Detecting Stealthy Botnets in a Resource-Constrained Environment using Reinforcement Learning

Sridhar Venkatesan, Massimiliano Albanese, Ankit Shah, Rajesh Ganesan, Sushil Jajodia
Center for Secure Information Systems
George Mason University
Fairfax, VA, USA
{svenkate,malbanes,ashah20,rganesan,jajodia}@gmu.edu

ABSTRACT

Modern botnets can persist in networked systems for extended periods of time by operating in a stealthy manner. Despite the progress made in the area of botnet prevention, detection, and mitigation, stealthy botnets continue to pose a significant risk to enterprises. Furthermore, existing enterprise-scale solutions require significant resources to operate effectively, thus they are not practical. In order to address this important problem in a resource-constrained environment, we propose a reinforcement learning based approach to optimally and dynamically deploy a limited number of defensive mechanisms, namely honeypots and network-based detectors, within the target network. The ultimate goal of the proposed approach is to reduce the lifetime of stealthy botnets by maximizing the number of bots identified and taken down through a sequential decision-making process. We provide a proof-of-concept of the proposed approach, and study its performance in a simulated environment. The results show that the proposed approach is promising in protecting against stealthy botnets.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Network security**;

KEYWORDS

Botnets; intrusion detection; reinforcement learning

1 INTRODUCTION

A botnet is a network of compromised machines controlled remotely by an attacker. Ever since its first appearance, the modus operandi of botnets has evolved to enable attackers to engage in a plethora of malicious activities. In the beginning, botnets were created and maintained for large-scale overt activities such as DDoS attacks and spam. Such botnets required the coordination of a large number of bots in order to be effective, i.e., the success of these attacks relied on the *size* of the botnet. However, over the past decade, attackers have identified new uses for botnets, such as data theft. These botnets need to *persist* in the target network for an extended period of time in order to operate effectively. To persist in the network, botnets require two properties: *stealth* and *resilience*.

Stealth is critical for a botnet's survival and is achieved by implementing any combination of *anti-signature* and *architectural stealth* [21]. Anti-signature stealth obfuscates a bot's observable behavior in order to evade detection. For instance, in order to evade network-based detection mechanisms, attackers can randomize the spatio-temporal properties of the traffic generated by bots. Several existing botnets, such as Zeus, implement anti-signature stealth by encrypting the payload while communicating with a peer or an external Command and Control (C&C) server. More advanced botnets adopt architectural stealth to construct their communication architecture such that the volume of malicious traffic intercepted by detectors is reduced. For instance, malware such as Duqu [22], Regin [10], and BlackPOS [12] – responsible for several exfiltration campaigns from corporate and government networks – constructs a communication architecture in which one of the bots acts as a proxy to aggregate data from the other bots and relays traffic to and from the external C&C server. This communication architecture was designed to limit the volume of suspicious traffic potentially intercepted by detectors at the network perimeter, thus reducing the likelihood of detection [10, 22].

In addition to being stealthy, a botnet's communication architecture must be resilient to a defender's bot takedown efforts. This property guarantees that the attacker can maintain a foothold in the target network and continuously communicate with the bots. Several advanced botnets, such as GameOver Zeus and Waledac, switch to a fallback channel (such as DGA, FastFlux) when the bots are unable to contact the C&C server using their primary communication channel (e.g., when all the peers of a bot are down). As a result, with the continuous development of new capabilities to ensure persistence, botnets have become an increasingly attractive tool for Advanced Persistent Threat (APT) actors.

Defending networks against attacks launched by stealthy botnets calls for network-wide and large-scale monitoring solutions. Due to the sophisticated nature of such attacks, no single defense mechanism is sufficient to prevent or detect them. Additionally, enterprise-scale solutions require protection mechanisms that are both proactive in preventing the propagation of botnets and reactive in detecting and responding to bots already present within the network. To address these needs, we propose to deploy – taking a defense-in-depth approach – a mix of two classes of countermeasures, namely *honeypots* and *network-based detectors*. While

This work was partially supported by the Army Research Office under grants W911NF-13-1-0421 and W911NF-13-1-0317, and by the Office of Naval Research under grant N00014-13-1-0703.

MTD'17, October 30, 2017, Dallas, TX, USA.

ACM ISBN 978-1-4503-5176-8/17/10…$15.00
https://doi.org/10.1145/3140549.3140552

honeypots are used to detect intrusion attempts [20], network-based detection mechanisms can identify – through behavioral analysis [2] – bots that coexists with benign machines. Both honeypots and network-based detectors can be treated as resources available to the defender, but a defender typically has constraints on the number of available resources. In order to optimally and dynamically deploy these defensive mechanisms, we developed a reinforcement learning (RL) model with the goal of reducing the lifetime of stealthy botnets in an resource-constrained environment.

Reinforcement learning is an algorithmic method for solving sequential decision-making problems wherein an agent (or decision-maker) interacts with the environment to learn *how* to respond under different conditions. Formally, the agent seeks to discover a policy that maps the system state to an optimal action. In the work presented here, the agent learns a policy that maximizes the total number of bots detected over time. As the location of bots is unknown prior to the deployment of defense mechanisms, the agent *estimates* the system state and the immediate reward of an action by monitoring network activity within different network segments. In particular, the agent monitors the behavior of hosts with respect to scanning and outgoing sessions within different subnets to guide the placement of defense mechanisms. To the best of our knowledge, this is the first model that employs an RL approach to optimally place detectors within a network by using information acquired through monitoring.

The rest of the paper is organized as follows. Section 2 discusses background information and related work. Sections 3 and 4 respectively describe the threat model and the defense mechanisms adopted here. Section 5 introduces the reinforcement learning model, whereas Section 6 presents simulation results and compares the performance of our approach against alternative strategies. Finally, Section 7 provides concluding remarks and indicates directions for future work.

2 BACKGROUND AND RELATED WORK

Over the past decade, several detection mechanisms – ranging from signature-based techniques, such as blacklisted C&C domains, to behavior-based techniques – have been proposed to detect botnets [2]. Among these techniques, network-based detection mechanisms [7–9, 29] have proved to be very promising. These mechanisms analyze the network behavior of all hosts within the target network and identify bots based on known invariant behaviors that are necessary for managing botnets. These invariants can be extracted from network traffic using a combination of packet-based, time-based, and behavior-based features [3]. However, despite being effective, these techniques do not scale well with traffic.

In the past, researchers have addressed the issue of scalability in Intrusion Detection System (IDS) by modeling it as a zero-sum game between the defender and the attacker [1, 11, 17, 26] in which the defender's objective is to optimally place a limited number of monitors to protect a set of target servers. The game-theoretic models in [1, 17] develop optimal placement strategies to detect intrusion attempts by considering all possible routes through which the attack can reach a target server from a given set of entry points, while the models in [11, 26] develop optimal placement strategies to minimize the attacker's control over the target server. Our reinforcement learning model differs from existing game-theoretic approaches in several ways. First, the RL model does not consider a mission-centric approach to guide placement decisions. Rather, the objective of the RL model is to maximize the number of bots detected within the network, thus driving detector placement towards securing the network as a whole. Next, although the RL model learns over a well-defined action space of the attacker, unlike existing stochastic game-theoretic models, it is oblivious to attacker's incentives in terms of the attacker's reward function for controlling machines within the network. Finally, similar to the models in [11, 26], the RL model attempts to reduce an attacker's persistence in the network, and considers a rich set of countermeasures (honeypots and botnet detection mechanisms) to detect and remove bots during different phases of the cyber kill chain.

3 THREAT MODEL

We model the lifecycle of a bot as shown in Figure 1. It begins when a benign system within the target network is compromised by either an external attacker through a client-side attack or by an existing bot within the network. To construct a resilient botnet, a new bot scans the network to discover benign systems to attack. Here, we assume that all the machines within the network are vulnerable and the corresponding exploits are available to the attacker. A bot can perform two types of scans: *worm-like* or *stealthy scan*.

In the worm-like scanning strategy, the bot sends random discovery probes to systems within its subnet, similar to the strategy employed by worms to propagate through a network [25]. These discovery probes include ICMP ping packets and incomplete TCP handshakes to determine whether a system is hosted at a given IP address and also to learn the configuration of the system, including OS version, services, etc. Due to the randomness in these scans, the bots may send discovery probes to machines that may raise red flags. For example, if a bot on a client machine sends discovery probes to another client machine, then the scanning activity may be flagged as anomalous in an enterprise network. In the stealthy scanning strategy, on the other hand, the bots first enumerate the active connections of the underlying host and then send discovery probes only to these machines. Several classes of malware employ this mechanism to move laterally through the network [23]. As one of the attacker's goals is to be stealthy, independent of the scanning strategy, we can reasonably assume the existence of an upper bound d_{max} on the number of discovery probes that a bot would send over a given period of time.

After enumerating the victim machines, the bot compromises these machines and adds them to its list of peers. As mentioned above, we assume that all machines are vulnerable and can be successfully exploited. We also assume that, in order to build a resilient botnet, each bot needs a minimum number p_{min} of peers. Upon recruiting new machines, the bot begins exchanging update messages with its peers (every ν time units). These messages inform the attacker about the status of each bot within the network and also include data stolen from the corresponding host machine. When an infected host is detected by the defender, it is restored to its original state. If the number of active peers of a bot drops below the predefined threshold p_{min}, then the bot returns to the scanning

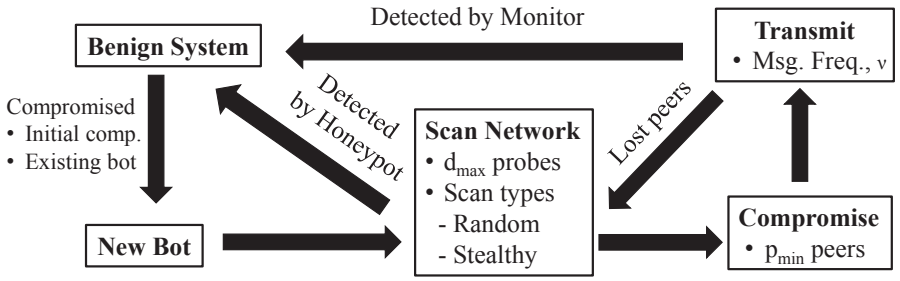


Figure 1: Lifecycle of a bot

state to recruit additional machines. Finally, to facilitate remote control by an attacker, the bots periodically check if they can reach the C&C server through their peers. If not, they establish a direct channel with the C&C server.

4 OVERVIEW OF DEFENSE MECHANISMS

In this work, we consider two classes of countermeasures: *honeypots* and *network-based detection mechanism*. An overview of these mechanisms is provided below.

Honeypots are systems that coexist with other machines within the network and are deliberately configured with vulnerabilities to lure attackers to scan and compromise them. Typically, honeypots are deployed in the DMZ to monitor attack attempts on production servers. However, many known attack campaigns establish their initial foothold by first compromising client machines through client-side attacks, social engineering, etc. Once established, an attacker moves laterally through the network by compromising additional machines and escalating privileges. To detect lateral movement of an attacker through insecure portions of the network, we need to deploy high-interaction honeypots in the internal network. High-interaction honeypots are dedicated machines that run a fully-functional operating system with vulnerable services [20]. With an increasing adoption of Software-Defined Networks (SDNs) to control the flow of traffic through a network, these honeypots can be physically located in a different region of the network while individual honeypots can be made discoverable from different network segments by installing the corresponding flow rules on the corresponding switches, using the centralized controller [18]. Although these honeypots provide a detailed insight into an attacker's intentions, they incur a high maintenance and operational cost. Therefore, we assume an upper bound H_{max} on the number of high-interaction honeypots that can be deployed.

Network-based detection mechanisms enable a defender to locate compromised machines that persist in the network and operate in a stealthy manner. Existing network-based detection techniques monitor traffic at the network gateway as they assume that all C&C servers are located outside the network. However, in the presence of a botnet that operates under architectural stealth, the volume of observable bot traffic at the network gateway is reduced [24], resulting in a low detection rate. A trivial countermeasure to capture higher volumes of bot traffic is to monitor traffic through all internal routers and switches. However, due to the poor scalability of these detection mechanisms, coupled with an ever-growing volume of internal benign traffic, detecting bots in a reasonable amount of time becomes infeasible. Thus, we only monitor a limited number M_{max} of network segments at any given time. Here, we are interested in optimally choosing the M_{max} segments to monitor and assume that M_{max} is given (a possible methodology to determine M_{max} is described in Appendix A). Furthermore, to improve network coverage, we dynamically select different segments of the network to monitor traffic. Recent work [24] has shown that continuously changing the monitored portion of the network improves the likelihood of intercepting botnet traffic. For the remainder of this paper, we consider all internal switches to be potential monitoring points.

5 REINFORCEMENT LEARNING MODEL

The defender's objective is to maximize the number of bots detected and removed using a limited number of resources (honeypots and monitors). In an enterprise, any machine that connects to the target network is susceptible to compromise and subsequent recruitment as a bot. Hence, determining the locations for placing defense mechanisms is critical to detect bots and curb their spread within the network. Furthermore, as bots can propagate through the network, the placement of these defenses must also dynamically change to detect bots in different subnetworks. Due to the evolving nature of the threat, we propose a reinforcement learning approach to guide the defender's sequential decision-making process of placing monitors and honeypots over time.

In our model, we consider an infinite horizon wherein the agent makes decisions on a periodic basis. The time between two consecutive decisions is referred to as an epoch. A timeline with the sequence of events that occur between consecutive decisions is shown in Figure 2. At each decision point, the agent determines the network segments that will be monitored during the next epoch. At the beginning of an epoch, as described in Section 3, bots perform one of two detectable activities, depending on the stage in their respective lifecycle: (i) scanning and subsequently compromising machines within the network (these bots are referred to as *scanning bots*), or (ii) exchanging update messages with their peers and the C&C server (referred to as *transmission bots*). The agent observes the network activity for a time period $\Delta t_{mon} \in [0, 1]$ — i.e., for a fraction of the epoch – during which (i) honeypots may be scanned and compromised by scanning bots, and (ii) traffic through the monitors is captured for analysis by a centralized bot detection mechanism. At time $t + \Delta t_{mon}$, the detector processes captured

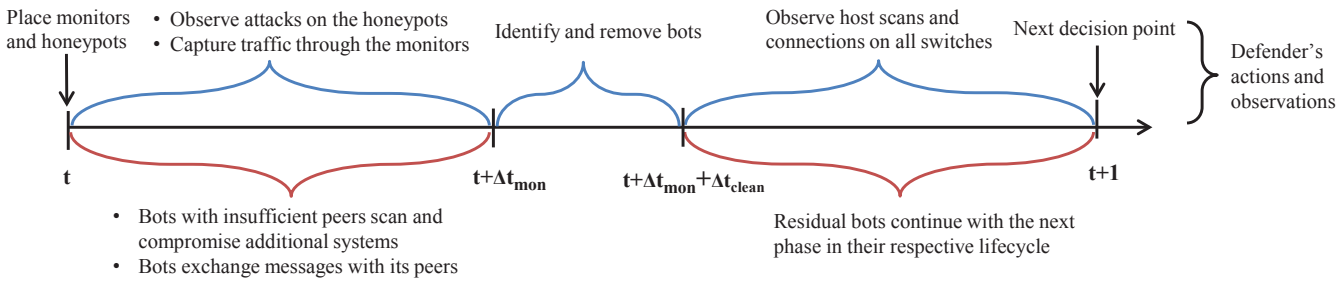


Figure 2: Timeline of defender's and attacker's actions and observations

traffic and identifies a set of potential bots. We assume that the network-based detection mechanism is imperfect, with a known true positive rate, while inference based on network activity on honeypots is assumed to be perfect[1], with a true positive rate of 1.

After identifying potential bots, the defender removes them by restoring the corresponding machines to their pristine state. Let $\Delta t_{clean} \in [0, 1]$ be the time[2] taken by the detector to process the captured traffic and subsequently remove the identified bots. In a resource-constrained setting with an imperfect detection mechanism, the defender may not have detected all the bots in the network. As a result, undetected bots continue with the next stage in their respective lifecycle. Bots with an insufficient number of peers will scan the network while bots with enough peers will exchange messages. The basic elements of the model are defined below.

Decision Variable. Given N potential monitoring points, for each point the agent may choose one of the following actions, denoted with symbols m, h, b, and e respectively: (m) passively monitor traffic traversing that monitoring point; (h) place a honeypot; (b) place both defense mechanisms; or (e) do nothing. Then, the set of decisions at time t is represented as a vector $x_t = (x_1^t, x_2^t, ..., x_N^t)$, where $x_i^t \in \{m, h, b, e\}$. Note that placing multiple monitors on the same monitoring point does not provide any additional benefit.

System State. The state of the system should capture the location of bots within the network. However, as the location of bots is unknown prior to the placement of defense mechanisms, we derive the state of the system by observing attack indicators in different segments of the network. Anomalous behaviors – such as a large number of unsuccessful login attempts, increase in the number of host scans, and a large number of outgoing sessions – are some of the most common symptoms of an ongoing attack [27]. Thus, in our model, we determine the potential locations of bots by observing anomalous behaviors in different segments of the network. In particular, to estimate the number of bots in different segments of the network, we track the total number of host scans and the total number of sessions that were recorded since the latest removal of bots from the network, i.e., in the time period $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$ in Figure 2. These features can be observed at all monitoring points – not just those where defense mechanisms have been deployed – with very low overhead.

In a network with N monitoring points, the state S_t of the system at any time t can be defined as a $2N$-dimensional vector $(\psi_1^h, \psi_1^s, \psi_2^h, \psi_2^s, ..., \psi_N^h, \psi_N^s)$, where ψ_i^h and ψ_i^s are, respectively, the host scans state and the sessions state of monitoring point i, with $i \in [1, N]$. In this work, we model the host scans state and the sessions state of each monitoring point as either LOW, MEDIUM or HIGH. In the presence of benign network activity, determining the accurate state of each feature (host scans or sessions) at different monitoring points is challenging. To address this issue, the defender must first establish a baseline behavior for each feature, for example by counting how many times a feature is observed during the time period $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$. If μ_i^f and σ_i^f are the mean and standard deviation of each feature $f \in \{h, s\}$ at monitoring point i, then the state at any given time t could be defined as:

$$\psi_i^f(t) = \begin{cases} HIGH, & \text{if } Total_i^f(t) \geq \mu_i^f + \sigma_i^f \\ MED, & \text{if } Total_i^f(t) \in (\mu_i^f - \sigma_i^f, \mu_i^f + \sigma_i^f) \\ LOW, & \text{if } Total_i^f(t) \leq \mu_i^f - \sigma_i^f \end{cases} \quad (1)$$

where, $Total_i^f(t)$ is total number of observations of feature f that were recorded during the time period $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$. In the following, when t is clear from the context, we will use ψ_i^f instead of $\psi_i^f(t)$. The intuition behind Eq. 1 is that any large deviation from the expected behavior is considered to be anomalous. It must be noted that the objective of this work is *not* to design a specific bot detector, but rather to develop a strategy for placing defense mechanisms to enable enterprise-scale botnet detection and mitigation. While fine-tuning the definition of ψ_i^f will yield more accurate results, it is beyond the scope of this work.

Reward Function. In an RL model, the choice of an optimal action is influenced by the immediate reward $R(S_t, x_t)$ of an action. Here, the reward of an action is defined as the number of bots that are correctly identified. However, taking an action x_t at time t, when the system is in state S_t, yields a reward that is measured at a later time, $t + \Delta t_{mon} + \Delta t_{clean}$. This is a class of time-lagged information acquisition problems, where we do not know the value of the current state until it is updated after the uncertainty in the bot activity is revealed. Therefore, the immediate reward of an action is *estimated* by using information from recent observations. Such problems occur in real world, such as when travel and hotel reservation decisions are done today for a future date and the value of making such decisions is unknown until the date has occurred [5, 14, 15].

We estimate the number of bots in a network segment by determining the number of hosts that have deviated from the expected

[1]Attempted access to a honeypot can be assumed an indicator of malicious activity.
[2]Both Δt_{mon} and Δt_{clean} are defined as a fraction of an epoch.

behavior. Similar to the motivation behind deriving the state of the system, the defender first establishes a baseline for the network activity of each machine across all monitoring points. Let $\mu^f_{mc,i}$ and $\sigma^f_{mc,i}$ be the mean and standard deviation of feature f for machine mc when observed from monitoring point i. We consider a simple threshold scheme to decide whether a machine is a potential bot. Given any machine mc and a monitoring point i, if $Total^f_{mc,i}(t)$ is the total number of observations of feature f that were recorded during the time period $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$, then machine mc is considered a potential bot if and only if $(\exists i \in [1, N])(Total^f_{mc,i}(t) \geq \mu^f_{mc,i} + 3 \cdot \sigma^f_{mc,i})$. It should be noted that this rule to identify suspicious machines can be modified based on the specific settings of the target network and does not limit the generality of the proposed RL model.

Post-decision System State. The post-decision system state, S^x_t, is the state to which the system transitions after the decision x_t is taken. Similar to the reward function, the change in the state of the system can only be observed at a later time, in this case $t + 1$. Therefore, we *estimate* the post-decision state of the system by determining the *expected* effect of a decision.

Our estimation is based on the rationale that the objective of placing a defense mechanism at a monitoring point is to remove bots from that portion of the network by identifying machines that exhibit anomalous behaviors. In particular, suppose that machines $mc_1, mc_2, ..., mc_k$ are identified as potential bots from the monitoring point i due to deviations w.r.t. a feature f. Then, placing a defense mechanism at time t (a honeypot if f is the host scans count, or a network-based detector if f is the sessions count) is expected to confirm, after a monitoring period Δt_{mon}, whether the suspected bots are actually bots and, if so, restore the corresponding machines $mc_j, j \in [1, k]$ to their pristine state. As a result of the cleaning process, the agent expects to record $\hat{\mu}^f_{mc_j,i}, \forall j \in [1, k]$ observations of feature f at the monitoring point i during the time $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$. Assuming that the machines that are not expected to be affected by this decision continue with their latest recorded behavior (i.e., the behavior exhibited during $[t-1+\Delta t_{mon}+\Delta t_{clean}, t)$), then the new post-decision state of monitoring point i for a feature f can be obtained by using Eq. 1, where the estimated total number of observations of feature f is given by $\widehat{Total}^f_i(t+1) = \sum_{j\in[1,k]} \hat{\mu}^f_{mc_j,i} + \sum_{j\notin[1,k]} \mu^f_{mc_j,i}$, with $\hat{\mu}^f_{mc_j,i}$ being the estimated behavior of machine mc_j after the placement of a defense mechanism. It must be noted that, since the baseline values $(\mu^f_{mc_j,i}, \sigma^f_{mc_j,i})$ of all machines at different monitoring points are established as a preprocessing step, the post-decision state reached by a system due to an action can be obtained immediately.

Exogenous Information. The exogenous information, or uncertainty, B_{t+1} is the information from the environment that is acquired after decision x_t. The uncertainty is attributed to the coexistence of benign and malicious behavior within the network, making it challenging to model the evolution of bots. In the RL model, the uncertainty is captured by observing network activity and extracting features from different monitoring points.

State transition function. The state transition function, defined as $S_{t+1} = \tau(S_t, x_t, B_{t+1})$, captures how the system state evolves. However, due to the absence of a model to predict B_{t+1}, the state transition probabilities are unknown. Hence, a reinforcement learning based approach is used to study the evolution of the system state.

Objective Function. The objective function is defined as the long-run total discounted value of the states $V^j(S)$ as the iteration index $j \to \infty$, which is derived using the recursive Bellman's optimality equation [4] below (Eq. 2). Here, $V^j(S)$ is the cumulative sum of discounted $R(S_t, x_t)$ rewards for the learning phase, whose iterations are indexed from 1 to j. In this work, we consider a 365-day cycle in which decisions are made at the start of each day. The learning phase goes through several iterations (indexed with j) of 365-day cycles. As the value of a state is measured in terms of number of correctly identified bots, the objective function will be to maximize the long-run total discounted value of the states $V^j(S)$: the higher the value of $V^j(S)$, the better the system state. The model strives to transition from one good state to another by making a decision that is guided by the highest value of the estimated future states that are reachable at any given time t.

5.1 Phases of Reinforcement Learning

RL achieves the objective through three phases, namely, exploration, learning, and learned. The recursive Bellman's optimality equation that updates the value of the states is given as follows:

$$V^j(\hat{S}^x_{t-1}) = (1 - \alpha^j)V^j(\hat{S}^x_{t-1}) + \alpha^j v^j \tag{2}$$

$$v^j = \left[\max_{x_t \in \mathcal{X}} \left\{R(S_t, x_t) + \beta V^j(\hat{S}^x_t)\right\}\right] \tag{3}$$

where $V^j(S)$ denotes the value of state S at the j-th iteration, $\hat{S}^x_t$ is the estimated post-decision state reached by the system at state S_t under the action x_t, α^j is the learning parameter that is decayed gradually, $\mathcal{X}$ is the set of all feasible decisions from which the model will choose a decision at every iteration, and β is the fixed discount factor that allows the state values to converge in a long run. It should be noted that the value of the estimated post-decision state $\hat{S}^x_{t-1}$ is updated at time t (in Eq. 3) using the estimated reward function and the value of the estimated post-decision states that can be reached under different actions. In a classical RL formulation, the immediate real rewards and the immediate value of the post-decision states at time t are known; hence, the value of the post-decision state at time $t - 1$ can be updated using Eq. 2 and Eq. 3. However, as both the rewards and post-decision states are estimated, we update the value of the post-decision state only after the real reward for an action is observed.

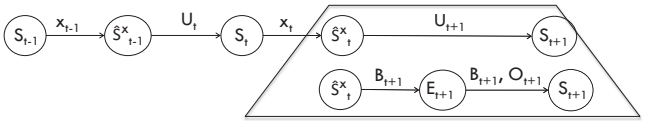


Figure 3: State transition diagram

A snapshot of the state-transition diagram is shown in Figure 3, in which U_t denotes the uncertainty (before and after removing bots) after taking an action, and O_{t+1} denotes the features observed at different monitoring points after removing the bots. The bot

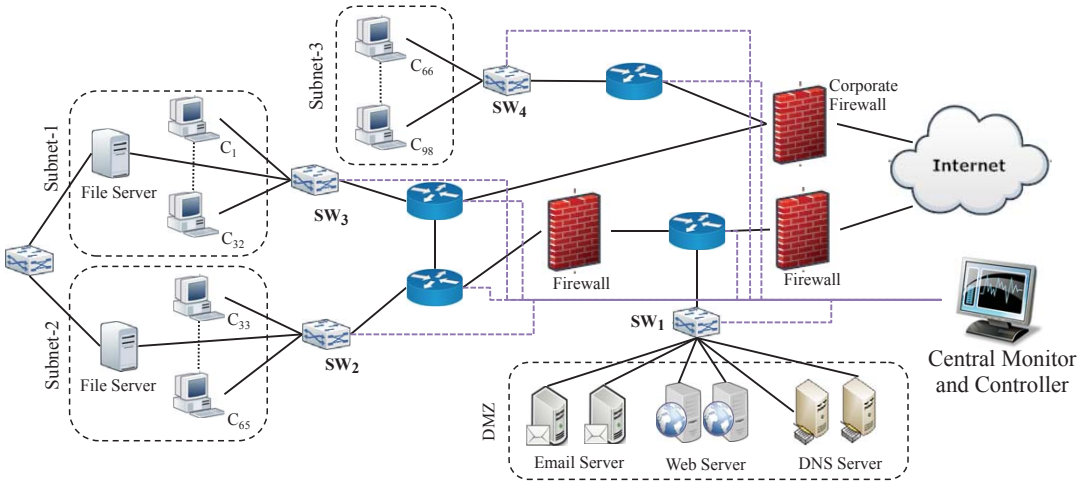


Figure 4: Enterprise network

Algorithm 1 Exploration and Learning Phase

Input: Baseline values ψ_i^f of each feature $f \in \{h, s\}$ for each monitoring point i, baseline values $\psi_{mc,i}^f$ for each machine mc, decision space $\mathcal{X}$, initial learning parameter $\alpha^0 = 0.8$ at time $t = 0$, discount parameter $\beta = 0.95$, number of iterations for learning $J = 1000$.
Output: State value function, $V(S)$, $\forall S$

1: $V(S) \leftarrow 0, \forall S$
2: **for all** $j = \{1, \ldots, J\}$ **do**
3: **if** $j \leq 0.3 \cdot J$ **then**
4: $Phase \leftarrow$ Exploration
5: **else**
6: $Phase \leftarrow$ Learning
7: **end if**
8: **for all** $t = \{1, \ldots, 365\}$ **do**
9: Observe features from the monitoring points and determine state S_t
10: **if** $Phase$ = Exploration **then**
11: Choose a random defense placement decision, x_t
12: **else**
13: Estimate immediate reward $R(S_t, x_t)$ and post-decision state $\hat{S}_t^x$, as described in Section 5, $\forall x_t \in \mathcal{X}$
14: Choose the action x_t' that gives the maximum value in Eq. 3
15: **end if**
16: **if** $t > 2$ **then**
17: Observe the real reward at $t + \Delta t_{mon} + \Delta t_{clean}$
18: Decay the learning parameter, $\alpha^j = \frac{\alpha^j}{1+e}$, where $e = \frac{j^2}{1.25 \cdot 10^{14}+j}$ // see [6]
19: Update value of post-decision state $V^j(\hat{S}_{t-1}^x)$ using Eq. 2 and Eq. 3 with the real reward and the value of α^j
20: **end if**
21: **end for**
22: **end for**

removal stage is denoted by E_{t+1}. In this model, during the learning phase, the choice of an action x_t when the system is in state S_t is determined by the estimated reward function $R(S_t, x_t)$. After taking the action x_t, the uncertainty U_{t+1} unfolds, transitioning the system to the state S_{t+1}. As the uncertainty U_{t+1} unfolds (shown in trapezoidal box in Figure 3), bots are removed at stage E_{t+1} and the agent observes the real reward which is then used to update the value of the estimated post-decision state $\hat{S}_{t-1}^x$. The three phases of learning are described below.

Exploration Phase. In this phase, the RL algorithm explores several non-optimal decisions and acquires the value of the system states that are visited. As described in Algorithm 1, Eq. 2 is used without the max operator in Eq. 3 by taking random decisions for placing defense mechanisms, and the values of $V^j(S_t^x)$ and $V^j(\hat{S}_{t-1}^x)$ are taken from the previously stored values, if the state was visited, or set to 0 otherwise. Since the algorithm begins with $V^0(S) = 0, \forall S$ at $j = 0$, exploration helps to populate the values of some of the states that are visited. Exploration is stopped after a certain number of iterations, which depends on the size of the state-space and the number of iterations planned for the learning phase. In our simulation, we stopped exploration after 30% of the total number of iterations. The idea here is to explore as many states as possible during the learning phase. A low value of this parameter would imply not enough states being explored, whereas a high value would lead to non-convergence of state values during the learning phase. Thus, our choice is reasonable and quite common for this class of problems.

Learning Phase. In this phase, the algorithm takes near-optimal decisions at time t, which are obtained from Eq. 3 with the max operator (lines 13-14 of Algorithm 1). The value of the post-decision state at time $t-1$ is updated at time $t+1$ as per Eq. 2 with the real rewards. After several iterations, learning is stopped when convergence of the value of the states is achieved, as measured in terms of the mean-square error of the stochastic gradient [15].

Learned Phase. This is the implementation phase of the RL. The inputs to this phase include the value of the states at the time when learning was terminated and the estimated reward function. In this phase, the RL algorithm takes optimal decisions at each time t, which is obtained from Eq. 3 with the max operator. The algorithm then evaluates all its feasible actions and chooses an action that takes the system to the post-decision state with the highest value.

6 SIMULATION RESULTS

In this paper, we consider the enterprise network shown in Figure 4 to study the performance of the proposed RL model. The network is composed of 106 machines, with 98 client machines and 8 servers that are distributed across 4 subnets. In this network, the four switches, SW_1, SW_2, SW_3, and SW_4, act as the monitoring points for

Table 1: Benign traffic - Request rate distribution

Traffic	Request Rate	Additional details
HTTP	$7.39e5 + 8.58e4 \cdot Beta(10, 8.26)$ (per day)	• 80% of requests to the Internet. 20% of requests to the internal Web servers. • Requests are evenly distributed across the internal Web servers.
FTP	Uniform random intervals of 1 - 5 hours	• Requests from machines are served by the respective local FTP servers. • FTP requests from Subnet-3 are served by the server in Subnet-1.
SMTP	Uniform random intervals of 1 - 30 mins	• Requests are evenly distributed across the internal Email servers.
DNS	Generate 5 requests to resolve a HTTP request	• Requests are evenly distributed across the internal DNS servers.

deploying network-based detectors and honeypots. These switches, as depicted in the figure, are remotely controlled by an administrator in order to: (i) activate/deactivate mirroring ports that mirror traffic to the central monitor for analysis; and (ii) insert flow rules on the switches to make honeypots discoverable from different subnets in an SDN-based network. It should be noted that the physical location of the honeypots does not influence an agent's decision about placement of defense mechanisms and, hence, their location is ignored in Figure 4.

Defender's Parameters. For the case study under consideration, we set $H_{max} = 2$ and $M_{max} = 2$, i.e., a maximum of two honeypots and two monitors can be deployed in the network. As there are only 4 potential monitoring points, we need to set the value of M_{max} to less than 4. This choice ensures that no more than 50% of the network is monitored at any given time. On the other end, the minimum possible value for H_{max} is 1, thus, by conservatively setting it to 2 in a network with over 100 nodes, the honeypots represent less than 2% of all the machines in the network. Even with such a small percentage of hosts acting as honeypots, our RL approach shows a very good performance.

Existing botnet detection mechanisms [7, 28] show a true positive rate of at least 90% when evaluated against known malware. However, if an attacker introduces perturbations to the malware's behavior, the performance of the detection mechanism may degrade. Hence, to account for such degradation, in this study we conservatively estimate the true positive rate at 70%[3].

In this work, we consider a 365-day scheduling horizon in which the agent makes decisions at the start of each 1-day epoch. During each epoch, for the first 12 hours, i.e., $\Delta t_{mon} = 0.5$, the defender observes the network activity on the deployed honeypots and mirrors traffic from the monitors to the central location. For the purposes of this simulation-based study, we set $\Delta t_{clean} = 0$, i.e., the time taken for the detection mechanism to analyze the captured traffic and restore the identified machines to their pristine state is considered to be negligible. As we do not consider false positives and the cost to clean a machine, there is no benefit in setting $\Delta t_{clean} > 0$.

Benign Traffic Generation. The network activity of benign machines plays a crucial role in setting the baseline behavior of the observable features at different monitoring points and consequently, in the defense placement decisions taken by an agent. In our simulation, the traffic characteristics were obtained from [19], where the authors developed traffic profiles for different types of protocols based on the traffic captured in a testbed environment. In the considered network, each machine generates four types of traffic: HTTP, FTP, SMTP, and DNS, with a traffic composition similar to that of [19]. Since the agent's decisions are influenced by the total number of recorded observations for each feature, the simulation only requires the distribution of the rate at which machines generate requests. A summary of the request rate distributions for different types of traffic is provided in Table 1. In addition to complete sessions (for HTTP, FTP and SMTP), we considered a 1% packet drop rate to simulate scanning-like behavior for benign machines. The process of defining the baseline behavior is outside the scope of this paper, but the details for this simulation-based study are provided in Appendix B.

Botnet Parameters. In addition to the benign traffic, compromised machines generate traffic compatible with the bot lifecycle described in Section 3. In our simulation, the parameters necessary to generate a bot's network activity were determined either based on known behavior obtained via malware analysis reports or through a conservative estimate that would enable a stealthy operation. At the start of the simulation, machines within the network are assumed to be compromised by client-side attacks with a 3% probability. After establishing the initial foothold, the bots scan the network and enlist $d_{max} = 3$ machines for subsequent compromise. As described in Section 3, the bots can perform two types of scans, random or stealthy, which they choose with probability 0.8 and 0.2 respectively. Here, we consider a stealthy botnet in which each bot compromises only $p_{min} = 1$ additional machine. Upon compromising and recruiting additional bots, the bots begin exchanging messages with their peers every $\nu = 15$ minutes. As shown in [16], 7 out of 11 periodic P2P botnets (ignoring Nugache) have periodicities of less than 15 minutes. Thus, our choice for the value of ν is realistic and models a stealthier botnet, compared to several existing P2P botnets. Finally, in this simulation, we considered a worst-case scenario where a botnet is always present in the network, i.e., if all the bots are removed by the defense mechanisms, then new bots are created through a client-side attack in the subsequent epoch. The spike in network activity due to bot traffic is illustrated in Appendix B.

6.1 Comparison with Alternative Strategies and Metrics

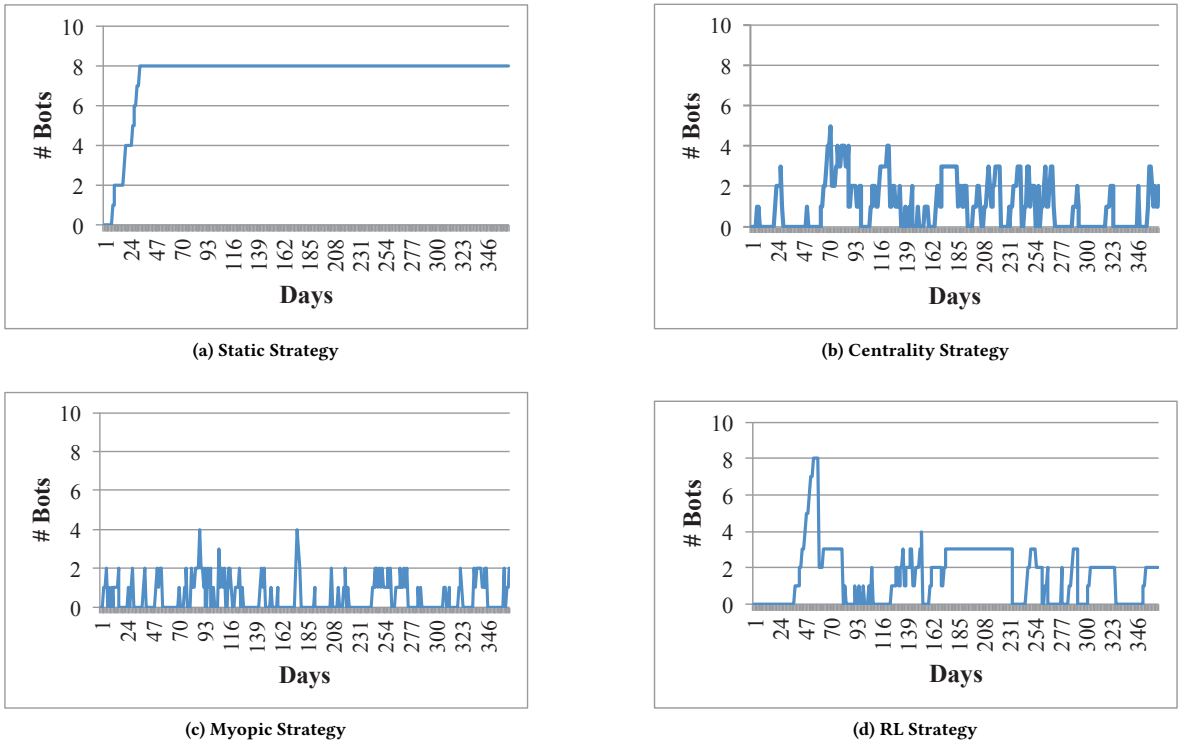
In order to study the effectiveness of the RL model to detect and remove stealthy botnets, we considered the following alternative placement strategies.

Static Strategy. In this strategy, the defender's main objective is to detect botnet activity in the network segments that host sensitive data using static defenses. In the network of Figure 4, the file servers in Subnet-1 and Subnet-2 are assumed to host sensitive data and,

[3]Here, we do not consider the false positive rate of a detection mechanism as the model does not penalize misclassifications. We plan to incorporate the cost of misclassifications as part of our future work.

Table 2: Performance comparison of different strategies

Strategy	Avg. Bot Liveprint	Avg. Bot Meanprint	Avg. Max. Bot Lifetime (days)	Avg. BPT (days)
Static	14.06	12.30	360.3	362.1
Centrality	11.28	2.89	48.94	74.86
Myopic	11.44	3.34	26.84	52.30
RL	12.90	2.97	20.0	37.40



(a) Static Strategy

(b) Centrality Strategy

(c) Myopic Strategy

(d) RL Strategy

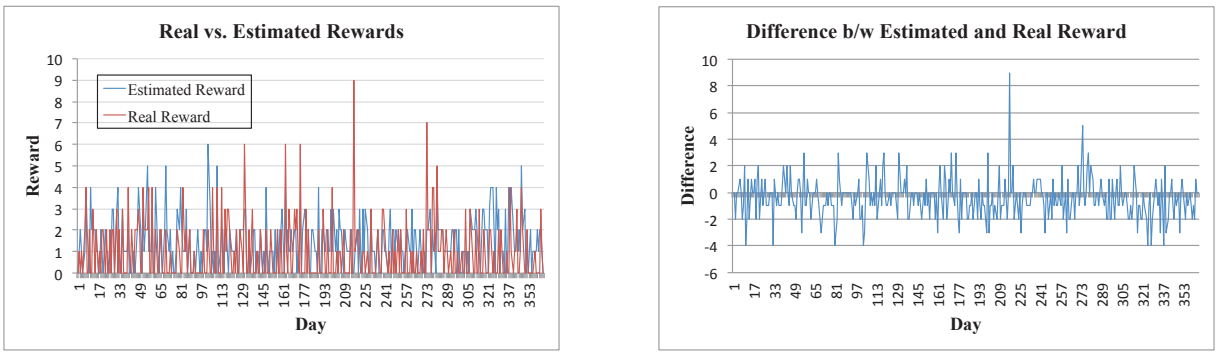
Figure 5: Total number of bots in the DMZ for one 365-day run

hence, the defender places both the monitors and the honeypots on switches SW_2 and SW_3.

Centrality-weighted Strategy. This strategy – proposed in [24] – is a proactive approach to dynamically place defense mechanisms in a network. Assuming that the file servers in Subnet-1 and Subnet-2 host sensitive data, this strategy models the network as a graph and computes a centrality metric referred to as *mission-betweenness centrality* for each monitoring point in the network. A monitoring point with a high centrality value is expected to intercept a larger volume of bot traffic and, hence, improve the likelihood of detecting botnets that operate under architectural stealth. After computing the centrality, the strategy chooses monitoring points randomly weighted by their centrality values: higher the centrality value, higher the probability of being chosen. Although the original work was designed to deploy only network-based detectors, in this work the placement of honeypots is also guided by the same approach.

Myopic Strategy: This strategy is motivated by the existing practice to analyze traffic in a network segment *only* when the network operator observes a significant spike in the network activity. In particular, the defender places a honeypot (respectively a network-based detector) on a network segment when the host scan state (respectively the sessions state) reaches the HIGH value. Unlike the earlier strategies, the defense placement decisions in this strategy are driven by the information acquired from the operating environment. However, dissimilar to the RL model, this strategy does not account for the long-term values of the state that the system may transition to as a result of the decision.

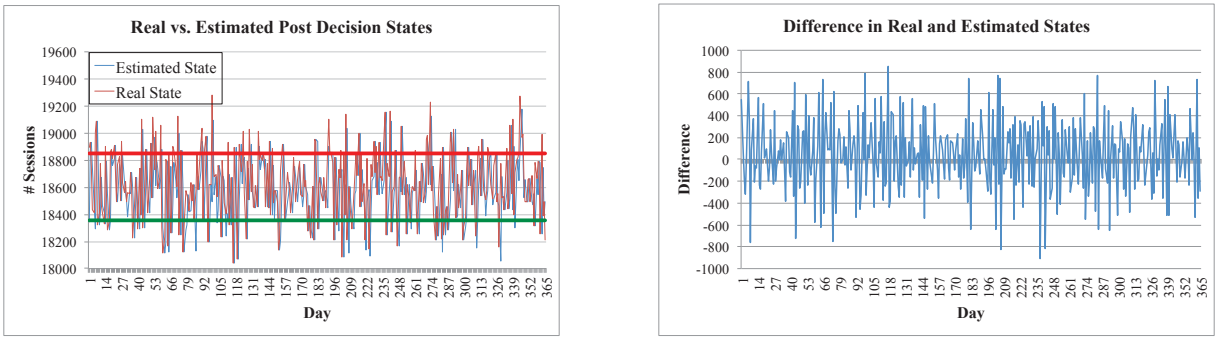
The following metrics were used to assess and compare the above strategies: (i) ***Bot Liveprint***: maximum number of bots present during the simulation; (ii) ***Bot Meanprint***: average number of bots present during the simulation; (iii) ***Maximum Bot Lifetime***: longest period of time during which any machine was controlled by an attacker; (iv) ***Botnet Persistence Time (BPT)***: maximum lifetime of a botnet, where a botnet's lifetime is defined as the time between the appearance of the first bot and removal of all bots.



(a) Real vs. Estimated Rewards

(b) Difference between Estimated and Real Rewards

Figure 6: Real and Estimated Rewards at switch SW_2



(a) Real vs. Estimated Post-Decision States

(b) Difference between Estimated and Real Post-Decision States

Figure 7: Real and Estimated Post-decision States at switch SW_2

6.2 Results

In this work, the network was simulated using PeerSim [13], a simulator designed to support scalable simulation of P2P protocols. All the strategies were evaluated during a 365-day cycle with placement decisions made at the beginning of each one-day epoch. For each strategy, the simulation was repeated for 50 runs in which each run was initialized with a different seed that controlled the operations of a bot. The performance of each strategy was averaged across all the runs and the average values were used for comparing strategies at the 95% confidence interval.

A summary of the results is provided in Table 2. As expected, the static placement strategy exhibits the worst performance. While the static strategy ensured that bots in Subnet-1 and Subnet-2 were removed, it could not detect bots in Subnet-3. As a result, once a botnet was established in Subnet-3, the bots continued to persist in the network till the end of the simulation (see Figure 5). The dynamic strategies, on the other hand, provide a significantly better protection against such stealthy and persistent botnets. In particular, the centrality-weighted and the myopic strategies reduce the maximum lifetime of a bot by at least 85% and, consequently, reduce the lifetime of a botnet (BPT) by more than 80%. The performance of these strategies strongly suggests that introducing *dynamism* while making defense placement decisions can significantly improve the security posture of the network.

We did not observe a statistically significant difference between the centrality-weighted and the myopic strategies with respect to bot liveprint and meanprint. However, the myopic strategy showed a significant improvement over the centrality-weighted strategy in reducing the lifetime of a bot as well as the lifetime of the botnet. This was because the decisions taken by the centrality-weighted strategy were largely concentrated around Subnet-1 and Subnet-2 (since, the corresponding switches had high centrality values) and hence, the bots in Subnet-3 and DMZ persisted for a longer period of time. The myopic strategy, on the other hand, took decisions based on the information obtained from the network; the strategy immediately reacted to spikes in the network activity and placed defense mechanisms on those subnets. Thus, when bots propagated and operated in Subnet-3 (or DMZ), the myopic strategy observed a spike in the network activity and removed the corresponding bots quicker than the centrality-weighted strategy (as shown in Figure 5). This improvement in performance strongly suggests that decisions that are guided by information acquired from the environment lead to effective defense placements.

Finally, amongst the dynamic strategies, the RL model showed the largest reduction of a bot's lifetime as well as of the botnet's

lifetime (BPT). Similar to the performance of the myopic strategy, the RL model's performance stems from the fact that its decisions are based on information acquired from the operating environment. However, the RL model synthesizes the acquired information to estimate the immediate reward (in terms of number of bots) and the post-decision state – whose long-term value is obtained through learning – to make decisions. A comparison of the estimated and real rewards and the post-decision states for one 365-day run is shown in Figure 6 and Figure 7 respectively. In the model, the post-decision states capture the impact of a decision on the bots that survive a defense placement decision. As a result, the RL model is able to *control* the evolution of a botnet within the network and thus performs better than its alternatives. Additionally, we performed sensitivity analysis on the parameters discussed above, and observed results consistent with the findings presented here.

7 CONCLUSIONS AND FUTURE WORK

As several noteworthy incidents have shown, modern botnets can operate in stealthy mode and persist in networked systems for extended periods of time. Despite the significant progress made in the area of botnet prevention, detection, and mitigation, stealthy botnets continue to pose significant risks for enterprises. Additionally, existing enterprise-scale solutions are inefficient. To address this important problem in a resource-constrained environment, we proposed a reinforcement learning based approach to optimally and dynamically deploy a limited number of defensive mechanisms, namely honeypots and network-based detectors, within the target network. The ultimate goal of the proposed approach is to reduce the lifetime of stealthy botnets by maximizing the number of bots identified through a sequential decision-making process. We provided a proof-of-concept of the proposed approach, and studied its performance in a simulated environment. The results show that our approach is promising in protecting against stealthy botnets. Planned future work includes but is not limited to (i) conducting more extensive experiments; (ii) considering the cost of restoring machines to their pristine state once bots have been located; and (iii) considering the implications of a detector's misclassifications.

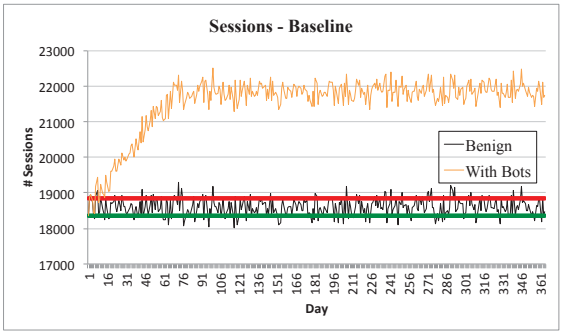
APPENDICES

A NUMBER OF MONITORING POINTS

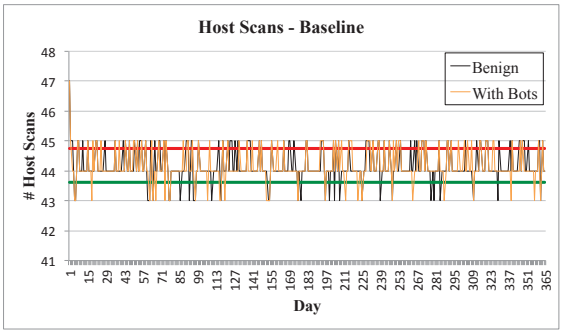
For the purpose of modeling, given the computation resources available for detection, let Q denote an upper bound on the number of packets that can be processed by the centralized detection algorithm in an amount of time $\Delta t_{clean} \ll \Delta t_{mon}$. Assuming that the volume of traffic across different segments of the network is uniform, say P packets per time unit, then the maximum number of segments that can be monitored concurrently can be approximated as $M_{max} = \left\lfloor \frac{Q}{P \cdot \Delta t_{mon}} \right\rfloor$, where Δt_{mon} is the length of the monitoring periodd, as shown in Figure 2. Here, if there are N potential monitoring locations, then all M_{max}-sized combinations of locations are admissible solutions. It should be noted that in reality the traffic volume across different portions of the network may not be uniform; however, the average traffic volume at different locations may be known through historical data. In such situations, the problem of finding feasible solutions for monitor placement is NP-Hard as it can be shown that the 0/1 Knapsack problem can be reduced to this problem in polynomial time. Although the problem is NP-Hard for large networks, the set of feasible solutions can be obtained in polynomial time for small-medium scale networks that have fewer monitoring locations.

B BASELINE BEHAVIOR

In order to model the baseline behavior of benign machines and bots, we run a 365-day simulation in which each machine generated traffic in accordance to the request rates in Table 1. During this simulation, we learned the feature statistics, μ_i^f and σ_i^f, of each feature $f \in \{h, s\}$ for each monitoring point i during each epoch, and $\mu_{mc,i}^f$ and $\sigma_{mc,i}^f$ for each feature f for each machine mc as observed from the monitoring point i. Figure 8 illustrates the baseline behavior for the sessions and host scans state of switch SW_2. In the figure, the red and green lines denotes thresholds for the HIGH, MED and LOW states. Figure 8 also illustrates the spike in network activity (w.r.t. the number of sessions and host scans) in the presence of such a botnet, when no defense mechanisms were installed in the network. It must be noted that, as there were no defense mechanisms, after the initial scan (during the first few epochs) and subsequent compromise, the bots do not return to their scanning state. Hence, no anomalous scanning activity is observed in subsequent epochs.



(a) Sessions



(b) Host Scans

Figure 8: Total number of sessions and scans observed from switch SW_2

REFERENCES

[1] Tansu Alpcan and Tamer Başar. 2006. An Intrusion Detection Game with Limited Observations. In *Proceedings of the 12th International Symposium on Dynamic Games and Applications*. Sophia-Antipolis, France.

[2] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. 2009. A Survey of Botnet Technology and Defenses. In *Proceedings of the Cybersecurity Applications & Technology Conference for Homeland Security (CATCH 2009)*. 299–304.

[3] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A. Ghorbani. 2014. Towards Effective Feature Selection in Machine Learning-Based Botnet Detection Approaches. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS 2014)*. IEEE, San Francisco, CA, USA, 247–255.

[4] Richard Bellman. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

[5] Rajesh Ganesan, Sushil Jajodia, Ankit Shah, and Hasan Cam. 2016. Dynamic Scheduling of Cybersecurity Analysts for Minimizing Risk Using Reinforcement Learning. *ACM Transactions on Intelligent Systems and Technology* 8, 1 (2016).

[6] Abhijit Gosavi. 2003. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Springer.

[7] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium*. San Jose, CA, USA, 139–154.

[8] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. 2007. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of the 16th USENIX Security Symposium*. Boston, MA, USA, 167–182.

[9] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*. San Diego, CA, USA.

[10] Kaspersky Lab. 2014. *The Regin Platform Nation-State Ownage of GSM Networks*. Technical Report. Kaspersky Lab.

[11] Karim Khalil, Zhiyun Qian, Paul Yu, Srikanth Krishnamurthy, and Ananthram Swam. 2016. Optimal Monitor Placement for Detection of Persistent Threats. In *Proceedings of the IEEE Global Communications Conference (IEEE GLOBECOM 2016)*. IEEE, Washington, DC USA.

[12] Marion Marschalek, Paul Kimayong, and Fengmin Gong. 2014. *POS Malware Revisited: Look What We Found Inside Your Cashdesk*. Technical Report. Cyphort Labs.

[13] Alberto Montresor and Márk Jelasity. 2009. PeerSim: A Scalable P2P Simulator. In *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009)*. Seattle, WA, USA, 99–100.

[14] Juliana M. Nascimento and Warren B. Powell. 2009. An Optimal Approximate Dynamic Programming Algorithm for the Lagged Asset Acquisition Problem. *Mathematics of Operations Research* 34, 1 (February 2009), 210–237.

[15] Warren B. Powell. 2011. *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (2nd ed.). John Wiley & Sons.

[16] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J. Dietrich, and Herbert Bos. 2013. SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (S&P 2013)*. IEEE, San Francisco, CA, USA, 97–111.

[17] Stephan Schmidt, Tansu Alpcan, Şahin Albayrak, Tamer Başar, and Achim Mueller. 2007. A Malware Detector Placement Game for Intrusion Detection. In *Proceedings of the 2nd International Workshop on Critical Information Infrastructures Security (CRITIS 2007)*. Springer, Benalmádena, Málaga, Spain, 311–326.

[18] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. 2016. Enhancing Network Security through Software Defined Networking (SDN). In *Proceedings of the 25th International Conference on Computer Communication and Networks (ICCCN 2016)*. IEEE, Waikoloa, HI, USA.

[19] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security* 31, 3 (May 2012), 357–37.

[20] Lance Spitzner. 2002. *Honeypots: Tracking Hackers*. Addison Wesley, Boston, MA, USA.

[21] Patrick John Sweeney. 2014. *Designing Effective And Stealthy Botnets for Cybet Espionage And Interdiction - Finding the Cyber High Ground*. Ph.D. Dissertation. Thayer School of Engineering, Darthmouth College.

[22] Symantec Security Response. 2011. W32.Duqu: The Precursor to the Next Stuxnet. https://www.symantec.com/connect/w32_duqu_precursor_next_stuxnet. (October 2011).

[23] Trend Micro. 2013. Lateral Movement: How Do Threat Actors Move Deeper Into Your Network? (2013).

[24] Sridhar Venkatesan, Massimiliano Albanese, George Cybenko, and Sushil Jajodia. 2016. A Moving Target Defense Approach to Disrupting Stealthy Botnets. In *Proceedings of the 3rd ACM Workshop on Moving Target Defense (MTD 2016)*. ACM, Vienna, Austria, 37–46.

[25] Yini Wang, Sheng Wen, Yang Xiang, and Wanlei Zhou. 2014. Modeling the Propagation of Worms in Networks: A Survey. *IEEE Communications Surveys & Tutorials* 16, 2 (2014), 942–960.

[26] Michael P. Wellman and Achintya Prakash. 2014. Empirical Game-Theoretic Analysis of an Adaptive Cyber-Defense Scenario (Preliminary Report). In *Proceedings of the International Conference on Decision and Game Theory for Security (GameSec 2014) (Lecture Notes in Computer Science)*, Vol. 8840. Springer, Los Angeles, CA, USA, 43–58.

[27] Michael West. 2009. *Computer and Information Security Handbook*. Morgan Kaufmann, Chapter Preventing System Intrusions, 39–51.

[28] Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz. 2014. Building a Scalable System for Stealthy P2P-Botnet Detection. *IEEE Transactions on Information Forensics and Security* 9, 1 (January 2014), 27–38.

[29] Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. 2011. Detecting stealthy P2P botnets using statistical traffic fingerprints. In *Proceedings of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011)*. IEEE, Hong Kong, China, 121–132.