	Intrusions	Collect data about security compromises
	None	Do not collect data
Containment	Block	Identify and block malicious activity
	Diffuse	Identify and mitigate against malicious activity
	Slow down	Identify and hinder malicious activity
	None	No action taken
Distribution appearance	Distributed	Honeypot is or appears to be composed of multiple systems
	Standalone	Honeypot is or appears to be one system
Communications interface	Network interface	Directly communicated with via a NIC
	Non-network interface	Directly communicated with via interface other than NIC (USB etc.)
	Software API	Honeypot can be interacted with via a software API (SSH, HTTP etc.)
Role in multi-tiered	Client	Honeypot acts as a server
Architecture	Server	Honeypot acts as a client

2.2 Anti-honeypot and Anti-detection

Honeypots can be deployed simply and quickly with virtualization tools and cloud services. Seiferts taxonomy from Table 1 provides the framework to create modern, relevant honeypots. For example, ConPot is a SCADA honeypot developed for critical industrial IoT architectures [13]; IoTPot is a bespoke honeypot designed to analyze malware attacks targeting IoT devices [2]. Botnets provide a mechanism for global propagation of cyber attack infection and control. They are under the control of a single C&C [14]. A typical botnet attack will consist of 2 sets of IP addresses. The first set of IPs is the compromised hosts. These are everyday compromised machines that are inadvertently participating in an attack. The second set of IPs is the C&C reporters and software

loaders. These are the sources from which the desired malware is downloaded. There is a multitude of communication channels opened between C&C, report and loader servers, bot victims and target. The main methods of communication are IRC (Internet Relay Chat), HTTP and P2P based models where bots use peer-to-peer communications. Botmasters obfuscate visibility by changing the C&C connection channel. They use Dynamic DNS (DDNS) for botnet communication, allowing them to shut down a C&C server on discovery and start up a new server for uninterrupted attack service. Malware developers became aware of the existence honeypots, capturing and analyzing attack activity. To counter this, dedicated honeypot detection tools were developed and evasion techniques we designed into malware [15]. Tools such as Honeypot Hunter performed tests to identify a honeypot [5]. False services were created and connected to by the anti-honeypot tool. Honeypots are predominately designed to prolong attacker interaction and therefore pretend to facilitate the creation and execution of these false services. This immediately tags the system as a honeypot. With the use of virtualization for honeypot deployments [4], anti-detection techniques issued simple kernel commands to identify the presence of virtual infrastructure instead of bare metal [16]. New versions of honeypots are redesigned and redeployed continuously to counter new malware methods. More recently, the Mirai botnet spawned multiple variants targeting IoT devices through various attack vectors [6]. Honeypots played an active part in capturing and analyzing the Mirai structure [17]. Variants were captured on Cowrie, a newer version of the popular Kippo honeypot [18]. Analysis of the Mirai variants found that Cowrie failed initial anti-honeypot tests before honeypot functionality was manually amended [19]. Examining the structure of the Mirai variant [20], when a "mount" command is issued, the honeypot returns it's standard response at which point the attack ends. It indicates that the variant is implementing anti-detection techniques. Virtual machine (VM) aware botnets, such as Conficker and Spybot scan for the presence of a virtualized environment and can refuse to continue or modify its methods [21].

2.3 Adaptive Honeypots

Machine learning techniques have previously been applied to honeypots. These techniques have analysed attacker behaviour retrospectively on a captured dataset. Supervised and unsupervised methods are used to model malware interaction and to classify attacks [22–24]. This analysis is a valuable source of information for security groups. However, by proactively engaging with an attacker, a honeypot can prolong interaction and capture larger datasets potentially leading to better analysis. To this end, the creation of intelligent, adaptive honeypots has been explored. Wagener [25] uses reinforcement learning to extract as much information as possible from the intruder. A honeypot called Heliza was developed to use reinforcement learning when engaging an attacker. The honeypot implemented behavioural strategies such as blocking commands, returning error messages and issuing insults. Previously, he had proposed the use of game theory [26] to define the reactive action of a honeypot towards attacker's

behaviour. Pauna [27] also presents an adaptive honeypot using the same reinforced learning algorithms and parameters as Heliza. He proposes a honeypot named RASSH that provides improvements to scalability, localisation and learning capabilities. It does this by using newer libraries with a Kippo honeypot and delaying responses to frustrate human attackers. Both Heliza and RASSH use PyBrain [28] for their implementation of reinforcement learning and use actions, such as insult and delay targeting human interaction. Whilst Heliza and RASSH implement reinforcement learning to increase command transitions from human interactions, our work pursues the goal of increasing command transitions to overcome automated honeypot detection techniques.

2.4 Reinforcement Learning in Honeypots

Reinforcement learning is a machine learning technique in which a learning agent learns from its environment, through trial and error interactions. Rather than being instructed as to which action it should take given a specific set of inputs, it instead learns based on previous experiences as to which action it should take in the current circumstance.

Markov Decision Processes. Reinforcement learning problems can generally be modelled using Markov Decision Processes (MDPs). In fact reinforcement learning methods facilitate solutions to MDPs in the absence of a complete environmental model. This is particularly useful when dealing with real world problems such as honeypots, as the model can often be unknown or difficult to approximate. MDPs are a particular mathematical framework suited to modelling decision making under uncertainty. A MDP can typically be represented as a four tuple consisting of states, actions, transition probabilities and rewards.

- S, represents the environmental state space;
- A, represents the total action space;
- p(.|s, a), defines a probability distribution governing state transitions $s_{t+1} \sim p(.|s_t, a_t)$;
- q(.|s, a), defines a probability distribution governing the rewards received $R(s_t, a_t) \sim q(.|s_t, a_t)$;

S the set of all possible states represents the agent's observable world. At the end of each time period t the agent occupies state $s_t \in S$. The agent must then choose an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions within state s_t. The execution of the chosen action, results in a state transition to s_{t+1} and an immediate numerical reward $R(s_t, a_t)$. Formula (1) represents the reward function, defining the environmental distribution of rewards. The learning agent's objective is to optimize its expected long-term discounted reward.

$$R_a s, s' = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{1}$$

The state transition probability $p(s_{t+1}|s_t, a_t)$ governs the likelihood that the agent will transition to state s_{t+1} as a result of choosing a_t in s_t.

$$P_a s, s' = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2}$$

The numerical reward received upon arrival at the next state is governed by a probability distribution $q(s_{t+1}|s_t, a_t)$ and is indicative as to the benefit of choosing a_t whilst in s_t. In the specific case where a complete environmental model is known, i.e. (S, A, p, q) are fully observable, the problem reduces to a planning problem and can be solved using traditional dynamic programming techniques such as value iteration. However if there is no complete model available, then reinforcement learning methods have proven efficacy in solving MDPs.

SARSA Learning. During the reinforcement learning process the agent can select an action which exploits its current knowledge or it can decide to use further exploration. Reinforcement learning provides parameters to help the learning environment decide on the reward and exploration values. Figure 1 models a basic reinforcement learning interaction process. We have to consider how the model in Fig. 1, applies to adaptive honeypot development. Throughout its deployment, the honeypot is considered to be an environment with integrated reinforcement learning. We are using SSH as an access point and a simulated Linux server as a vulnerable environment. SSH is responsible for 62% of all compromise attempts [29]. Within this environment, the server has states that are examined and changed with bash scripts. Examples are iptables, wget, sudo, etc. The reinforcement learning agent can perform actions on these states such as to allow, block or substitute the execution of the scripts. The environment issues a reward to the agent for performing that action. The agent learns from this process as the honeypot is attacked and over time learns the optimum policy π^*, mapping the optimal action to be taken each time, for each state s. The learning process will eventually converge as the honeypot is rewarded for each attack episode. This temporal difference method for on-policy learning uses the transition from one state/action pair to the next state/action pair, to derive the reward. State, Action, Reward, State, Action also known as SARSA, is a common implementation of on-policy reinforcement learning (3). The reward policy Q is estimated for a given state s_t and a given action a_t. The environment is

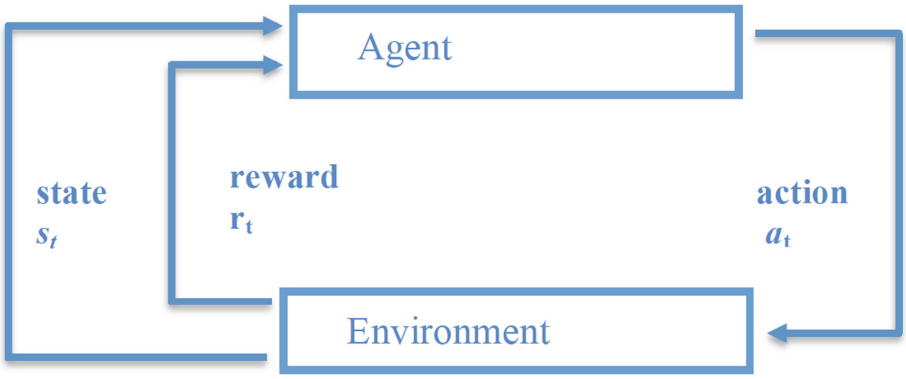

Fig. 1. Reinforcement learning model.

explored using a random component ϵ or exploited using learned Q values. The estimated Q value is expanded with a received reward r_t plus an estimated future reward $Q(s_{t+1}, a_{t+1})$, that is discounted (γ). A learning rate parameter is also applied (α).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3}$$

As each attack is considered an episode, the policy is evaluated at the end of each episode. SARSA is implemented with the following parameters:

- epsilon-greedy policy
  The honeypot environment is unknown to the learning agent. We want it to learn without prior knowledge and eventually converge. To this end, we set our explorer to ϵ-greedy.
- Discount factor $\gamma = 1$
  γ is applied when a future reward is estimated. For our honeypot no discount factor is applied, as attack episodes are readily defined commands between SSH open and close events. This allows for retrospective reward calculations.
- Step size $\alpha = 0.5$
  A step size parameter is applied for the creation of the state/action space.

3 Reinforcement Learning Honeypot Implementation

Previous contributions demonstrate that there is a relationship between honeypot and malware development and evolution. Botnet traffic is highly automated as it attacks, compromises and reports without any intervention. From a malware developer's perspective, there is very little human interaction, post launch. Honeypot evasion techniques can be implemented, as new honeypots are uncovered. This paper uses reinforcement learning to prolong interaction with an attack sequence. Initial bot commands will attempt to return known honeypot responses, or positives to false requests. Depending on the response, the bot will either cease or amend its actions. We want our honeypot to learn and be rewarded for prolonging interaction. Therefore our reward function is to increase the number of commands from the attack sequence. Attacker commands can be categorised into the following:

- L - Known Linux bash commands
  wget, cd, mount, chmod, etc.
- C - Customised attack commands
  Commands executing downloaded files
- CC - Compound commands
  Multiple commands with bash separators/operators
- NF - known commands not facilitated by honeypot
  The honeypot is configured for 75 of the 'most used' bash commands.
- O - Other commands
  Unhandled keystrokes such as ENTER and unknown commands.

We propose a transition reward function whereby the learning agent is rewarded if a bot command is an input string i, comprised of bash commands (L), customised commands (C) or compound commands (CC). Therefore Y = C ∪ L ∪ CC. For example 'iptables stop' is an input string i that transitions the state (s) of iptables on a Linux system. Other commands or commands not facilitated by the honeypot are not given any reward. We propose an action set a = allow, block, substitute. Allow and Block are realistic responses to malware behaviour. Botnets can use complex if-else structures to determine next steps during compromise [30]. Using Substitute to return an alternative response to an attack command potentially increases the number of attack transitions to newer commands. This action set coupled with state set Y, creates a discrete state/action space. The reward function is as follows:

$$r_t(s_i, a) = \begin{cases} 1, \text{if i } \in \text{ Y} \\ 0, \text{otherwise} \end{cases} \tag{4}$$

$$\text{where Y} = \text{C} \cup \text{L} \cup \text{CC}$$

The formula for transition reward r_t, based on state/action (s,a), is as follows:

- If the input string i is a customised attacker command C or a known bash command L or a compound command CC, then reward $r_t(s_i, a) = 1$
- otherwise the reward $r_t(s_i, a) = 0$

A model of the adaptive learning process from the reinforcement learning model in Fig. 1, is shown in Fig. 2.

The adaptive honeypot has the following elements:

- Modified honeypot. Cowrie is a widely used honeypot distribution that logs all SSH interactions in a MySQL database. This has been modified to generate the parameters to pass to the learning agent. Depending on the action selected by the learning agent, the honeypot will allow, block or substitute attack commands.
- SARSA agent. This module receives the required parameters from the adaptive honeypot and calculates $Q(s_t, a_t)$ as per Eq. (3). It determines the responses chosen by the adaptive honeypot and learns over time which actions yield the greatest amount of reward.

More detailed learning and reward functionality for this adaptive honeypot is available [31].

4 Results

We deployed two Cowrie honeypots at the same time; one adaptive as detailed in the previous section, and one standard high interaction. The adaptive honeypot was developed to generate rewards on 75 states and is freely available [32]. This was facilitated within PyBrain, which allows us to define the state/action

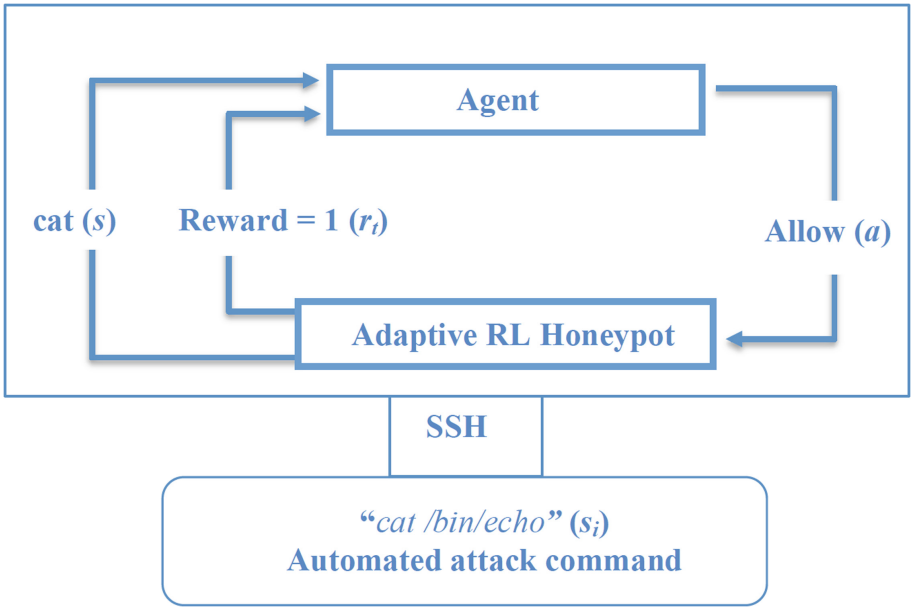

Fig. 2. Adaptive honeypot with reinforcement learning process.

space. We used Amazon Web Services (AWS) EC2 to facilitate an Internet facing honeypots. Cowrie, PyBrain, MySQL and other dependencies were installed on the adaptive honeypot EC2 instance. Both were accessible through SSH and immediately started to record malware activity. Initially it logged dictionary and bruteforce attempts. To compare performance, we undertake to extract all commands executed on the honeypots. Therefore events such as failed attempts, dictionary and bruteforce attacks are excluded as they represent pre-compromise interactions. Thereafter it captured other malware traffic including a Mirai-like bot [20]. These commands all represent interactions post-compromise. This bot became the dominant attacking tool over a 30-day period, until over 100 distinct attacks were recorded on the honeypot. Other SSH malware interacted with the honeypot. But these were too infrequent to excessively modify the rewards. Prior to presenting the results it is important to discuss the format of the bot, a sample of which is displayed in Table 2. In reality it has a sequence of 44 commands. The high interaction honeypot only ever experienced the first 8 commands in the sequence before the attack terminated. During 100 captures of the same bot, it never captured more than 8 transitions in the attack sequence. The adaptive honeypot initially experienced only the first 8 commands but then the sequence count started to increase as the honeypot learned from its state actions and rewards. Examining the entire bot structure, a mount command appears in the initial command sequence. A standard high-interaction honeypot has preconfigured parameters, and the response of mount is a known anti-evasion technique [19]. Figure 3 presents a comparison of the cumulative transitions for

Table 2. Sample of bot commands and categories

Sequence	Bot command	Category
38	/gweerwe323f	Other
39	cat/bin/echo	L
40	cd/	L
41	wget http:// <IP >/bins/usb_bus.x86 -O - >usb_bus; chmod 777 usb_bus	CC
42	./usb_bus	C

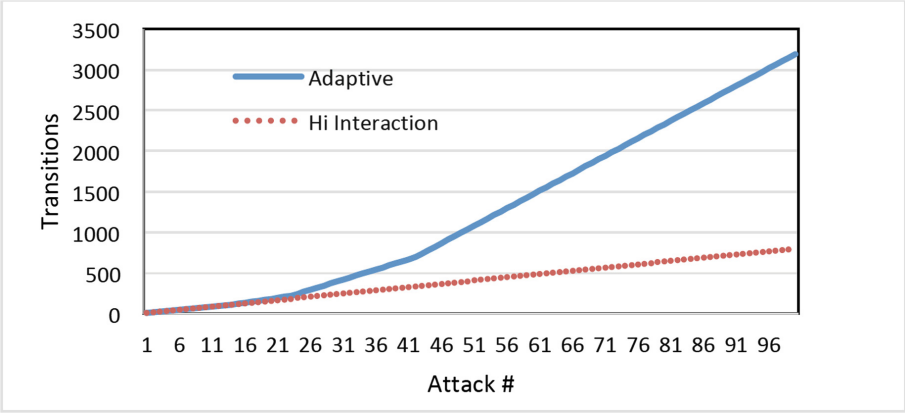

Fig. 3. Cumulative transitions for all commands.

both honeypots. This includes all command transitions for Y. At attack 16, the adaptive honeypot increased the number of commands in the sequence to 12. Examining the logs, we find that for the first time, the adaptive honeypot substituted a result for cat command and blocked a mount command. At attack 12 the honeypot blocked a compound echo command and continued to do so until attack 42. This is a very interesting result when compared to the high interaction honeypot's linear accumulation. The honeypot continued to learn until attack 42 when it allowed all compound commands in the sequence to be executed. Investigating the machinations of the bot, through sandboxing and code exploration, could explain this learning evolution. This is however beyond the scope of this article. The adaptive honeypot subsequently collected four times more command transitions than a standard high interaction honeypot. This task demonstrates that the adaptive honeypot was rewarded to increase the command transitions captured. By doing so it overcame initial honeypot identification techniques and continued to reward the learning agent in the event of further evasion measures.

Previous research has used reinforcement learning to create adaptive honeypots provisioning human interaction. Heliza presented findings from an adaptive honeypot deployed on the Internet. When compared to a high interaction

honeypot, it produced three times more transitions to attack commands after 347 successful attacks. We point out that malware is an automated process and providing actions such as insult and delay are not relevant. After 100 successful attacks, our adaptive honeypot overcame initial honeypot detection techniques and subsequently captured four times more attack commands. It demonstrates that our state action space formalism for automated malware in more effective and produces a larger dataset for analysis. The captured dataset and bot analysis are valuable elements for further honeypot research. As an example, we used the captured dataset as an input stream and considered how the honeypot can be optimised by performing policy evaluation. We implemented the adaptive honeypot within a controlled Eclipse environment and modified the explorer component using PyBrain. The learning agent uses this explorer component to determine the explorative action to be executed. PyBrain provides for the following policies [28]:

- Epsilon greedy
  A discrete explorer that mostly executes the original policy but sometimes returns a random action.
- Boltzmann Softmax
  A discrete explorer that executes actions with a probability that depends on their action values.
- State dependent
  A continuous explorer that disrupts the resulting action with added, distributed random noise.

We ran the attack sequence for each of the policies and found that the Boltzmann explorer performed better in the controlled environment, resulting in more command transitions (Fig. 4). Boltzmann learned the best action to overcome detection at attack 11. State dependent learned the best action to overcome detection at attack 52. Epsilon greedy was the least effective as it learned the best action to overcome detection at attack 67. This policy evaluation demonstrates that parameters within the learning environment can be optimised to inform the development and deployment of more effective adaptive honeypots.

5 Limitations and Future Work

This paper improves upon an existing RL implementation by reducing the action set and simplifying the reward. By doing so it targets automated attack methods using SSH as a popular attack vector [23]. It is important to note that all interactions captured on both the high-interaction and adaptive honeypots were automated. There were no interactions showing obvious human cognition such as delayed responses, typing errors etc. This was verified by the timestamps of an entire attack sequence being executed in seconds. However, there are many honeypots deployed to capture malware using known vulnerabilities at different application and protocol levels [33]. There also exists tools to detect honeypots deployed on various attack vectors [5]. Therefore our research can be considered to have a narrow focus. We have presented previous and current adaptive

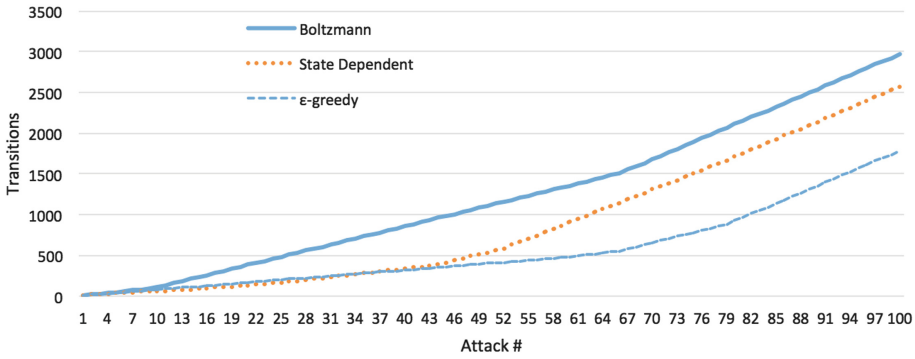

Fig. 4. Cumulative transitions for policy evaluation

honeypots targeting human and automated malware. We have also demonstrated that reinforcement learning can overcome initial detection honeypot techniques and prolong attack interaction. It is not unreasonable to consider a similar approach for application and protocol dependent honeypots. Further research into adaptive honeypot deployment could incorporate machine learning libraries such as PyBrain into these application and protocol dependent honeypots. Prior to pursuing this future work, research is required to determine if malware targeting these other honeypots is predominately human or automated. Honeypot operators have the tools to target human [25] or automated [31] attack traffic. Thereafter, adaptive honeypots using reinforcement learning can facilitate detection evasion. This ultimately leads to prolonged attack engagement and the capture of more valuable datasets.

Table 3. Modified Seifert's taxonomy

Class	Value	Note
Interaction level	High	High degree of functionality
	Low	Low degree of functionality
	Adaptive	*Learns from attack interaction*

6 Conclusions

This paper presents a state action space formalism designed to overcome anti detection techniques used by malware to evade honeypots. The reinforcement learning function is configured to reward the honeypot for increasing malware interaction. After an initial period, the honeypot learned the best method to avoid an attack command known to discover the presence of a honeypot. A standard high interaction honeypot always failed to maintain an attack beyond 8 commands. This indicated that the bot detected a honeypot and ceased operations. Our adaptive honeypot learned how to overcome this initial detection,

and cumulatively collected four times more command transitions. Honeypots are designed to capture malware for retrospective analysis, which helps understand attack methods and identify zero day attacks. Once discovered, malware developers modify attack methods to evade detection and release them as new variants. By modifying parameters within PyBrain we were able evaluate learning policies. The collected dataset is a valuable tool and can be used as an input stream into honeypots in a controlled environment. It allows us to ascertain the best policy for future deployments. Continuous capture and analysis using this method can ensure that more effective adaptive honeypots are deployed for new malware variants. IoT, ICS, and wireless sensor networks are example of where RFDs are targeted by new malware. Rather than releasing new versions of honeypots when discovered, an adaptive honeypot can learn how to evade the anti detection methods used. It gives security developers an advantage in the cat-and-mouse game of malware development and discovery. For future work it is incumbent on honeypot developers to revisit Seifert's taxonomy in Table 1 and consider adding to interaction level class and value, a sample of which is shown in Table 3.

References

1. Bellovin, S.M.: Packets found on an internet. ACM SIGCOMM Comput. Commun. Rev. **23**(3), 26–31 (1993)
2. Pa, Y.M.P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., Rossow, C.: IoTPOT: analysing the rise of IoT compromises. EMU **9**, 1 (2015)
3. Watson, D., Riden, J.: The honeynet project: data collection tools, infrastructure, archives and analysis. In: WOMBAT Workshop on 2008 IEEE Information Security Threats Data Collection and Sharing. WISTDCS 2008, pp. 24–30 (2008)
4. Provos, N., et al.: A virtual honeypot framework. In: USENIX Security Symposium, vol. 173, pp. 1–14 (2004)
5. Krawetz, N.: Anti-honeypot technology. IEEE Secur. Priv. **2**(1), 76–79 (2004)
6. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**(7), 80–84 (2017)
7. Spitzner, L.: Honeypots: Tracking Hackers, vol. 1. Addison-Wesley, Reading (2003)
8. Zhang, F., et al.: Honeypot: a supplemented active defense system for network security. In: 2003 Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies. PDCAT-2003, pp. 231–235. IEEE (2003)
9. Spitzner, L.: Honeytokens: the other honeypot (2003). https://www.symantec.com/connect/articles/honeytokens-other-honeypots. Accessed 17 Feb 2014
10. Seifert, C., Welch, I., Komisarczuk, P.: Taxonomy of honeypots. Technical report CS-TR-06/12, School of Mathematical and Computing Sciences, Victoria University of Wellington, June 2006
11. Kuwatly, I., et al.: A dynamic honeypot design for intrusion detection. In: 2004 Proceedings of The IEEE/ACS International Conference on Pervasive Services. ICPS 2004, pp. 95–104. IEEE (2004)
12. Prasad, R., Abraham, A.: Hybrid framework for behavioral prediction of network attack using honeypot and dynamic rule creation with different context for dynamic blacklisting. In: 2010 Second International Conference on Communication Software and Networks. ICCSN 2010, pp. 471–476. IEEE (2010)

13. Jicha, A., Patton, M., Chen, H.: SCADA honeypots: an indepth analysis of Conpot. In: 2016 IEEE Conference on Intelligence and Security Informatics (ISI) 2016
14. Vormayr, G., Zseby, T., Fabini, J.: Botnet communication patterns. IEEE Commun. Surv. Tutor. **19**(4), 2768–2796 (2017)
15. Wang, P., et al.: Honeypot detection in advanced botnet attacks. Int. J. Inf. Comput. Secur. **4**(1), 30–51 (2010)
16. Holz, T., Raynal, F.: Detecting honeypots and other suspicious environments. In: 2005 Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. IAW 2005, pp. 29–36. IEEE (2005)
17. Antonakakis, M., et al.: Understanding the Mirai botnet. In: USENIX Security Symposium, pp. 1092–1110 (2017)
18. Valli, C., Rabadia, P., Woodward, A.: Patterns and patter-an investigation into SSH activity using kippo honeypots (2013)
19. Not capturing any Mirai samples. https://github.com/micheloosterhof/cowrie/issues/411. Accessed 02 Feb 2018
20. SSH Mirai-like bot. https://pastebin.com/NdUbbL8H. Accessed 28 Nov 2017
21. Khattak, S., et al.: A taxonomy of botnet behavior, detection, and defense. IEEE Commun. Surv. Tutor. **16**(2), 898–924 (2014)
22. Hayatle, O., Otrok, H., Youssef, A.: A Markov decision process model for high interaction honeypots. Inf. Secur. J.: A Global Perspect. **22**(4), 159–170 (2013)
23. Ghourabi, A., Abbes, T., Bouhoula, A.: Characterization of attacks collected from the deployment of Web service honeypot. Secur. Commun. Netw. **7**(2), 338–351 (2014)
24. Goseva-Popstojanova, K., Anastasovski, G., Pantev, R.: Using multiclass machine learning methods to classify malicious behaviors aimed at web systems. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE), pp. 81–90. IEEE (2012)
25. Wagener, G., Dulaunoy, A., Engel, T., et al.: Heliza: talking dirty to the attackers. J. Comput. Virol. **7**(3), 221–232 (2011)
26. Wagener, G., State, R., Dulaunoy, A., Engel, T.: Self adaptive high interaction honeypots driven by game theory. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 741–755. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05118-0_51
27. Pauna, A., Bica, I.: RASSH-reinforced adaptive SSH honeypot. In: 2014 10th International Conference on Communications (COMM), pp. 1–6. IEEE (2014)
28. Schaul, T., et al.: PyBrain. J. Mach. Learn. Res. **11**(Feb), 743–746 (2010)
29. Initial analysis of four million login attempts. http://www.honeynet.org/node/1328. Accessed 17 Nov 2017
30. Dowling, S., Schukat, M., Melvin, H.: A ZigBee honeypot to assess IoT cyberattack behaviour. In: 2017 28th Irish Signals and Systems Conference (ISSC), pp. 1–6. IEEE (2017)
31. Dowling, S., Schukat, M., Barrett, E.: Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware. J. Cyber Secur. Technol. 1–17 (2018) https://doi.org/10.1080/23742917.2018.1495375
32. An adaptive honeypot using reinforcement learning implementation. https://github.com/sosdow/RLHPot. Accessed 19 Dec 2017
33. Bringer, M.L., Chelmecki, C.A., Fujinoki, H.: A survey: recent advances and future trends in honeypot research. Int. J. Comput. Netw. Inf. Secur. **4**(10), 63 (2012)