Adaptive Cyber Defenses for Botnet Detection and Mitigation

Massimiliano Albanese[1](✉), Sushil Jajodia[1], Sridhar Venkatesan[2], George Cybenko[3], and Thanh Nguyen[4]

[1] George Mason University, Fairfax, VA, USA
{malbanes,jajodia}@gmu.edu
[2] Perspecta Labs, Basking Ridge, NJ, USA
svenkatesan@perspectalabs.com
[3] Dartmouth College, Hanover, NH, USA
george.cybenko@dartmouth.edu
[4] University of Oregon, Eugene, OR, USA
tnguye11@uoregon.edu

Abstract. Organizations increasingly rely on complex networked systems to maintain operational efficiency. While the widespread adoption of network-based IT solutions brings significant benefits to both commercial and government organizations, it also exposes them to an array of novel threats. Specifically, malicious actors can use networks of compromised and remotely controlled hosts, known as botnets, to execute a number of different cyber-attacks and engage in criminal or otherwise unauthorized activities. Most notably, botnets can be used to exfiltrate highly sensitive data from target networks, including military intelligence from government agencies and proprietary data from enterprise networks. What makes the problem even more complex is the recent trend towards stealthier and more resilient botnet architectures, which depart from traditional centralized architectures and enable botnets to evade detection and persist in a system for extended periods of time. A promising approach to botnet detection and mitigation relies on Adaptive Cyber Defense (ACD), a novel and game-changing approach to cyber defense. We show that detecting and mitigating stealthy botnets is a multi-faceted problem that requires addressing multiple related research challenges, and show how an ACD approach can help us address these challenges effectively.

1 Introduction

Organizations increasingly rely on complex networked systems to maintain operational efficiency. While the widespread adoption of network-based IT solutions brings significant benefits to both commercial and government organizations, it also exposes them to an array of novel threats. For instance, advanced

The work presented in this chapter was support by the Army Research Office under grant W911NF-13-1-0421.

persistent threats (APTs) and distributed denial-of-service (DDoS) attacks can bypass traditional defenses by leveraging an arsenal of diverse and sophisticated cyber tools. Specifically, malicious actors can use networks of compromised and remotely controlled hosts, known as botnets, to execute a number of different cyber attacks and engage in criminal or otherwise unauthorized activities. Most notably, botnets can be used to exfiltrate highly sensitive data from target networks, including military intelligence from government agencies and proprietary data from enterprise networks. In a society that has significantly shifted from producer of goods to producer of information-centric services, protecting sensitive and mission-critical data from competitors, state actors, and organized crime has become increasingly critical for the well-being of many commercial and government organizations.

What makes the problem even more complex is the recent trend toward stealthier and more resilient botnet architectures, which depart from traditional centralized architectures and enable botnets to evade detection and persist in a system for extended periods of time. Botnets can achieve resilience through either anti-signature or architectural stealth [40]. Anti-signature stealth entails the capability of manipulating the characteristics of bot-generated traffic to mask features that could be observed by signature-based detectors. On the other hand, architectural stealth entails the capability of establishing an overlay network that minimizes exposure of malicious traffic to detectors. For these reasons, botnets have recently gained significant attention in both the industry and the research community.

One promising approach to botnet detection and mitigation relies on moving-target defense (MTD), a novel and game-changing approach to cyber defense, which is part of the broader trend towards Adaptive Cyber Defense (ACD). MTD has the potential to create asymmetric uncertainty, providing the defender with a tactical advantage over the attacker [18]. Cyber attacks are typically preceded by a reconnaissance phase in which adversaries gather critical information about the target system, including network topology, service dependencies, and unpatched vulnerabilities. System and network configurations are typically static, and do not reconfigure, adapt, or regenerate except in deterministic ways to support maintenance and uptime requirements. In such a static scenario, it is only a matter of time for malicious actors to acquire accurate knowledge about the target system, engineer reliable exploits, and plan their attacks. To address this systemic weakness, MTD techniques are designed to continuously change or shift a system's attack surface [18], which has been formally defined as the *"subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack"* [23]. Continuously reshaping a system's attack surface increases complexity and cost for malicious actors, forcing them to continuously reassess their cyber operations.

In this chapter, we present a holistic, ACD-based approach to botnet detection and mitigation. To dominate the complexity of the problem, we decompose it into three related sub-problems, and tackle them individually. In particular, we presents solutions to (i) optimally deploy a set of detectors,

(ii) identify botnet traffic, and (iii) reduce the overall lifetime of a botnet. We validate our approach through simulation and experiments, and show that our solution is effective in mitigating botnet activity.

The remainder of the chapter is organized as follows. Section 2 discusses related work. Section 3 briefly discusses the threat model and our assumptions, whereas Sect. 4 provides an overview of the research challenges we are addressing. Then, Sects. 5, 7, and 8 discuss the three related challenges and corresponding solutions in detail. Finally, Sect. 9 gives some concluding remarks and indicates directions for future work.

2 Related Work

In response to botnet-borne threats, researchers have developed many different detection mechanisms. The performance of these mechanisms primarily depends on the set of features used to identify malicious traffic. Current research mostly focuses on studying a combination of packet-based, time-based, and behavior-based features to isolate bot traffic from the traffic mix [5]. For instance, BotHunter [16] exploits the sequence of messages between bots and a command and control (C&C) server in a centralized botnet architecture, while Zhang *et al.* exploit a combination of packet-based and time-based features to identify hosts that may potentially belong to a P2P botnet [49]. However, as the accuracy of feature-based detection techniques improves over time, botnets respond with more advanced evasion techniques [38]. On the other hand, architectural stealth techniques aim at building topology-aware botnets to reduce exposure of malicious traffic to detectors. They exploit the fact that detection mechanisms are likely to be deployed on nodes where they can monitor all traffic entering or exiting the network (e.g., network gateways) or significant volumes of internal traffic (e.g., routers). Thus, botmasters can design stealthy communication architectures capable of evading detection techniques such as those described in [16,49] by minimizing observable bot traffic. To this end, Sweeney studied the importance of the physical location of bots (referred to as the *cyber high ground*) to perform stealthy missions such as data exfiltration, and designed a P2P botnet that can effectively exfiltrate data from a network's mission-critical nodes, while maintaining a small network footprint [40].

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 [3,20,34,46], where 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 [3,34] 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 [20,46] develop optimal placement strategies to minimize the attacker's control over the target server.

3 Threat Model and Assumptions

In our threat model, the attacker's ultimate goal is to exfiltrate data from mission-critical nodes, while remaining stealthy and persisting in the system for an extend period of time. To this end, we make the following assumptions, based on previous work by Sweeney [40].

- The attacker can discover the topology of the network, and is aware of what nodes are mission-critical. Reports by Kaspersky labs [19] and Mandiant [1] show that threat actors can infiltrate an organization's network and persist in the system for several years, mapping out the organization and exfiltrating sensitive data and valuable intellectual property.
- Exfiltrating large volumes of data generates abnormally large network flows which in turn may trigger alerts. To avoid detection, the attacker partitions the data to be exfiltrated into m segments $d_1, d_2, \ldots, d_m$, and transmits these segments over a temporal span $\mathscr{T} = \langle t_1, t_2, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, i.e., at each time point t_i, the attacker transmits a data segment d_i to a C&C site. The attacker is said to have *successfully exfiltrated from a mission-critical node* if and only if all the m data segments are exfiltrated by time t_m.
- The attacker is aware of the detector placement strategy employed by the defender.

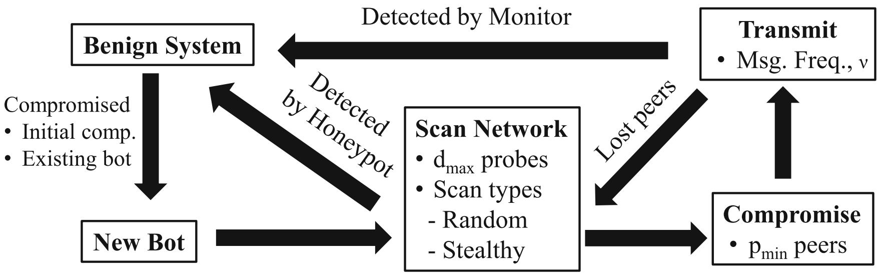


Fig. 1. Lifecycle of a bot

We model the lifecycle of a bot as shown in Fig. 1. It begins when a benign system in 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 [45]. 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 of 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, this activity may be flagged as anomalous in an enterprise network. In the stealthy scanning strategy, on the other hand, bots first enumerate active connections of the underlying hosts and then send discovery probes only to these machines. Several classes of malware employ this mechanism to move laterally through the network [2]. 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 target machines, a 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. 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 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 Research Challenges

Stealthy botnets, due to their ability to evade traditional defenses, are intrinsically difficult to detect and mitigate. Their very nature makes them extremely powerful tools in the hands of APT actors, whose primary goal is to remain undetected and persist within target systems for extended periods of time. From a defensive perspective, the problem of detecting and mitigating stealthy botnets can be broken down into three closely related challenges – captured in the infographic of Fig. 2 – which can be addressed separately, yet in a coordinated fashion, to dominate the complexity of the problem.

In real-world scenarios, it is unfeasible to monitor all network activity in depth. Thus, the first challenge is to deploy a limited number of detectors so as to maximize the likelihood of intercepting botnet-related activity. The second challenge is to analyze traffic collected by deployed detectors in order to isolate malicious data flows and identify bots responsible for those flows. This capability would enable the defender to take down bots and restore compromised hosts to a secure state. Finally, the third challenge is to reduce the overall lifetime of a botnet. Taking down some of the bots in a botnet is only a temporary solution, as residual bots can compromise additional machines to restore the full functionality

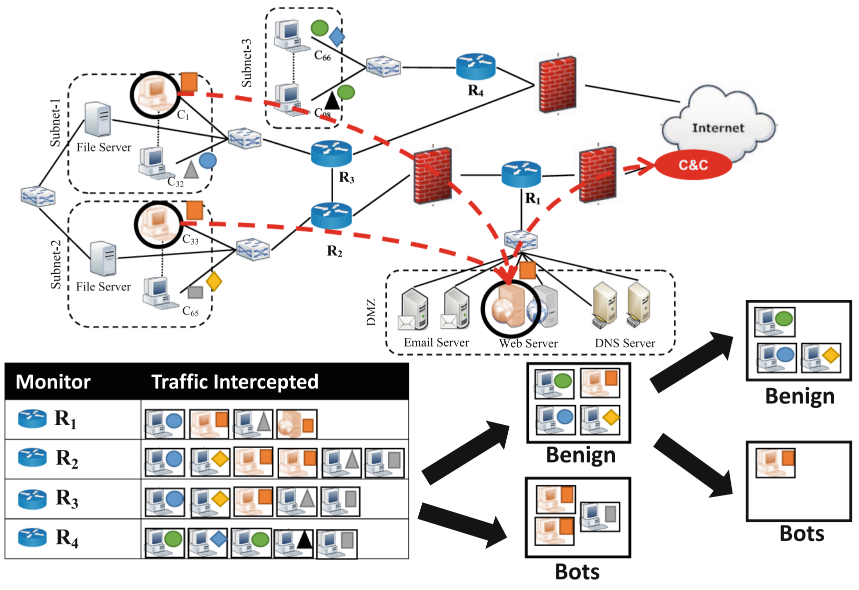


Fig. 2. Botnet detection and mitigation in a sample network scenario

of the botnet. In practice, the defender can claim victory only when every bot has been removed from the system. To achieve this goal, we need to develop a process that iterates through multiple cycles of data collection, analysis, and response, until no further botnet activity can be detected.

5 Detector Placement

As mentioned earlier, collecting and analyzing all traffic traversing every router in a complex network would prove to be a daunting task. Whether the analysis and detection capabilities are distributed or centralized, this solution would not only incur a significant computational cost, but could also increase false positive rates. To address this challenge, we have developed several heuristic detector placement strategies that select subsets of a network's nodes based on their centrality [41]. To this aim, we model a network as a graph, where nodes correspond to hosts and network devices, and edges represent the connectivity between them. A centrality measure captures important properties of a graph to determine how important or central each node is with respect to a given function or mission, which in our case is the botnet's mission to exfiltrate data from the target network to an external server. Centrality measures have found application in a wide range of domains, from social networks to citation ranking, and a prime example is PageRank, the algorithm used by Google to measure the importance of webpages.

Architecturally stealthy botnets are aware of the target network's topology and can potentially discover the location of detectors. Based on this information, attackers attempt to create detector-free paths within the network by compromising additional hosts to be used as proxies. To overcome the limitations of a purely static solution, we can adopt an MTD approach and periodically alter the placement of detectors, so as to introduce uncertainty about their location and force the attacker to perform additional, potentially detectable actions to maintain a functional botnet.

5.1 Preliminary Definitions

Let $G = (V, E)$ be a graph representing the physical topology of the network, where V is a set of network elements (e.g., routers and end hosts) and E captures the connectivity between them. Let $N = \{h_1, h_2, ..., h_n\}$ be a set of mission-critical hosts. Let Π_G denote the set of all simple paths $\pi(v_i, v_j)$ between any pair of nodes $(v_i, v_j) \in V \times V$. Traffic between any two nodes is routed using a routing algorithm, which can be formally defined as a mapping $R_A : V \times V \rightarrow \Pi_G$, such that

$$R_A(u, v) = \langle u, z_1, z_2, \ldots, z_r, v\rangle, \forall (u, v) \in V \times V$$

where $\langle u, z_1, z_2, \ldots, z_r, v\rangle \in \Pi_G$ is the path followed by traffic from u to v. Note that we slightly abuse notation and, for the sake of presentation, we may treat a path $\pi \in \Pi_G$ as a set of nodes. Although most routing algorithms attempt to route traffic along the shortest path from source to destination, our approach does not rely on the assumption that traffic is routed along the shortest path, but rather on the more general assumption that we can predict what paths the algorithm will select for routing traffic. However, for the sake of presentation, and without limiting the generality of our approach, we do assume that the networks being studied implement a shortest path routing algorithm.

In order to exfiltrate data from the set N of mission-critical nodes, the attacker compromises a set $B \subseteq V$ of network nodes – referred to as *bots* and such that $B \cap N \neq \emptyset$ – and creates an overlay network to forward captured data to a remote C&C server.

Definition 1 (Exfiltration Path). Given the set $N \subseteq V$ of mission-critical nodes for a network $G = (V, E)$ and a set $B \subseteq V$ of nodes controlled by the attacker, an *overlay path* is a sequence $\pi_o(b_0, \mathsf{C\&C}) = \langle b_0, b_1, b_2, \ldots, b_r, \mathsf{C\&C}\rangle$ of bots – with $b_0 \in N \cap B$ and $b_i \in B$ for each $i \in [1, r]$ – chosen by the attacker to forward traffic from mission-critical node b_0 to a remote C&C site. The *exfiltration path* corresponding to an overlay path $\pi_o(b_0, \mathsf{C\&C})$ is the sequence of nodes in V traversed by traffic exfiltrated through $\pi_o(b_0, \mathsf{C\&C})$, and it is defined as:

$$\pi_e(b_0, \mathsf{C\&C}) = \langle b_0, v_1^0, v_2^0, \ldots, v_{l_0}^0, b_1, v_1^1, v_2^1, \ldots, v_{l_1}^1, b_2, \ldots, b_r, v_1^r, v_2^r, \ldots, v_{l_r}^r, \mathsf{C\&C}\rangle$$

where $R_A(b_i, b_{i+1}) = \langle b_i, v_1^i, v_2^i, \ldots, v_{l_i}^i, b_{i+1}\rangle, \forall i \in [0, r-1]$ is the routing path from b_i to b_{i+1} and $R_A(b_r, \mathsf{C\&C}) = \langle b_r, v_1^r, v_2^r, \ldots, v_{l_r}^r, \mathsf{C\&C}\rangle$ is the path from b_r to C&C.

Example 1. In the example of Fig. 3, if $v_5 \in B$ is a bot and the attacker chooses to exfiltrate traffic through the overlay path $\pi_o(v_1, \mathsf{C\&C}) = \langle v_1, v_5, \mathsf{C\&C}\rangle$, then the corresponding exfiltration path is $\pi_e(v_2, \mathsf{C\&C}) = \langle v_1, v_3, v_5, v_8, \mathsf{C\&C}\rangle$.

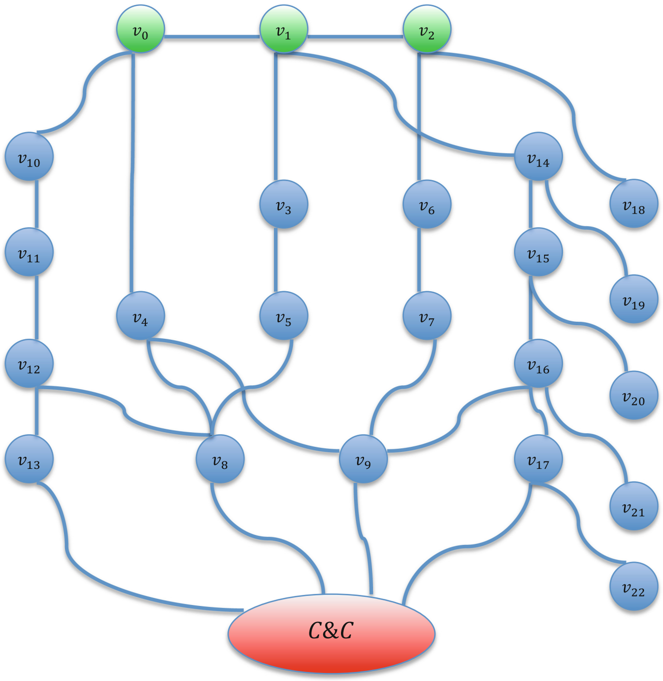


Fig. 3. Example of network graph

For a given set of mission-critical nodes N, the defender's objective is to intercept and detect exfiltration traffic. In order to monitor the network for botnet activity, the defender can deploy detectors on a subset of nodes $D \subset V$. One of several botnet detection mechanisms can be used to detect botnet activity [15,16,48,49]. These detection mechanisms leverage the fact that bots need to communicate with their peers or the $\mathsf{C\&C}$ server to relay captured data.

Definition 2 (Detection). Given the set $N \subseteq V$ of mission-critical nodes for a network $G = (V, E)$ and the set $B \subseteq V$ of bots controlled by the attacker, an exfiltration attempt over an exfiltration path $\pi_e = \langle v_0, v_1, v_2, \ldots, v_r, \mathsf{C\&C}\rangle$, with $v_0 \in N \cap B$, is said to be *detected* iff the exfiltrated traffic traverses a detector node, that is, $\exists d \in D$ s.t. $d \in \pi_e$. A botnet is said to be *stealthy* with respect to N, iff no exfiltration path between nodes in $N \cap B$ and a $\mathsf{C\&C}$ site can be detected.

Unfortunately, existing detection mechanisms suffer from false positives and false negatives, therefore exfiltration attempts may go undetected even when a detector is placed on a node along the exfiltration path. However, a prudent attacker will opt for creating more bots in order to establish a detector-free path, rather than having the traffic routed through detectors, irrespective of their false negative rate. Based on these considerations, and in order to simplify the presentation of our analysis, we ignore the accuracy of detectors and assume that any exfiltration attempt going through a detector node is detected. We will reconsider the problem of detecting exfiltration traffic later in this chapter.

In order to exfiltrate data from a mission-critical node $h \in N$ to a $\mathsf{C\&C}$ site in a stealthy manner, the attacker must identify a detector-free path $\pi_e^*(h, \mathsf{C\&C}) \in \Pi_G$, and forward data through it. The set of all detector-free paths represents the *exfiltration surface* of the network, which can be formally defined as follows.

Definition 3 (Exfiltration Surface). Given the set $N \subseteq V$ of mission-critical nodes for a network $G = (V, E)$, let $D \subset V$ be a set of detector nodes. The *exfiltration surface* of G with respect to D is the set of detector-free paths $\psi_D = \{\pi_e(h, \mathsf{C\&C}) \mid h \in N \wedge \pi_e(h, \mathsf{C\&C}) \in \Pi_G \wedge \pi_e(h, \mathsf{C\&C}) \cap D = \emptyset\}$. We use Ψ to denote the set of all possible exfiltration surfaces from mission-critical nodes N to $\mathsf{C\&C}$ sites.

In [42], we proposed an approach to deploy detectors on selected network nodes, so as to reduce the exfiltration surface by either completely disrupting communication between bots and $\mathsf{C\&C}$ nodes, or at least forcing the attacker to create more bots, thereby increasing the botnet's footprint and the likelihood of detection. As the detector placement problem is intractable, we proposed heuristics based on several centrality measures. Specifically, we showed that the *iterative mission-betweenness centrality strategy* yields the best results. In this strategy, after a node has been selected as a detector, the mission-betweenness centrality of all non-detector nodes is recomputed, and the node with the highest centrality is chosen for placing an additional detector. In practice, this approach prevents two or more detectors from being placed on the same high-centrality path. Although this strategy significantly increases an attacker's effort, the resulting exfiltration surface is static. Therefore, a persistent attacker can gather enough information to precompute the exfiltration surface of the target system and identify a detector-free path to exfiltrate data. We overcome this limitation by designing detector placement strategies that *dynamically* change the exfiltration surface by continually altering the placement of detectors, as discussed in the following subsections.

5.2 Defender's Model

In our defender's model, we consider a resource-constrained setting where the defender can only deploy k detectors. In practice, an upper bound on the number of detectors can be determined by considering the number of systems in the network that can perform detection tasks without impacting the performance of applications running on them. A dual problem is that of minimizing the number of detectors needed to satisfy predefined security requirements. In the following, we formally define the notion of *detector placement*.

We assume that the defender is aware of the location of potential C&C sites. For an enterprise network, C&C locations could include any destination outside the network perimeter. Similarly, for an ISP network, potential C&C sites could be located outside the administered domain. Furthermore, it has been shown that certain IP address ranges are known to participate in malicious campaigns [8,28]. This information can be leveraged to identify potential C&C locations, but, due to the conservative estimate on the location of potential C&C sites, simply blacklisting traffic to these locations would adversely affect legitimate users.

Definition 4 (k-placement). Given a network $G = (V, E)$, a *k-placement* over G is a mapping $pl : V \rightarrow \{0, 1\}$ such that $\sum_{v \in V} pl(v) = k$. Vertices v such that $pl(v) = 1$ are called *detector nodes.* We will use $\mathscr{P}_k$ to denote the set of all possible k-placements.

To address the limitations of a static placement and increase the probability of detection, we can continually shift the exfiltration surface by dynamically changing the location of detectors. In our analysis, we discretize time as a finite sequence of integers $\mathscr{T} = \langle t_1, t_2, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, with $m \in \mathbb{N}$, such that for all $1 \leq i < m$, $t_i < t_{i+1}$, and model how placements can evolve over time.

Definition 5 (Temporal k-placement). A *temporal k-placement* is a function $tp : \mathscr{T} \rightarrow \mathscr{P}_k$. We will use $\mathscr{P}_k^T$ to denote the set of all possible temporal k-placements.

Intuitively, for each time point in $\mathscr{T}$, a temporal k-placement deploys detectors on k network nodes. In order to create uncertainty for the attacker with respect to the location of detectors, we choose temporal k-placement functions by using a probability distribution over all temporal k-placements.

Definition 6 (Temporal probabilistic k-placement). A *temporal probabilistic k-placement* (tp-k-placement) is a function $\tau : \mathscr{P}_k^T \rightarrow [0, 1]$ such that $\sum\limits_{tp \in \mathscr{P}_k^T} \tau(tp) = 1$.

Example 2. Figure 4 shows an example of temporal probabilistic k-placement τ for the graph of Fig. 3 and for $k = 2$. Each table in the figure represents a different temporal k-placement tp[1]. Note that $\sum_{tp \in \mathscr{P}_k^T} \tau(tp) = 1$. For any given temporal k-placement tp, the i-th column in the corresponding table – with $i \in \{1, 2, \ldots, m\}$ – represents the k-placement pl that tp associates with time point t_i. Note that, for each k-placement pl, $\sum_{v \in V} pl(v) = k$. This example assumes that only certain nodes, namely v_3, v_4, v_5, and v_6, can host detectors.

[1] For the sake of presentation, we assume that those shown are the only possible temporal k-placements in $\mathscr{P}_k^T$).

$$\tau\left(\begin{array}{|c|c|c|c|c|c|}\hline & t_1 & t_2 & t_3 & \dots & t_m \\ \hline v_3 & 1 & 1 & 0 & \dots & 1 \\ v_4 & 1 & 0 & 1 & \dots & 1 \\ v_5 & 0 & 0 & 1 & \dots & 0 \\ v_6 & 0 & 1 & 0 & \dots & 0 \\ \hline\end{array}\right) = 0.3 \quad \tau\left(\begin{array}{|c|c|c|c|c|c|}\hline & t_1 & t_2 & t_3 & \dots & t_m \\ \hline v_3 & 0 & 1 & 1 & \dots & 1 \\ v_4 & 1 & 0 & 0 & \dots & 0 \\ v_5 & 0 & 1 & 0 & \dots & 0 \\ v_6 & 1 & 0 & 1 & \dots & 1 \\ \hline\end{array}\right) = 0.2 \quad \tau\left(\begin{array}{|c|c|c|c|c|c|}\hline & t_1 & t_2 & t_3 & \dots & t_m \\ \hline v_3 & 0 & 0 & 0 & \dots & 1 \\ v_4 & 0 & 1 & 1 & \dots & 1 \\ v_5 & 1 & 1 & 0 & \dots & 0 \\ v_6 & 1 & 0 & 1 & \dots & 0 \\ \hline\end{array}\right) = 0.5$$

Fig. 4. Example of temporal probabilistic k-placement

Let the indicator random variable $I^v_{t_i}$ be associated with the event that node v is chosen as a detector at time t_i. Given a temporal probabilistic k-placement τ, the probability with which a node $v \in V$ will be chosen as a detector at time t_i can be derived as

$$pr^v_{t_i} = Pr(I^v_{t_i} = 1 \mid \tau) = \sum_{tp \in \mathscr{P}^T_k \ s.t.\ (\exists pl \in \mathscr{P}_k)(tp(t_i) = pl \wedge pl(v) = 1)} \tau(tp) \qquad (1)$$

Thus, at time t_i, the defender selects k nodes for detector placement by sampling from the distribution defined by Eq. 1. We denote such a strategy as $\mathscr{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$.

5.3 Metrics

To evaluate the performance of a defender strategy, we present two metrics: the *minimum detection probability* and the *attacker's uncertainty*. The minimum detection probability provides a theoretical lower bound on the probability that an exfiltration activity is detected due to the detector placement strategy. On the other hand, the attacker's uncertainty is measured as the entropy in the location of the detectors from the attacker's point of view: the higher the entropy, the higher the attacker's effort required to discover the location of detectors.

5.3.1 Minimum Detection Probability

As mentioned earlier, to be succssfull, the attacker needs to exfiltrate data segments $d_1, d_2, \dots, d_m$ over a temporal span $\mathscr{T} = \langle t_1, t_2, \dots, t_m \rangle \subseteq \mathbb{N}^m$, while remaining undetected. At each time point t_i, the defender chooses a strategy, $\mathscr{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$ and samples k nodes without replacement. Let D_{t_i} denote the set of detectors at time t_i. Following defender's placement of detectors, the attacker begins exfiltrating data segment d_i. For a chosen overlay path $\pi_o(h, \mathsf{C\&C})$, the traffic will traverse the corresponding exfiltration path $\pi_e(h, \mathsf{C\&C}) = \langle h, v_{i_1}, v_{i_2}, \dots, v_{i_l}, \mathsf{C\&C} \rangle$, with $h \in N$. Therefore, the probability that the attacker's exfiltration of data segment d_i is detected is given by:

$$detectPr(\mathscr{D}_{t_i}, d_i, \pi_e(h, \mathsf{C\&C})) = 1 - \prod_{v \in \pi_e(h, \mathsf{C\&C}) \setminus \{h, \mathsf{C\&C}\}} \left(1 - pr^v_{t_i}\right) \qquad (2)$$

Algorithm 1. *minimumDetectionProb*$(G, \mathscr{D}_{t_i}, N, \mathsf{C\&C})$

```
Input: a connectivity graph $G(V, E)$, a defender strategy, $\mathscr{D}_{t_i} \sim \{pr^v_{t_i}\}$, a set $N \subseteq V$ of mission-
    critical nodes, a potential $\mathsf{C\&C}$ location
Output: the minimum detection probability of strategy $\mathscr{D}_{t_i}$ at time $t_i$ for graph $G(V, E)$ with
    respect to mission-critical nodes $N$ and the potential $\mathsf{C\&C}$ location
1: $H(V', E') \leftarrow$ dual graph of $G(V, E)$, where $V' = E$ and $(e, f) \in E'$ iff $e$ and $f$ share a common
   vertex $v \in V$
2: $b \leftarrow \epsilon$        // an arbitrarily small value
3: for all $(e, f) \in E'$ do
4:     $v \leftarrow$ the common vertex of $e$ and $f$ in $V$
5:     if $pr^v_{t_i} < 1$ then
6:         $W'(e, f) \leftarrow \log_b(1 - pr^v_{t_i})$
7:     else
8:         $W'(e, f) \leftarrow \infty$
9:     end if
10: end for
11: // $\forall v \in V$, let $\mathscr{E}(v)$ denote the set $\{e \mid e \in E \wedge e$ is incident on $v \in V\}$
12: for all $h$ in $N$ do
13:     for all $e$ in $\mathscr{E}(h)$ do
14:         for all $c$ in $\mathscr{E}(\mathsf{C\&C})$ do
15:             $S \leftarrow$ length of the shortest path from $e$ to $c$ in $H$
16:             $detectPr(h, e, c) \leftarrow 1 - b^S$
17:         end for
18:     end for
19:     $detectPr(h) \leftarrow \min_{(e,c) \in \mathscr{E}(h) \times \mathscr{E}(\mathsf{C\&C})} (detectPr(h, e, c))$
20: end for
21: return $\min_{h \in N} (detectPr(h))$
```

A rational attacker – who is aware of the defender's strategy – will choose a path that minimizes the probability of detection. Therefore, the path chosen by the attacker to exfiltrate d_i is:

$$\pi_e^{i^*}(h, \mathsf{C\&C}) = \underset{\pi_e(h,\mathsf{C\&C})}{\operatorname{argmin}} \left(detectPr\left(\mathscr{D}_{t_i}, d_i, \pi_e\left(h, \mathsf{C\&C}\right)\right)\right) \tag{3}$$

In other words, Eq. 3 can be used to compute the minimum detection probability that a defender strategy $\mathscr{D}_{t_i}$ can guarantee at time t_i. Finally, an exfiltration activity is said to be *detected* when *any* of the m data flows is detected. Therefore, the minimum probability with which a strategy $\mathscr{D}_{t_i}$ detects an exfiltration activity is given by

$$eDetectPr\left(\{\mathscr{D}_{t_i}\}_{i \in [1,m]}\right) = 1 - \prod_{d_i} \left(1 - \min_{\pi_e}\left(detectPr\left(\mathscr{D}_{t_i}, d_i, \pi_e\right)\right)\right) \tag{4}$$

Given a graph $G(V, E)$, the minimum detection probability of a strategy $\mathscr{D}_{t_i}$ at time t_i – i.e., $\min_{\pi_e}\left(detectPr\left(\mathscr{D}_{t_i}, d_i, \pi_e\right)\right)$ – can be computed using Algorithm 1. At a high-level, the algorithm transforms the graph $G(V, E)$ into a weighted dual graph $H(V', E')$ in which the edge weights are a function of the probability that the corresponding vertex in $G(V, E)$ does not host a detector. Specifically, at time t_i, the path detection probability over any path $\pi_e(u, v)$ (given by Eq. 2) can be re-written as $1 - b^S$, where $S = \left(\sum_{x \in \pi_e(u,v)} \log_b\left(1 - pr^x_{t_i}\right)\right)$ and b is an arbitrarily small value. Here, b^S is the upper bound on the probability that the path $\pi_e(u, v)$ will be free of detectors. Therefore, each edge in E' corresponding to a node $v \in V$ is assigned a weight $\log_b(1 - pr^v_{t_i})$. Following this assignment, the algorithm determines the shortest path length between the vertices in V' that correspond to edges incident on the mission-critical and $\mathsf{C\&C}$ vertices in V. The shortest path length represents the maximum probability that data exfiltration is not detected, and the vertices in V corresponding to edges on this shortest path form the path $\pi_e^{i^*}(h, \mathsf{C\&C})$.

In particular, after generating the dual graph $H(V', E')$ on Line 1, Algorithm 1 assigns weights to all the edges $(e, f) \in E'$ based on the probability $pr^v_{t_i}$ that the corresponding vertex $v \in V$ is chosen for detector placement (Lines 3–10). If a detector is placed on vertex $v \in V$ with probability 1, then any exfiltration over a path that contains v will be detected. A rational attacker will avoid such paths and hence the algorithm sets the weight of the corresponding edges in E' as ∞ (Line 8). On the other hand, if the probability is less than 1, then the corresponding edge is assigned a weight $\log_b(1 - pr^v_{t_i})$ (Line 6). Next, Line 15 computes the length of the shortest path between vertices e and c in V', which correspond to the edges in E that are incident on mission-critical nodes and $\mathsf{C\&C}$, respectively. Finally, Line 16 computes the minimum detection probability over all the paths from a mission-critical node $h \in N$ to $\mathsf{C\&C}$ that traverse edges e and c. Line 19 computes the minimum detection probability for each mission-critical node h by considering all the paths to $\mathsf{C\&C}$. Finally, the minimum detection probability with respect to all mission-critical nodes in N for graph $G(V, E)$ is computed on Line 21.

In the worst case, Algorithm 1 takes $O(|E|^2)$ time to generate the edge-dual graph $H(V', E')$ as all pairs of edges in E are checked for a common vertex. As a result, in the worst case $|E'| = O(|E|^2)$. Lines 3–10 run in time $O(|E'|)$ and Line 11 can be computed in time $O(|V|^2)$ by traversing the adjacency matrix of G. To compute the shortest paths between vertices in H (Line 15) that correspond to mission-critical node h and $\mathsf{C\&C}$ in G, we can leverage the Fibonacci heap implementation of Dijkstra's single-source shortest path algorithm [10]. The complexity of computing the shortest path lengths for a node $h \in N$ (Lines 13–19) is given by $O\left(\mathscr{E}(h) \cdot (|E'| + |V'| \log |V'|)\right)$. Therefore, in the worst case, the time complexity for computing the shortest path lengths for all mission-critical nodes is $O(|E| \cdot (|E|^2 + |E| \cdot \log |E|))$.

As the time complexity of the algorithm is dominated by the shortest paths computation, the time complexity is $O(|E| \cdot (|E|^2 + |E| \cdot \log |E|))$. The worst-case time complexity for computing the minimum detection probability is $O(|V|^6)$. However, for practical network topologies, our simulation results indicate that the processing time does not exceed $O(|V|^3)$.

5.4 Attacker's Uncertainty

Probabilistic deployment of detectors introduces uncertainty for the attacker with respect to the location of the detectors. Depending upon the nature of the deployed detector (either active or passive), the attacker may progressively learn the location of these detectors through probing. For instance, if an enterprise network deploys an active IDS, a simple probing strategy could consist in sending malicious packets to a node suspected of hosting a detector and, depending on whether the node accepts or rejects the packets, the attacker can determine the node's detection state. In an ISP network, an attacker can leverage probing strategies described by Shinoda *et al.* [35], and by Shmatikov and Wang [36] to identify the presence of detectors in a network.

The uncertainty introduced by a dynamic placement strategy can be quantified by measuring the entropy in locating the detectors at any time t_i. Let $X^-_{t_i}$ be the random variable that maps the set V of potential locations to the corresponding probability of being chosen for detector placement. Therefore, the entropy due to a strategy $\mathscr{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$ is given by:

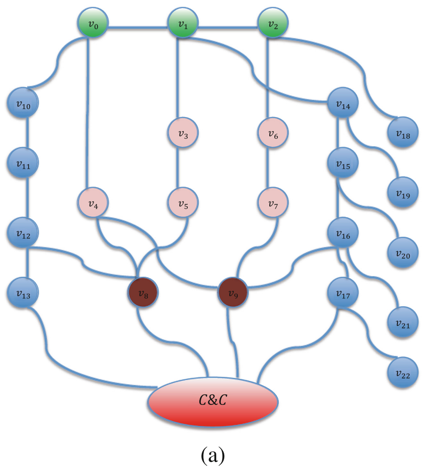
$$H(X^-_{t_i} \mid \mathscr{D}_{t_i}) = -\sum_{x \in V} P(X^-_{t_i} = x) \log(P(X^-_{t_i} = x)) \tag{5}$$

where $\log(P(X^-_{t_i} = x)) = 0$, when $P(X^-_{t_i} = x) = 0$. Note that, based on the above definition of entropy, higher entropy translates into a greater advantage for the defender over the attacker.

5.5 Defender's Strategies

To illustrate the effectiveness of different defender strategies, consider again the network in Fig. 3, which includes mission-critical nodes $N = \{v_0, v_1, v_2\}$. The attacker's objective is to exfiltrate data from any node in N to a C&C server. To protect mission-critical nodes from data exfiltration, we consider the following strategies for placing k detectors.

- *Static Iterative Centrality Placement.* In this strategy, the defender chooses nodes based on the iterative mission-betweenness centrality algorithm proposed in [42]. The defender first computes the mission-betweenness centrality of a node v as $C_M(v) = \sum_{(s,t) \in N \times \text{C\&C} \; s.t. \; v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the number of shortest paths between s and t and $\sigma_{st}(v)$ is the number of those paths that go through v. Upon computing the mission-betweenness centrality for all the nodes, the defender chooses the node with the highest centrality for detector placement. For each subsequent detector placement, the centrality $C_M(v)$ of all non-detector nodes is re-computed and the node with the highest centrality among the non-detector nodes is picked for placing the next detector. In the example of Fig. 3, assume that the defender can place $k = 2$ detectors. Then, node v_9 (or v_8) will be chosen to the place the first detector followed by v_8 (or v_9) to place the second detector.
- *Uniform Random Placement.* The static nature of the above strategy enables an attacker to pre-compute the location of detectors and compromise nodes along a detector-free path. Therefore, in order to create uncertainty about the exact location of detectors, in this strategy, the defender chooses k nodes to place detectors uniformly at random.



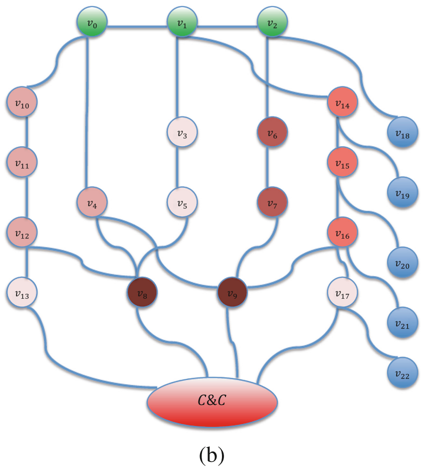


Fig. 5. Candidate detector locations for the network of Fig. 3, based on the (a) centrality-weighted strategy, and (b) expanded centrality-weighted strategy

- *Centrality-Weighted Placement.* Although the uniform random strategy introduces uncertainty for an attacker, it may consider nodes that do not lie on any simple path between mission-critical nodes and C&C. As a result, the uniform strategy may provide a low minimum detection probability. In this strategy, to improve detection guarantees, the defender places k detectors by randomly choosing nodes according to a probability distribution that weights nodes based on their mission-betweenness centrality, i.e., nodes with higher values of $C_M(v)$ have more chances of being chosen for detector placement. For the example shown in Fig. 3, the nodes colored in brown in Fig. 5a are the only nodes considered for detector placement by this strategy (the darkness is proportional to the relative weight of the corresponding node).
- *Expanded Centrality-Weighted Placement.* One of the major drawbacks of the centrality-weighted strategy is that coverage of the exfiltration surface is limited. In fact, that strategy considers only the nodes on the set of all shortest paths between the mission-critical nodes and C&C. In this strategy, the coverage of the exfiltration surface is expanded by considering all the nodes on paths that are up to δ times longer than the shortest paths. Let Π_e be the set of all such paths. The revised centrality of a node v is computed as $C_E(v) = \sum_{(s,t)\in N\times \mathsf{C\&C}\ s.t.\ v\neq t} \frac{\sigma'_{st}(v)}{\sigma'_{st}}$, where σ'_{st} is the number of simple paths in Π_e between s and t and $\sigma'_{st}(v)$ is the number of those paths that go through v. In the example of Fig. 3, when $\delta = 0.25$, the nodes colored in brown in Fig. 5b will be considered for randomizing the placement.

5.6 Simulation Results

We evaluated the proposed strategies using real ISP network topologies obtained from the Rocketfuel dataset [37] and synthetic topologies generated using graph-theoretic properties of typical ISP networks. The Rockfuel dataset provides router-level topologies for 10 ISP networks. For each network, Table 1 provides a summary of the total number of routers within the network and the number of external routers (located outside the ISP) to which the ISP routers are connected. As connections to external routers are outside the monitored domain, we considered a worst-case scenario in our simulations and assumed that all the external routers are potentially routing traffic to C&C servers.

Table 1. Summary of ISP networks from [37]

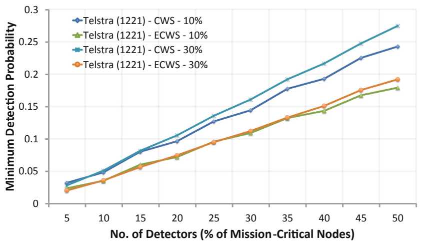
ASN	Name	No. of routers	No. of ext. routers	ASN	Name	No. of routers	No. of ext. routers
1221	Telstra	2998	329	3356	Level3	3447	1827
1239	Sprintlink	8341	1004	3967	Exodus	895	520
1755	Ebone	605	310	4755	VSNL	121	80
2914	Verio	7102	2432	6461	Abovenet	2720	2066
3257	Tiscali	855	444	7018	AT& T	10152	722

To study the influence of network topology on the performance of a strategy, we evaluated these strategies using simulated medium-scaled ISP networks comprising 3,000 nodes. At the router level, such networks are known to exhibit scale-free network properties wherein the degree distribution follows a power-law distribution. In order to accurately capture the connectivity of an ISP network at the router level, the BRITE network topology generator [26] was used to generate these networks. Ten such networks were considered, with mission-critical nodes varying between 10% and 30% of the network size and 500 C&C locations chosen at random for each network.

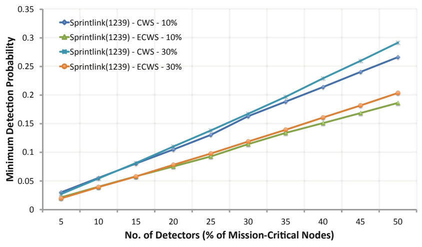
For the ISP networks from the Rocketfuel dataset, we varied the size of the detector set as a fraction of the number of mission-critical nodes, whereas, for synthetic topologies, we varied the size of the detector set as a fraction of the network size. These simulations were intended to study the impact on the amount of resources that a network administrator might be willing to invest (proportional to either the number of nodes that need to be protected or the size of the network) to detect exfiltration. In all our simulations, we set $\delta = 0.5$ for the expanded centrality-weighted strategy and tested the statistical significance of the results using paired t-tests at 95% confidence interval. For the sake of presentation, we show the results for a subset of the topologies from the Rocketfuel dataset.

5.6.1 Minimum Detection Probability

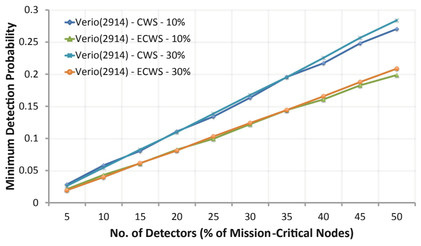
As illustrated in Figs. 6 and 7, the probability of detecting exfiltration attempts increases linearly with the number of detectors. We observed that variations in the detection probability for different synthetic networks were less than 1% and hence, for the sake of presentation, we only show mean values.



(a) Telstra



(b) Sprintlink



(c) Verio

Fig. 6. Minimum detection probability for different networks using centrality-weighted (CWS) and expanded centrality-weighted (ECWS) strategies

It can be observed that, independently of the number of mission-critical nodes and detectors, the centrality-weighted strategy outperforms the expanded centrality-weighted strategy. This trend can be attributed to the scale-free nature of the topology in which most of the paths traverse a small portion of the nodes. As the expanded strategy considers a larger number of paths and distributes the placement probability across the nodes on these paths, the nodes with high centrality will be chosen with a lower probability than in the case of the centrality-weighted strategy. In these simulations, we observed that the static iterative centrality strategy could not detect exfiltration of data segments in any of the networks as there was at least one detector-free path between one of the mission-critical nodes and a C&C location.

5.6.2 Attacker's Uncertainty

As mentioned earlier, among the detector placement strategies, the static iterative centrality strategy does not introduce any uncertainty for the attacker, whilst the uniform random strategy introduces the highest uncertainty in the location of detectors. To study the attacker's uncertainty in the location of detectors due to the proposed strategies, we computed the relative entropy

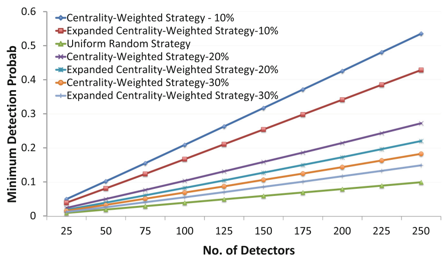
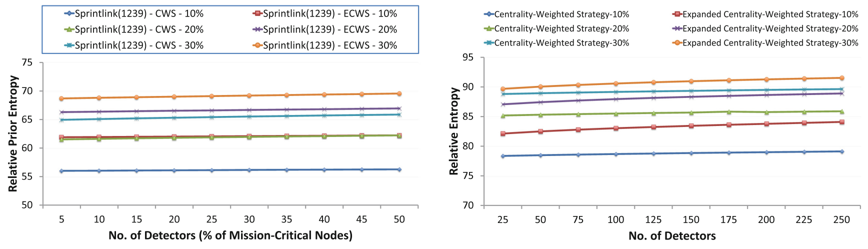


Fig. 7. Minimum detection probability for different strategies using synthetic topologies

introduced by the centrality-weighted and the expanded centrality-weighted strategies w.r.t. the uniform random strategy. As shown in Fig. 8, the centrality-weighted strategy and its expanded version create a level of uncertainty that lies in-between the two ends of the entropy spectrum. In particular, as the expanded strategy potentially considers more nodes, the number of combinations of detector locations, and hence the uncertainty introduced by it is higher than the centrality-weighted strategy. Therefore, for ISP networks, the choice of different centrality-weighted strategies offers a trade-off between entropy and detection probability.



(a) Sprintlink network (b) Synthetic network

Fig. 8. Relative increase in entropy for the attacker introduced by different strategies

5.6.3 Processing Time

In this section, we evaluate the performance of Algorithm 1 in computing the minimum detection probability and the performance of various detector placement strategies as a function of the network size. For each network size, we generated 10 ISP-type topologies, with 10% of the nodes being mission-critical and 500 C&C locations chosen randomly. We varied the number of detectors from 3% to 7% of the network size and observed similar trends in the processing time. For the sake of presentation, we only show the results when the total number of detectors is set to 3% of the network size. The processing time was averaged

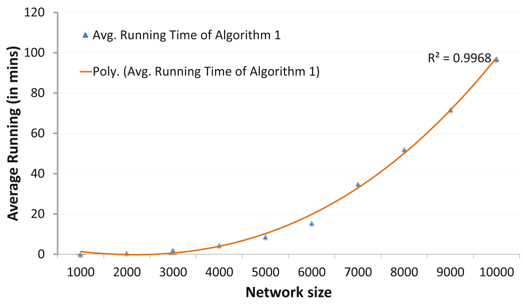


Fig. 9. Processing time for computing the minimum detection probability using Algorithm 1.

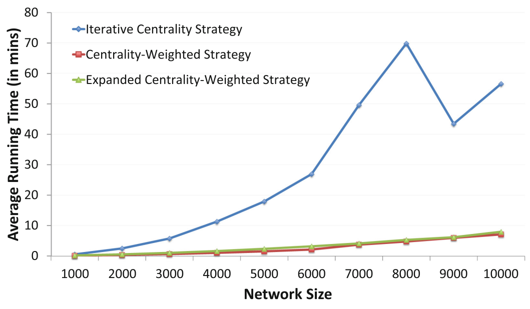


Fig. 10. Processing time for different detector placement strategies

over the 10 topologies for each network size. All experiments were conducted on an AMD Opteron processor with 4 GB memory running Ubuntu 12.04.

Although, in theory, the worst-case processing time of Algorithm 1 is $O(|V|^6)$, for practical network settings, it can be observed (see trend line in Fig. 9) that the execution time grows as $O(|V|^3)$ with an R^2 value of 0.9968. Finally, as shown in Fig. 10, the dynamic strategies compute the detector locations faster than its static alternative. This is because, the static iterative centrality strategy has to re-compute the shortest paths multiple times to determine the location of the detectors.

6 A Game-Theoretic Approach to Detector Placement

In this section, we consider a game-theoretic approach to design effective detection placement strategies. We formulate the botnet defense problem as a Stackelberg security game, thus accounting for the strategic response of attackers to deployed defenses. We consider two formulations of data exfiltration: (i) *uni-exfiltration*, where the source bot routes the stolen data along a single path designated by the attacker; and (ii) *broad-exfiltration*, where each bot propagates the received stolen data to all other bots in the network.

We propose algorithms to compute defense strategies for these data exfiltration formulations: ORANI (Optimal Resource Allocation for uni-exfiltration Interception) and ORABI (Optimal Resource Allocation for Broad-exfiltration Interception). Both ORANI and ORABI employ the double-oracle method [25] to control exploration of the exponential strategy spaces available to attacker and defender. Our main algorithmic contributions lie in defining mixed-integer linear programs (MILPs) and greedy heuristics for implementing the defender and attacker's best-response oracles.

6.1 Game Model: Uni-Exfiltration

Our game model for uni-exfiltration is built on the botnet model introduced in Sect. 5. We model the botnet defense problem as a *Stackelberg security game* (SSG) [21]. In such a game, the defender commits to a mixed (randomized) strategy to allocate limited security resources to protect important targets. The attacker then optimally chooses targets with respect to the distribution of defender allocations.

In the botnet exfiltration game, the attacker attempts to steal sensitive network data. Compromising a mission-critical node enables the attacker to steal data from that node. Compromising other nodes in the network helps the attacker relay the stolen data to a C&C server outside the network, which he controls, through a sequence of compromised nodes (bots) forming an overlay path. Routing between consecutive bots on this paths is beyond the attacker's control, and the sequence of all nodes traversed by exfiltrated traffic is referred to as an *exfiltration path*, as defined Sect. 5.1. In this game model, we consider the case in which the attacker does not divide stolen data into multiple segments before relaying it to C&C.

In our Stackelberg game model, the defender moves first by allocating detection resources, and the attacker responds with a plan for compromise and exfiltration to evade detection. The defender placement of detectors is randomized, so any attack plan succeeds with some probability. As formalized in Definition 2, an exfiltration attempt is detected if there is a detector on the exfiltration path.

Definition 7 (Strategy Space). The strategy spaces of the players are as follows:
***Defender*:** The defender has $K^d < |V|$ detection resources available for deployment on network nodes. We denote by $D = \{D_i \mid D_i \subseteq V, |D_i| \leq K^d\}$ the set of all *pure defense strategies* of the defender. Let $x = \{x_i\}$ be a *mixed strategy* of the defender where $x_i \in [0, 1]$ is the probability that the defender plays D_i, and $\sum_i x_i = 1$.
***Attacker*:** The attacker can compromise up to $K^a < |V|$ nodes. We denote by $A = \{A_j = (B_j, \Pi_j) \mid B_j \subseteq V, |B_j| \leq K^a, \Pi_j = \{\pi_j(c, C\&C) \mid c \in B_j \cap N\}\}$ the set of all pure strategies of the attacker. Each pure strategy A_j consists of: (i) B_j: a set of compromised nodes; and (ii) Π_j: a set of exfiltration paths over B_j.

A simple scenario of the botnet defense game is shown in Fig. 11. The model specification is completed by defining the payoff structure, which is zero-sum.

Definition 8 (Game Payoff). Each mission-critical node $c \in N$ is associated with a value, $r(c) > 0$, representing the importance of data stored at that node. Successfully exfiltrating data from c yields the attacker a payoff $r(c)$, and the defender receives a payoff $-r(c)$. For prevented exfiltrations, both players receive zero.

Note that the maximum achievable payoff for a defender is zero, obtained by preventing all exfiltration attempts. In general terms, let $U^a(D_i, A_j)$ denote the payoff to the attacker if the defender plays D_i and the attacker plays A_j. Since the game is zero-sum, the defender payoff $U^d(D_i, A_j) \equiv -U^a(D_i, A_j)$. The payoff can be decomposed across mission-critical nodes, which is formulated as follows:

$$U^a(D_i, A_j) \equiv \sum_{c \in N} r(c)h(c) \tag{6}$$

where $h(c)$ indicates if the attacker successfully exfiltrates data from node $c \in N$:

$$h(c) = \begin{cases} 1 & \text{if } c \in B_j \text{ and } D_i \cap \pi_j(c, C\&C) = \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The expected utility for the attacker when the defender plays mixed-strategy x is

$$U^a(x, A_j) = \sum\nolimits_i x_i U^a(D_i, A_j)$$

which is negated to obtain the expected defender payoff $U^d(x, A_j)$.

A defender mixed strategy that maximizes $U^d(x, A_j)$, given that the attacker plays a best response and breaks ties in favor of the defender, constitutes a *Strong Stackelberg Equilibrium* (SSE) of the game.

6.2 ORANI: An Algorithm for Uni-Exfiltration Games

In zero-sum games, the first mover's SSE strategy is also a maximin strategy [22]. Therefore, finding an optimal mixed defense strategy can be formulated as follows:

$$\max\nolimits_x U_*^d \tag{8}$$

$$\text{s.t. } U_*^d \le U^d(x, A_j), \ \forall j \tag{9}$$

$$\sum\nolimits_i x_i = 1, \ x_i \ge 0, \ \forall i, \tag{10}$$

where U_*^d is the defender's utility for playing mixed strategy x when the attacker best-responds. Constraint (9) ensures the attacker chooses an optimal action

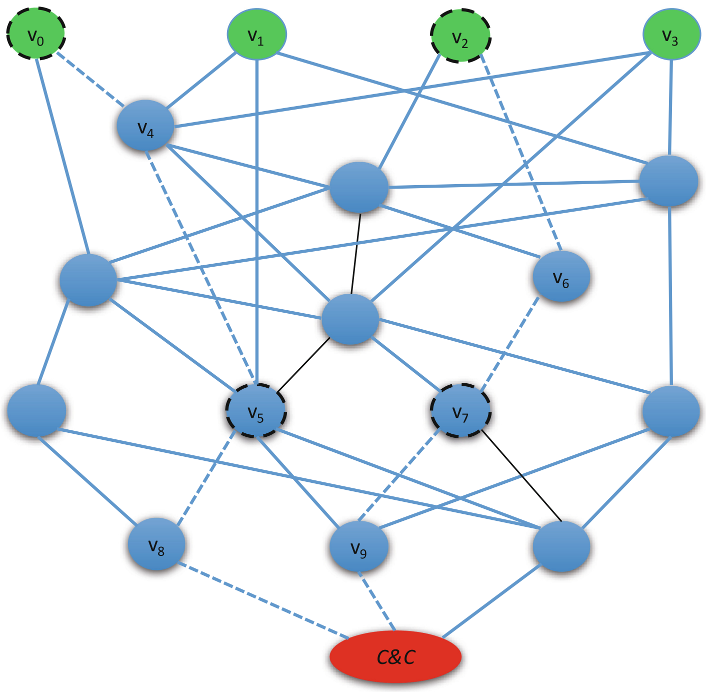


Fig. 11. An example scenario of the botnet exfiltration game with $(K^a = 4, K^d = 1)$. Four mission-critical nodes are $N = \{0, 1, 2, 3\}$. A possible pure strategy of the attacker A_j can be: (i) a set of compromised nodes $B_j = \{0, 2, 5, 7\}$; and (ii) a set of exfiltration paths $\Pi_j = \{\pi_j(0), \pi_j(2)\}$ to exfiltrate data from stealing bots 0 and 2 to the attacker's server $C\&C$. These exfiltration paths $\pi_j(0) = P(0, 5) \cup P(5, C\&C)$ and $\pi_j(2) = P(2, 7) \cup P(7, C\&C)$ relay stolen data via relaying bots 5 and 7 respectively, where $P(0, 5) = (0 \rightarrow 4 \rightarrow 5)$, $P(5, C\&C) = (5 \rightarrow 8 \rightarrow C\&C)$, $P(2, 7) = (2 \rightarrow 6 \rightarrow 7)$ and $P(7, C\&C) = (7 \rightarrow 9 \rightarrow C\&C)$ are routing paths fixed by the network system. If the defender allocates its one detector to node 9, the attacker fails at exfiltrating data from node 2 since $9 \in \pi_j(2)$ but succeeds from node 0 since $9 \notin \pi_j(0)$.

against x, that is, $U^d_* = \min_j U^d(x, A_j) = \max_j U^a(x, A_j)$. Solving (8–10) is computationally expensive due to the exponential number of pure strategies of the defender and the attacker. To overcome this computational challenge, ORANI applies the *double-oracle* method [17,25]. Algorithm 2 presents a sketch of ORANI.

ORANI starts by solving a maximin sub-game of (8–10) by considering only small seed subsets D and A of pure strategies for the defender and attacker (Line 2). Solving this sub-game yields a solution (x^*, a^*) with respect to the strategy subsets. ORANI iteratively adds new best pure strategies D_o and A_o to the current strategy sets D and A (Lines 3–5). These strategies D_o and A_o are chosen by the oracles to maximize the defender and attacker utility, respectively, against the current (in iteration) counterpart solution strategies a^* and x^*. This iterative

Algorithm 2. ORANI Algorithm Overview

```
Initialize the sets of pure strategies: $A = \{A_j\}$ and $D = \{D_i\}$ for some $j$ and $i$;

1: repeat
2:     $(x^*, a^*) = \text{MaximinCore}(D, A)$
3:     $D_o = \text{DefenderOracle}(a^*)$
4:     $A_o = \text{AttackerOracle}(x^*)$
5:     $A = A \cup \{A_o\}$, $D = D \cup \{D_o\}$
6: until converge
```

process continues until the solution *converges*: when no new pure strategy can be added to improve the players' utilities. At convergence, the latest solution (x^*, a^*) is an equilibrium of the game [25]. Following this general methodology, the specific contribution of ORANI is in defining MILPs representing the attacker and the defender oracle problems in botnet exfiltration games. These problems are proved to be NP-hard. We thus introduce greedy heuristics to approximately solve these oracle problems significantly faster. These MILPs and heuristics are described in detail in [30].

6.3 Data Broad-Exfiltration

In the botnet defense game model with respect to uni-exfiltration (Sect. 6.1), for each stealing bot, the attacker is assumed to only select a single exfiltration path from that bot to exfiltrate data. In this section, we study the botnet defense game model with respect to an alternative data broad-exfiltration. In particular, for every stealing bot, the attacker is able to broadcast the data stolen by this bot to all other compromised nodes via corresponding routing paths. Once receiving the stolen data, each compromised node continues to broadcast the data to all other compromised nodes. The game model for broad-exfiltration is motivated by the botnet models studied by Rossow *et al.* [32]. Overall, there is a higher chance that the attacker can successfully exfiltrate network data with broad-exfiltration compared to uni-exfiltration. In the following, we briefly describe the botnet defense game model with data broad-exfiltration. The corresponding algorithm, ORABI, to compute an optimal mixed defense strategy is built based upon the double oracle methodology. The algorithm's computation and complexity is described in detail in [30].

In the game model with data broad-exfiltration, the strategy space of the defender remains the same as shown in Sect. 6.1. On the other hand, since the attacker now can broadcast the data, we can abstractly represent each pure strategy of the attacker as a set of compromised nodes $A_j \equiv B_j$ only. Given a pair of pure strategies (D_i, B_j), we need to determine payoffs the players receive. Note that in the case of broad-exfiltration, given (D_i, B_j), the attacker succeeds in exfiltrating the stolen data from a stealing bot if there is an exfiltration path among all the possible exfiltration paths over the compromised set B_j from this bot to $C\&C$ which is not blocked by D_i. Therefore, the players receive a payoff computed as in (6) where the binary indicator $h(c)$ for each mission-critical node $c \in N$ is now determined as:

$$h(c) = \begin{cases} 1 & \text{if } \exists c \in B_j \ \& \ \exists \pi_j(c, C\&C) \\ & \text{ s.t. } D_i \cap \pi_j(c, C\&C) = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

In fact, when players plays (D_i, B_j), we can determine if there is an exfiltration path from a stealing bot $c \in B_j \cap N$ which is not blocked by D_i by using depth or breath-first search over the compromised set B_j, which runs in polynomial time.

6.4 Experiments

We evaluate both solution quality and runtime performance of our algorithms compared with previously proposed defense policies. We conduct experiments based on two different datasets: (i) synthetic network topologies generated using JGraphT[2], capturing scale-free properties [9] of many real-world networks; and (ii) real-world network topologies derived from the Rocket-fuel dataset [37]. Each data point in our results is averaged over 50 different samples of network topologies.

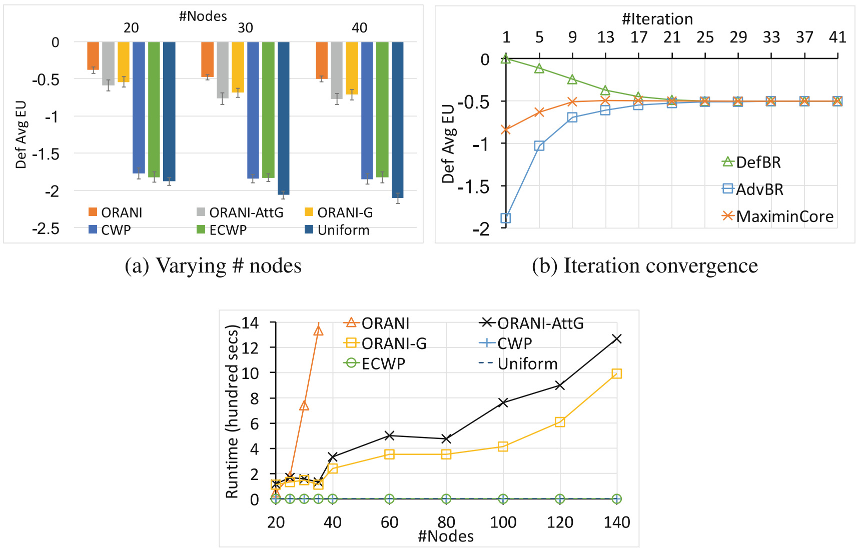
6.4.1 Synthetic Network Topology

Data Uni-Exfiltration. We compare six different algorithms: (i) **ORANI** – both exact oracles; (ii) **ORANI-AttG** – exact defender oracle and greedy attacker oracle; (iii) **ORANI-G** – both greedy oracles; (iv&v) Centrality-Weighted Placement (CWP) and Expanded Centrality-Weighted Placement (ECWP) – heuristics proposed in Sect. 5 to generate a mixed defense strategy based on the centrality values of nodes in the network; and (vi) Uniform – generating a uniformly-mixed defense strategy. We consider CWP, ECWP, and Uniform as the three baseline algorithms.

In the first experiment (Fig. 12(a)), we examine solution quality of the algorithms with varying number of nodes. In Fig. 12(a), the x-axis is the number of nodes. The y-axis is the averaged expected utility of the defender obtained by the evaluated algorithms. The data value associated with each mission-critical node is generated uniformly at random within $[0, 1]$. Intuitively, the higher averaged expected utility an algorithm gets, the better the solution quality of the algorithm is. Figure 12(a) shows that all of our algorithms, **ORANI**, **ORANI-AttG**, and **ORANI-G**, defeat the baseline algorithms in obtaining a much higher utility for the defender.

In our second experiment (Fig. 12(b)), we examine the convergence of the double oracle used in **ORANI**. The x-axis is the number of iterations of adding new strategies for both players until convergence. The y-axis is the average of

[2] A free Java graph library available at http://jgrapht.sourceforge.net.



(a) Varying # nodes (b) Iteration convergence

(c) Runtime performance

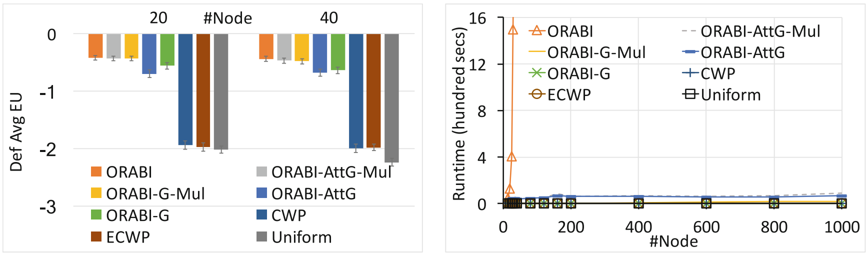
Fig. 12. Uni-exfiltration: random scale-free graphs

the defender's expected utility at each iteration with respect to the defender oracle, the attacker oracle, and the Maximin core. The number of nodes in the graph is set to 40. Figure 12(b) shows that **ORANI** converges quickly, i.e., after approximately 25 iterations. This result implies that there is only a small set of pure strategies involved in the equilibrium despite an exponential number of strategies in total. In addition, **ORANI** can find this set of pure strategies after a small number of iterations.

In our third experiment (Fig. 12(c)), we investigate runtime performance. In Fig. 12(c), the x-axis is the number of nodes in the graphs and the y-axis is the runtime on average in hundreds of seconds. As expected, the runtime of **ORANI** grows exponentially when $|V|$ increases. In addition, by using the greedy heuristics, **ORANI-AttG** and **ORANI-G** run significantly faster than **ORANI**. For example, **ORANI** reaches 1333 s on average when $|V| = 35$ while **ORANI-AttG** and **ORANI-G** reach 1266 and 990 s respectively when $|V| = 140$.

Data Broad-Exfiltration. In the case of data broad-exfiltration, we compare eight algorithms: (i) **ORABI** – both exact oracles; (ii) **ORABI-AttG** – exact defender oracle and greedy attacker oracle; (iii) **ORABI-G** – both greedy oracles; (iv) **ORABI-AttG-Mul** – exact defender oracle and greedy-multi attacker oracle; (v) **ORABI-G-Mul** – both greedy-multi oracles; and (vi, (vii, (viii) CWP, ECWP, and Uniform.

Our experimental result on solution quality is shown in Fig. 13(a). Figure 13(a) shows that all of our five evaluated algorithms obtain a much higher averaged expected utility for the defender compared to the baseline algorithms.



(a) Solution quality (b) Runtime performance

Fig. 13. Broad-exfiltration: random scale-free graphs

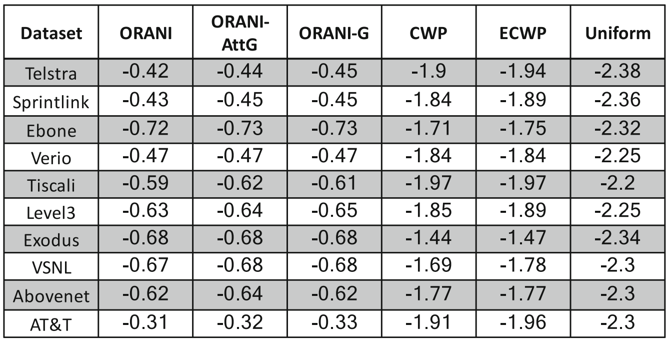
By adding multple new strategies at each iteration, both **ORABI-AttG-Mul** and **ORABI-G-Mul** perform approximately as well as **ORABI** while outperforming **ORABI-AttG**, and **ORABI-G**.

In addition, Fig. 13(b) shows that our algorithms with greedy heuristics can scale up to large graphs. For example, when $|V| = 1000$, the runtime of **ORABI-AttG-Mul**, **ORABI-G-Mul**, **ORABI-AttG**, and **ORABI-G** reaches 89, 20, 71, and 2 s respectively. We conclude that **ORABI** is the best algorithm for small graphs while **ORABI-AttG-Mul** and **ORABI-G-Mul** are proper choices for large-scale graphs.

Finally, we investigate the benefit to the attacker from broad-exfiltration compared to uni-exfiltration. We run **ORANI** and **ORABI** on the same set of 50 scale-free graph samples generated by uniformly at random with 20, 30, 40 nodes in each graph respectively. Among all the samples, there are only 58%, 72%, and 52% of the 20-node, 30-node, and 40-node graphs respectively for which the attacker obtains a strictly higher utility by using broad-exfiltration. This result shows that the attacker does not always benefit from broad-exfiltration. Despite broad-exfiltration, the data exchange between any pairs of compromised nodes must follow fixed routing paths specified by the network system, thus constraining the data exfiltration possibilities.

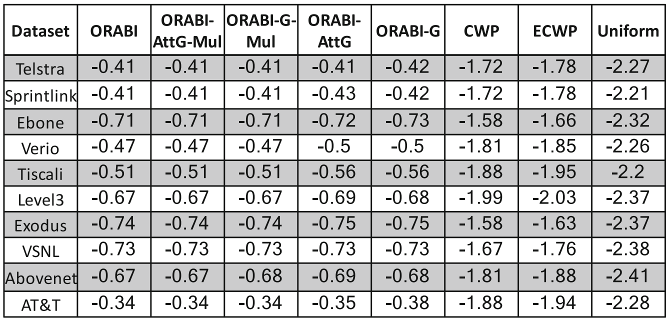
6.4.2 Real-World Network Topology

Our third set of experiments is conducted on real-world network topologies from the Rocketfuel dataset [37]. Overall, the dataset provides router-level topologies of 10 different ISP networks: Telstra, Sprintlink, Ebone, Verio, Tiscali, Level3, Exodus, VSNL, Abovenet, and AT&T. In this set of experiments, we mainly focus on evaluating the solution quality of our algorithms compared with the three baseline algorithms. For each of our experiments, we randomly sample fifty 40-node sub-graphs from every network topology using random walk. In addition, we assume that all external routers located outside the ISP can potentially route data to the attacker's server. Each data point in our experimental results is averaged over 50 different graph samples. The defender's averaged expected utility obtained by the evaluated algorithms is shown in Figs. 14 and 15.



Dataset	ORANI	ORANI-AttG	ORANI-G	CWP	ECWP	Uniform
Telstra	-0.42	-0.44	-0.45	-1.9	-1.94	-2.38
Sprintlink	-0.43	-0.45	-0.45	-1.84	-1.89	-2.36
Ebone	-0.72	-0.73	-0.73	-1.71	-1.75	-2.32
Verio	-0.47	-0.47	-0.47	-1.84	-1.84	-2.25
Tiscali	-0.59	-0.62	-0.61	-1.97	-1.97	-2.2
Level3	-0.63	-0.64	-0.65	-1.85	-1.89	-2.25
Exodus	-0.68	-0.68	-0.68	-1.44	-1.47	-2.34
VSNL	-0.67	-0.68	-0.68	-1.69	-1.78	-2.3
Abovenet	-0.62	-0.64	-0.62	-1.77	-1.77	-2.3
AT&T	-0.31	-0.32	-0.33	-1.91	-1.96	-2.3

Fig. 14. Uni-exfiltration: defender's average utility



Dataset	ORABI	ORABI-AttG-Mul	ORABI-G-Mul	ORABI-AttG	ORABI-G	CWP	ECWP	Uniform
Telstra	-0.41	-0.41	-0.41	-0.41	-0.42	-1.72	-1.78	-2.27
Sprintlink	-0.41	-0.41	-0.41	-0.43	-0.42	-1.72	-1.78	-2.21
Ebone	-0.71	-0.71	-0.71	-0.72	-0.73	-1.58	-1.66	-2.32
Verio	-0.47	-0.47	-0.47	-0.5	-0.5	-1.81	-1.85	-2.26
Tiscali	-0.51	-0.51	-0.51	-0.56	-0.56	-1.88	-1.95	-2.2
Level3	-0.67	-0.67	-0.67	-0.69	-0.68	-1.99	-2.03	-2.37
Exodus	-0.74	-0.74	-0.74	-0.75	-0.75	-1.58	-1.63	-2.37
VSNL	-0.73	-0.73	-0.73	-0.73	-0.73	-1.67	-1.76	-2.38
Abovenet	-0.67	-0.67	-0.68	-0.69	-0.68	-1.81	-1.88	-2.41
AT&T	-0.34	-0.34	-0.34	-0.35	-0.38	-1.88	-1.94	-2.28

Fig. 15. Broad-exfiltration: defender's average utility

Figures 14 and 15 show that all of our algorithms obtain higher defender expected utility than the three baseline algorithms. Further, the greedy algorithms – **ORANI-AttG**, **ORANI-G**, and **ORABI-AttG**, **ORABI-G** – are shown to consistently perform well on all the ISP network topologies compared to the optimal ones – **ORANI** and **ORABI** respectively. In particular, the average expected defender utility obtained by **ORANI-G** is only ≈ 3% lower than **ORANI** on average over the 10 network topologies.

7 Bot Identification

To identify and remove bots, we have developed a novel network-based detection scheme, called DeBot, which can identify traffic flows potentially associated with data exfiltration attempts. The proposed solution intercepts traffic from different monitoring points and leverages differences in the network behavior of botnets and benign users to identify suspicious flows. After deploying a number of detectors or monitors as described earlier, we analyze flow characteristics to identify suspicious hosts and use periodogram analysis to identify malicious flows. The fundamental assumption behind the use of periodogram analysis is that exfiltration traffic tends to be relatively more periodic than normal or benign traffic. This approach has been evaluated against different architecturally stealthy botnets in the CyberVAN testbed [7] – developed and maintained by Vencore Labs – and its performance has been compared to two state-of-the-art detection techniques, which we refer to as Stealthy P2P Detector and BSampling. The results indicate that DeBot is effective in detecting botnet activity and mapping out the botnet's architecture, and it outperforms existing solutions with respect to false positive rates. As shown in Fig. 16, the proposed approach to detect exfiltration by stealthy botnets can be divided into four phases: *preprocessing*, *observation*, *refinement*, and *analysis*.

In the *preprocessing phase*, we compute the rate at which traffic snapshots should be captured at different monitoring points within the network. We will refer to this rate as the *snapshot rate*. In DeBot, the monitoring period T (e.g., 24 h) is divided into smaller epochs, Δt (e.g., 30 mins). At each epoch in the *observation phase*, the detection mechanism randomly chooses a monitoring point based on the snapshot rates and inspects traffic traversing it during that epoch. During the monitoring period, DeBot maintains a score for each host, which is updated based on the similarity of the host's traffic pattern with other hosts within its neighborhood. At the end of the observation phase, DeBot aggregates the scores for each host based on $\frac{T}{\Delta t}$ traffic snapshots captured from different vantage points. The aggregated score is then used to identify suspicious hosts H_B by comparing the score of each host with the scores of other hosts within its neighborhood.

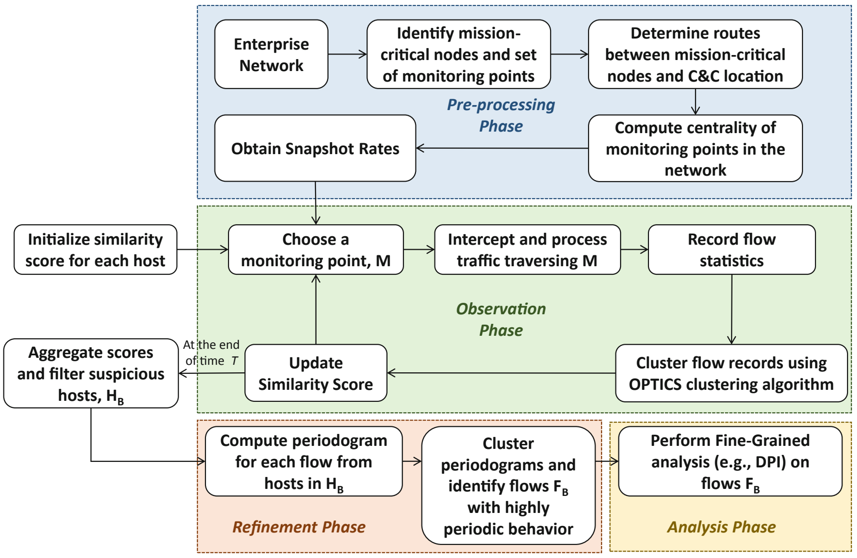

Enterprise Network
Identify mission-critical nodes and set of monitoring points
Determine routes between mission-critical nodes and C&C location
Pre-processing Phase
Obtain Snapshot Rates
Compute centrality of monitoring points in the network
Initialize similarity score for each host
Choose a monitoring point, M
Intercept and process traffic traversing M
Record flow statistics
Observation Phase
Aggregate scores and filter suspicious hosts, H_B
At the end of time T
Update Similarity Score
Cluster flow records using OPTICS clustering algorithm
Compute periodogram for each flow from hosts in H_B
Cluster periodograms and identify flows F_B with highly periodic behavior
Perform Fine-Grained analysis (e.g., DPI) on flows F_B
Refinement Phase
Analysis Phase

Fig. 16. Overview of DeBot

In the *refinement phase*, DeBot identifies flows corresponding to bot traffic by exploiting the periodic communication pattern between bots. For each host in H_B, it uses periodogram analysis to identify flows that are relatively more periodic than other flows and marks them as suspicious. Then, in the *analysis phase*, suspicious flows are further analyzed using fine-grained analysis techniques such as Deep Packet Inspection.

7.1 Preprocessing Phase

The objective of an exfiltration campaign is to periodically transfer sensitive data to a remote attacker-controlled server. Typically, in an enterprise network, sensitive data is confined to a few servers which we refer to as *mission-critical* servers. Exfiltrated data traverses several intermediate forwarding devices, such as switches, routers and gateways, before reaching the remote server. Any of these internal devices can be used as a monitoring point. In the proposed detection scheme, traffic is mirrored from these devices to a central location for analysis. In large enterprises, a mirroring mechanism is is usually already in place to remotely monitor performance.

Given the sparseness of malicious flows, it is critical to identify monitoring points that are likely to capture such flows. For instance, consider the enterprise network in Fig. 17, with the sensitive data stored in the file servers in Subnet-1 and Subnet-2. The file servers host redundant copies of the data. To exfiltrate this data, an attacker may choose one of several botnet communication architectures with varying degrees of exposure of malicious flows to detectors. For example, an attacker could compromise one of the clients in Subnet-1, say h_1 with IP 192.168.1.2, mount the share drive, and directly transfer sensitive data to C&C.

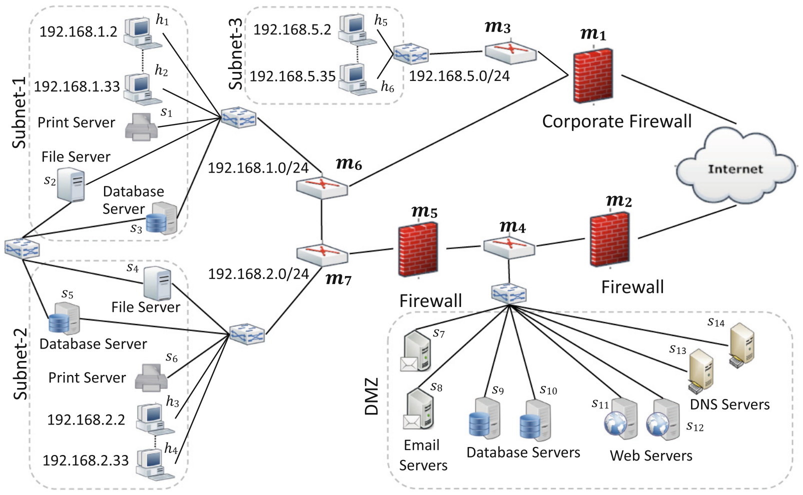


Fig. 17. Enterprise network example

Due to internal routing policies, the traffic traverses two intermediate devices – router m_6 and firewall m_1 – before exiting the network. For the purpose of presentation, we denote this communication architecture as a path, $h_1 \rightarrow m_6 \rightarrow m_1 \rightarrow$ C&C. Other possible exfiltration paths include, but not limited to: $h_3 \rightarrow m_7 \rightarrow m_6 \rightarrow m_1 \rightarrow$ C&C, where h_3 is a compromised host in Subnet-2, $h_1 \rightarrow m_6 \rightarrow m_7 \rightarrow m_5 \rightarrow m_4 \rightarrow s_9 \rightarrow m_4 \rightarrow m_2 \rightarrow$ C&C, in which a database server s_9 was compromised and used as a relay. It can be observed that the percentage of malicious flows intercepted by different monitoring points depends on the internal routing policy and the attacker's choice of communication architecture.

Table 2. Number of flows vs. number of bot flows

Monitoring point	No. of unique flows	No. of unique bot flows	% of bot flows
m_1	5,248	0	0
m_2	149,392	3,451	2.31
m_3	106,448	1,724	1.62
m_4	913,680	7,766	0.85
m_5	690,748	4,352	0.63
m_6	126,156	1,728	1.37
m_7	149,580	3,455	2.31
Total	1,146,784	11,124	0.97

Detecting malicious traffic within large networks calls for a scalable detection mechanism. Although capturing traffic from all monitoring points would ensure that all malicious flows are intercepted, such approach is not scalable. Therefore, it is crucial to identify the most effective set of monitoring points so as to limit the amount of data to be analyzed, whilst ensuring that a sufficient number of malicious flows are intercepted for the detection mechanism to be able to distinguish malicious flows from benign flows.

To understand the impact of different monitoring points on processing time and accuracy of a detection mechanism, we simulated the network scenario of Fig. 17 in the CyberVAN testbed [7], which can generate benign user traffic. In the network of Fig. 17, we consider a stealthy botnet with a communication architecture composed of a server in the DMZ and four compromised hosts, two in Subnet-1 and two in Subnet-2. The bots exfiltrate data from the file server and forward it to the server in the DMZ, which aggregates data and relays it to C&C. Table 2 shows the number of flows intercepted at different monitoring points during a 12-h monitoring period: when m_4 is chosen as a monitoring point, the detection mechanism processes 6 times more records than m_7, while intercepting only twice as many bot flows as m_7. This example shows that the relationship between the volume of traffic monitored and the number of malicious flows intercepted is not linear.

To improve the likelihood of intercepting malicious flows in resource-constrained environments, Venkatesan *et al.* [41] proposed a dynamic monitoring strategy that exploits graph-theoretic properties of the network, which is modeled as a graph $G(V, E)$, with a subset of nodes $M_c \subseteq V$ identified as mission-critical. For each potential monitoring point $m \in M \subset V$, they compute a new centrality measure, known as the *mission-betweenness centrality* $C_M(m)$, which is a function of the fraction of shortest paths between mission-critical nodes and C&C that traverse m. The time horizon is divided into smaller observation epochs and in each epoch a monitoring point m is chosen with probability $\frac{C_M(m)}{\sum_{m' \in M} C_M(m')}$. As the above strategy only considered monitoring points on shortest paths, to improve coverage the authors proposed the *expanded centrality-weighted strategy*, which also considers monitoring points on paths that are δ times longer than the shortest paths.

In this work, we adopt the principle behind the dynamic strategy mentioned above – i.e., choosing monitoring points with high centrality – and modify it to account for internal routing policies. In [41], the authors assumed that traffic between systems is routed through the shortest path. However, an enterprise network is segmented into subnets and the route between any two systems depends on the routing policies at different monitoring points. Such policies are influenced by several factors such as network load and security policies. We use the tracert tool to identify the routes traversed by traffic between two systems s and t and in turn a set of monitoring points that can intercept that traffic. We use $\mathscr{R}$ to denote the set of all routes between systems in a network.

As mentioned earlier, stealthy botnets reduce exposure to detectors by compromising additional systems and using them as proxies to relay traffic to C&C. In an enterprise network, most communication patterns follow a client-server model. Thus, to avoid suspicious patterns, compromised servers could

Algorithm 3. $computeSnapshotRates(M, M_c, S, \mathscr{R})$

Input: a set M of potential monitoring points, a set M_c of mission-critical nodes, a set S of potential proxy servers, and a set R of routes between pairs of nodes in M
Output: the snapshot rate $P(m)$ for each monitoring point $m \in M$

```
 1: for all $m_c \in M_c$ do
 2:     $\mathscr{R}'_{m_c} \leftarrow \emptyset$
 3:     for all $s \in S$ do
 4:         $\mathscr{R}'_{m_c} \leftarrow \mathscr{R}'_{m_c} \cup \{R_1 || R_2 \mid (R_1, R_2) \in \mathscr{R}_{m,s} \times \mathscr{R}_{s,\text{C\&C}}\}$
 5:     end for
 6: end for
 7: $C_B(m) \leftarrow 0, \forall m \in M$ // Initialize mission-betweenness centrality
 8: for all $m_c$ in $M_c$ do
 9:     $\sigma(m) \leftarrow 0, \forall m \in M$
10:     for all $R \in \mathscr{R}'_{m_c}$ do
11:         for all $m \in M \cap R$ do
12:             $\sigma(m) \leftarrow \sigma(m) + 1$
13:         end for
14:     end for
15:     $C_B(m) \leftarrow C_B(m) + \frac{\sigma(m)}{|\mathscr{R}'_{m_c}|}, \forall m \in M$
16: end for
17: $P(m) \leftarrow \frac{C_B(m)}{\sum_{m' \in M} C_B(m')}, \forall m \in M$
18: return $P$
```

take the role of a proxies for bots within the network. However, we make a conservative assumption that all systems (both clients and servers) can act as proxies for the botnet. This assumption creates a worst-case scenario for the defender, thereby making it challenging to design a detection mechanism. To collect large traffic samples of such botnets, we first compute the centrality of all the monitoring points, similarly to the expanded centrality-weighted strategy in [41].

We use algorithm *computeSnapshotRates* (Algorithm 3) to compute the snapshot rates. For each mission-critical node m_c and potential proxy s, the algorithm first determines all the paths through s by concatenating routes from m to s, $\mathscr{R}_{m,s}$, with routes from s to C&C. Here, we conservatively assume that any destination outside the network is a potential C&C server. The resulting set of routes $\mathscr{R}'_m$ is used to compute the mission-betweenness centrality of each monitoring point (lines 7–16). Finally, the snapshot rate for each monitoring point is computed on line 17. Assuming that the topology of the network remains static during the entire monitoring period, this is a one-time computation. The snapshot rate of a monitoring point $m \in \mathscr{M}$ is the probability that m is chosen by DeBot for analyzing traffic traversing it during an epoch. Randomness introduces uncertainty for the attacker and increases the cost and complexity of establishing a stealthy botnet architecture.

7.2 Observation Phase

DeBot identifies suspicious hosts by comparing their network characteristics with other hosts within their neighborhood. The neighborhood of a host is the set of hosts that are expected to exhibit similar network characteristics in the absence of malicious activity. Hosts whose characteristics deviate from their neighboring hosts are classified as suspicious. In this chapter, without loss of generality, we assume that hosts in the same subnet exhibit similar behavior, thus a host's neighborhood is represented by its subnet. Prior to starting this phase, DeBot initializes the similarity scores of host pairs, which quantify the similarity in the network behavior of any two hosts. At the beginning of each observation epoch $\Delta t_i, i \in \left[1, \frac{T}{\Delta t}\right]$, DeBot selects a monitoring point m_i based on the snapshot rates computed in the preprocessing phase. Traffic traversing m_i during Δt_i is intercepted and statistics of each flow are recorded. In this work, a flow is uniquely identified by the tuple (src, dst, sport, dport, protocol). Flow statistics are used as features to cluster flows, and subsequently update the similarity scores of host pairs based on the number of common clusters between them. Finally, at the end of the time horizon, DeBot identifies suspicious hosts by comparing the aggregate behavior of each host with other hosts within its subnet.

For each flow f, DeBot records the median number of packets sent ($pkts_{sent}(f)$) and received ($pkts_{recv}(f)$), and the median number of bytes sent ($bytes_{sent}(f)$) and received ($bytes_{recv}(f)$) during an epoch Δt_i. We refer to the tuple $\langle pkts_{sent}(f),\ pkts_{recv}(f),\ bytes_{sent}(f),\ bytes_{recv}(f)\rangle$ as the statistics of flow f. A TCP session is considered a flow when a SYN packet is acknowledged by a SYN-ACK packet. However, as the traffic snapshot may include incomplete sessions, a TCP session is included in the flow record table $\mathscr{F}_i$ if at least one packet and its acknowledgment are intercepted during the same epoch. For UDP packets, only flows in which a request is followed by a response are considered.

To identify flows that exhibit similar network behavior, the flow records in $\mathscr{F}_i$ are clustered using the OPTICS clustering algorithm [4], a density-based clustering algorithm that, unlike K-means, can identify arbitrarily shaped clusters by grouping closely-spaced records. OPTICS uses a priority queue to linearly order the input records so that records that are closely-spaced are placed together. In OPTICS, a group of records is identified as a cluster if two conditions hold: (i) it includes at least *minPts* records; and (ii) for any two records in the cluster, there is a sequence of records within the cluster such that every pair of consecutive records is within a distance ϵ. Records that do not belong to any cluster are labeled as noise.

As DeBot operates on traffic snapshots, selecting an optimal value for *minPts* is crucial to ensure that intercepted bot flows form a cluster and are not treated as noise. If *minPts* is too high, the traffic snapshot might not have intercepted a sufficient number of bot flows to form a cluster, whereas, if it is too low, it will lead to creation of multiple clusters. Therefore, the choice of *minPts* is influenced by both the frequency ν of bot communication and the length Δt of the observation window. The relationship between the three variables can be approximated as $minPts = \kappa \cdot \frac{\Delta t}{\nu}$ where $0 < \kappa < 1$ is a constant. In order to limit the number of meaningful clusters, *minPts* is fixed and the length of an observation epoch is expressed as a function of ν, i.e., $\Delta t = \frac{minPts}{\kappa} \cdot \nu$. In order words, the choice of Δt bounds the frequency with which bots can send/receive update messages without losing stealth.

As mentioned earlier, DeBot tracks the similarity in network behavior between hosts. Let $\mathscr{N}(h)$ denote the set of hosts in the neighborhood of host h, and let a scoring function $sim(h_i, h_j)$ quantify the similarity between two hosts h_i and h_j. Before the observation phase, the scoring function is initialized as $sim(h_i, h_j) = 1, \forall h_j \in \mathscr{N}(h_i)$. As noted earlier, in our work all hosts that are in the same subnet as host h are considered its neighbors. Let $\mathscr{C}_k$ denote the set of flow clusters at the end of an observation epoch Δt_k, and let $C_{h_i} \subseteq \mathscr{C}_k$ be the subset of clusters containing flows from/to host h_i. The scoring function is updated as follows:

$$sim(h_i, h_j) = \lambda(m_k, h_i) \cdot \left(\frac{|C_{h_i} \cap C_{h_j}|}{|C_{h_i} \cup C_{h_j}|} \right) + (1 - \lambda(m_k, h_i)) \cdot sim(h_i, h_j) \quad (11)$$

where $\lambda(m_k, h_i)$ is a scalar-valued function that models the rate at which the similarity score is updated. To define this function, we first define the *visibility* of a monitoring point m as the set of hosts whose incoming and outgoing traffic traverses m. For instance, in the network of Fig. 17, the visibility of M_3 is the set of hosts in the subnet 192.168.5.0/24. In DeBot, if a host is not visible to the current monitoring point, then the score is updated at a slower rate. In particular, the $\lambda()$ function is defined as:

$$\lambda(m_k, h_i) = \begin{cases} 0.5 & \text{if } h_i \in \mathit{visibility}(m_k) \\ 0.25 & \text{otherwise} \end{cases}$$

At the end of the observation phase, the aggregate network score of a host is computed as the sum of the similarity scores of the host with hosts in its neighborhood, i.e., $\mathit{agg\_score}(h_i) = \sum_{h_j \in \mathscr{N}(h_i)} \mathit{sim}(h_i, h_j)$. A high aggregate score implies that the host exhibited network characteristics similar to the hosts in its neighborhood while a low score implies that the host's network characteristics deviated from the other hosts. Based on this rationale, a host h_i is identified as suspicious if its aggregate score is less than $\mu_{agg}(\mathscr{N}(h_i)) - \sigma_{agg}(\mathscr{N}(h_i))$, where $\mu_{agg}(\mathscr{N}(h_i))$ and $\sigma_{agg}(\mathscr{N}(h_i))$ are, respectively, the mean and standard deviation of the aggregate scores of hosts in the neighborhood of h_i.

7.3 Refinement Phase

A bot participating in an exfiltration campaign regularly communicates with its peer bots or C&C to send or receive updates. Table 3 shows the observed communication frequency of different instances of POS malware. Such periodic behavior has also been observed in botnets that are known for stealing credentials, such as Storm, Waldec, and Zeus [32].

In DeBot, we leverage the periodic communication feature of bots to identify malicious host pairs. To determine whether a host h_i is periodically communicating with another host h_j, the communication pattern between h_i and h_j is treated as a signal in the time domain and transformed to the frequency domain using Discrete Fourier Transform (DFT). After the transformation, the Power Spectrum Density (PSD) of different frequencies is analyzed and compared with the PSD of other connections generated by host h_i to identify periodic communications. Details are provided in the following subsections.

Table 3. Communication frequency of malware

POS malware	Victim	Period
BlackPOS [24]	Target	10 mins
FrameworkPOS [24]	Home Depot	60 mins + random mins
Backoff [24]	UPS	45 s
Punkey [27]	CiCi's Pizza (suspected)	20 mins or 45 mins

7.3.1 Detecting Periods Using Periodogram Analysis

Let $TS_{i,j} = \{ts_1, ts_2, ...\}$ be the set of timestamps at which a connection was initiated from host h_i to h_j. The monitoring period $[0, T]$ is divided into equally-spaced time points $T_{i,j} = \{t_1, t_2, ..., t_N\}$, where $t_{k+1} - t_k = \Delta s$ and $N = \frac{T}{\Delta s}$. When traffic between two hosts h_i and h_j is continuously monitored, the corresponding connection pattern is treated as a signal that has been sampled at evenly-spaced time intervals, $X_{h_i,h_j}(t_k), \forall t_k \in T_{i,j}$, defined as:

$$X_{h_i,h_j}(t_k) = \begin{cases} 1, \exists ts_l \in TS_{i,j},\ ts_l \in (t_{k-1}, t_{k+1}) \\ 0, \text{otherwise} \end{cases}$$

A Discrete Fourier Transform (DFT) converts a signal in the time domain to the frequency domain by expressing the signal as a sum of sinusoidal components using the equation:

$$F_{h_i,h_j}(\omega) = \sum_{k=1}^{N} X_{h_i,h_j}(t_k) e^{-i\omega t_k} \tag{12}$$

where $\omega = 1, ..., N$ and $e^{i\theta} = cos(\theta) + i \cdot sin(\theta)$. Essentially, the DFT coefficient, $F_{h_i,h_j}(\omega)$, at frequency ω *correlates* the signal X_{h_i,h_j} with a sequence of sine and cosine waves at frequency ω – the higher the coefficient value, the greater the similarity. The strength of each frequency in the signal is computed by the power spectrum density. Several methods exists to estimate the power spectral density [39]. In this work, we use the periodogram method as it is computationally less expensive than other methods. The periodogram of the time series X_{h_i,h_j} is given by:

$$P_{h_i,h_j}(\omega) = \frac{1}{N} |F_{h_i,h_j}(\omega)|^2 = \frac{1}{N} \left[\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) cos\ \omega t_k \right)^2 + \left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) sin\ \omega t_k \right)^2 \right] \tag{13}$$

Figure 18 shows the periodogram of a sample of Zeus traffic obtained from a public repository [13]. In this network trace, the bot connected with its peer bot every 60 s, which, in the periodogram, is represented by the frequency corresponding to the highest power. Employing the above approach directly in DeBot, however, presents several limitations:

- *Unevenly-spaced observations*: Equation 13 assumes that the traffic being analyzed was sampled at equally-spaced time intervals. However, DeBot employs a dynamic monitoring strategy which only captures snapshots of traffic from different monitoring points. Thus, there may be long periods of unobserved connection patterns between pairs of hosts, and the above periodogram analysis may not accurately estimate the power of different frequencies in the signal.
- *Detecting periodicity*: DFT treats every discrete time series as periodic. Thus, labeling a connection pattern as periodic based on high peaks in the

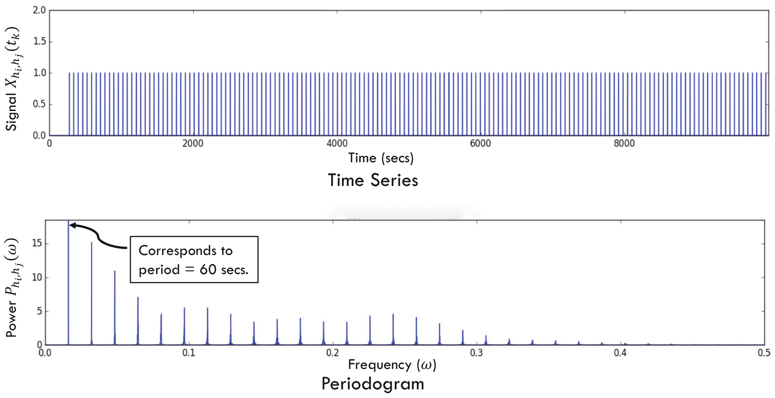


Fig. 18. Communication pattern and periodogram of a Zeus bot

periodogram will lead to a large number of false positives. Furthermore, *random fluctuations* due to noisy data and *spectral leakage* due to finite-length sampling may also produce peaks at frequencies that do not correspond to the true frequency of the signal.

In addition to addressing the above limitations, the detection mechanism should be robust in the following two scenarios, which we also address in the following subsections.

- *Random perturbations*: As malicious flows are detected based on their periodicity, bots can evade detection by introducing random perturbations to the connection pattern.
- *False positives*: Legitimate applications, such as software updates and email clients, also generate periodic flows, which may be misclassified as malicious.

7.3.2 Lomb-Scargle Periodogoram

The dynamic monitoring strategy results in sampling traffic between two hosts h_i, h_j at time points $t_k, k = 1, 2, ..N$ that are not evenly spaced. To study the periodicity of an unevenly-spaced discrete time series, we use the Lomb-Scargle periodogram to estimate the power spectrum [33]. The Lomb-Scargle periodogram modifies the classical periodogram given in Eq. 13 by introducing a time translation parameter τ:

$$P_{h_i,h_j}(\omega) = \frac{1}{2} \cdot \left[\frac{\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) cos\ \omega(t_k - \tau) \right)^2}{\sum_{k=1}^{N} cos^2 \omega(t_k - \tau)} + \frac{\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) sin\ \omega(t_k - \tau) \right)^2}{\sum_{k=1}^{N} sin^2 \omega(t_k - \tau)} \right] \tag{14}$$

where

$$\tau = (1/2\omega) tan^{-1} \left[\left(\sum_{k=1}^{N} sin(2\omega t_k) \right) \Big/ \left(\sum_{k=1}^{N} cos(2\omega t_k) \right) \right]$$

The power spectrum obtained from Eq. 14 is shown to be statistically equivalent to the least squares fit of a sinusoidal wave applied to the discrete time series [33]. High peaks in the resulting periodogram are not sufficient to conclude that the signal is periodic. Noise in a signal can also produce large spurious peaks in the periodogram. To extract the candidate periods that are due to harmonic components in the signal – and not due to noise – a threshold power-level is determined using significance tests. For a given level of confidence, a significance test models the pure noise as a Gaussian distribution $\mathscr{N}(\mu, \sigma)$ and determines a threshold power-level z_0 below which a power is considered to be pure noise [11]. Thus, if there are no frequencies whose power is greater than the threshold z_0, then the signal is considered to be non-periodic. One of the limitations of the significance test is that the threshold is sensitive to the choice of parameters μ and σ for the Gaussian distribution [11]. Furthermore, existing non-parametric methods [44] are applicable to evenly-spaced time series and, thus, cannot be directly adopted in our setting.

7.3.3 Relative-Periodicity

In this work, instead of checking whether the connection pattern from host h_i to host h_j is periodic using significance test, we determine if it is *relatively* periodic by comparing its periodogram with that of other connection patterns generated by h_i during the monitoring period. Thus, we use the system's typical network behavior (instead of white noise) as the baseline to check for periodicity.

Let $\mathscr{P}_{h_i} = \{P_{h_i,h_j}(\omega)\}$ be the set of periodograms of connection patterns originating from host h_i (obtained using Eq. 14). To determine which periodogram exhibits an anomalously higher periodicity, the periodograms in $\mathscr{P}_{h_i}$ are clustered using agglomerative clustering. While clustering, the difference in periodic structures between two periodograms is assessed using the power distance metric [44]. The power distance between P_{h_i,h_j} and P_{h_i,h_k} is computed by first identifying the set of frequencies $\omega_{i,j}$ with the K-highest powers in P_{h_i,h_j}. The power distance is then defined as:

$$pDist = ||P_{h_i,h_j}(\omega_{i,j}) - P_{h_i,h_k}(\omega_{i,j})||$$

In our evaluation, we set $K = 1000$. To ensure that the total energy is constant, before computing $pDist$, the powers are normalized as follows:

$$X(t) = \frac{X(t) - \frac{1}{N}\sum\limits_{i=1}^{N} X(i)}{\sqrt{\sum\limits_{i=1}^{N}\left(X(t) - \frac{1}{N}\sum\limits_{i=1}^{N} X(i)\right)}}, t = 1, 2, ...N$$

In the proposed hierarchical cluster analysis of periodograms, the linkage criteria between sets of periodograms was computed using the Ward's method. After building a hierarchical structure of the periodograms, clusters are formed by the set of periodograms whose pairwise distance is less than a threshold γ.

The value of γ was set to $0.95 \cdot max_d$ where max_d is the maximum distance between any two sets of periodograms as determined by the Ward's method. Finally, if a cluster contains only one periodogram, the connection pattern of the corresponding host pair is considered to be relatively periodic. The rationale behind this approach is that connection patterns corresponding to bot flows are anomalously more periodic than other connection patterns from the same hosts, thus the corresponding periodogram will form an individual cluster. The host pairs (h_i, h_j) that are identified as relatively periodic are marked as suspicious for further analysis.

Finally, in the analysis phase, flows generated by host pairs marked as suspiciously periodic can be analyzed using fine-grained tools such as Deep Packet Inspection or submitted for manual inspection to the security operations center.

8 Botnet Lifetime

Ultimately, the defender's objective is to eradicate a botnet from the network. 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 this need and solve the third challenge mentioned earlier, we propose to deploy—in a defense-in-depth approach—a mix of two classes of countermeasures, namely honeypots and network-based detectors. While honeypots are used to detect intrusion attempts—including a bot's attempt to compromise another machine—network-based detection mechanisms can identify (through behavioral analysis) bots that coexist with benign machines. Both honeypots and network-based detectors can be treated as resources available to the defender in limited supply due to cost constraints.

8.1 Reinforcement Learning Model

To optimally and dynamically deploy these mechanisms in an iterative fashion, and consequently reduce the lifetime of botnets, we developed a solution based on a reinforcement learning model [43]. Reinforcement learning (RL) 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 [14]. Formally, the agent seeks to discover a policy that maps the system state to an optimal action. In our work, the agent learns a policy that maximizes the total number of bots detected and removed over time. As the location of bots is unknown prior to the deployment of defense mechanisms, the agent estimates the short-term and long-term rewards of an action by monitoring network activity within different network segments. In particular, the agent monitors the behavior of hosts with respect to potential attack indicators (e.g., scanning activity and number of outgoing sessions) to inform the next iteration of detector placement. This approach was compared to three other strategies, namely:

- a static strategy, which does not modify detector placement over time;
- an MTD centrality-based strategy, which periodically alters detector placement based on topology-driven centrality measures;
- a myopic strategy, which makes placement decisions to optimize short-term benefits based on feedback from the operating environment.

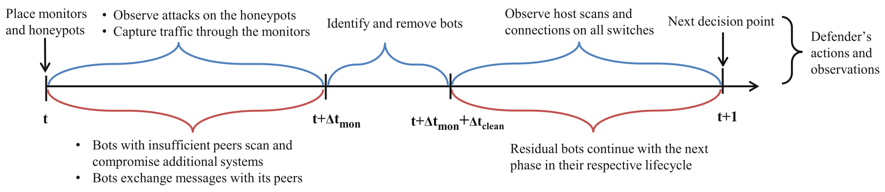

Place monitors and honeypots
• Observe attacks on the honeypots
• Capture traffic through the monitors
Identify and remove bots
Observe host scans and connections on all switches
Next decision point
Defender's actions and observations
t
t+Δt_mon
t+Δt_mon+Δt_clean
t+1
• Bots with insufficient peers scan and compromise additional systems
• Bots exchange messages with its peers
Residual bots continue with the next phase in their respective lifecycle

Fig. 19. Timeline of defender's and attacker's actions and observations

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 Fig. 19. 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 Sect. 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 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[3], 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[4] 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 [47]. 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 Fig. 19. 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

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

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:

$$\psi_i^f(t) = \begin{cases} HIGH, \text{ if } Total_i^f(t) \geq \mu_i^f + \sigma_i^f \\ MED, \text{ if } Total_i^f(t) \in (\mu_i^f - \sigma_i^f, \mu_i^f + \sigma_i^f) \\ LOW, \text{ if } Total_i^f(t) \leq \mu_i^f - \sigma_i^f \end{cases} \tag{15}$$

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. 15 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 [12,29,31].

We estimate the number of bots in a network segment by determining the number of hosts that have deviated from the expected 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 $\mu_{mc,i}^f$ and $\sigma_{mc,i}^f$ 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 $Total_{mc,i}^f(t)$ 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 $(\exists i \in [1, N])(Total_{mc,i}^f(t) \geq \mu_{mc,i}^f + 3 \cdot \sigma_{mc,i}^f)$. 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_t^x, 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. 15, 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 co-existence 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 [6] below (Eq. 16). 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.

8.2 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:

$$V^j(\hat{S}^x_{t-1}) = (1 - \alpha^j)V^j(\hat{S}^x_{t-1}) + \alpha^j \nu^j \tag{16}$$

$$\nu^j = \left[\max_{x_t \in \mathscr{X}} \left\{ R(S_t, x_t) + \beta V^j(\hat{S}^x_t) \right\} \right] \tag{17}$$

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, $\mathscr{X}$ 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. 17) 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 Eqs. 16 and 17. 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.

A snapshot of the state-transition diagram is shown in Fig. 20, 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 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

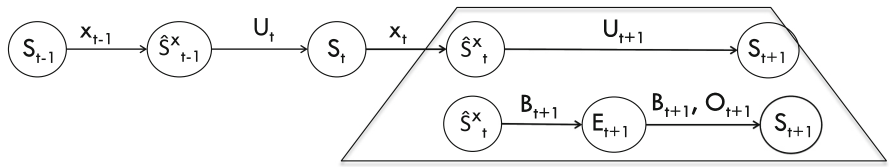


Fig. 20. State transition diagram

Algorithm 4. Exploration and Learning Phase

```
Input: Baseline values $\psi_i^f$ of each feature $f \in \{h, s\}$ for each monitoring point $i$, baseline values
    $\psi_{mc,i}^f$ for each machine $mc$, decision space $\mathscr{X}$, 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 $j = \{1, ..., J\}$ 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 $t = \{1, ..., 365\}$ 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 8.1, $\forall x_t \in \mathscr{X}$
14:             Choose the action $x_t'$ that gives the maximum value in Eq. 17
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 [14]
19:             Update value of post-decision state $V^j(\hat{S}_{t-1}^x)$ using Eq. 16 and Eq. 17 with the real
                reward and the value of $\alpha^j$
20:         end if
21:     end for
22: end for
```

Fig. 20), 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 4, Eq. 16 is used without the max operator in Eq. 17 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. 17 with the max operator (lines $13 - 14$ of Algorithm 4). The value of the post-decision state at time $t - 1$ is updated at time $t+1$ as per Eq. 16 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 [31].

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. 17 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.

8.3 Simulation Results

Differently from the myopic strategy, the reinforcement learning strategy also considers long-term benefits of possible detector placements. Figure 21 shows how the number of bots in the network changes over time for each of the four strategies considered. If the number of bots reaches 0, then the botnet has been completely removed from the system. As expected, the static placement strategy exhibits the worst performance: once a botnet is established, bots that are not observable by the static detectors can persist in the network indefinitely. MTD strategies, on the other hand, provide significantly better protection as they introduce uncertainty about the location of detectors. The myopic strategy shows

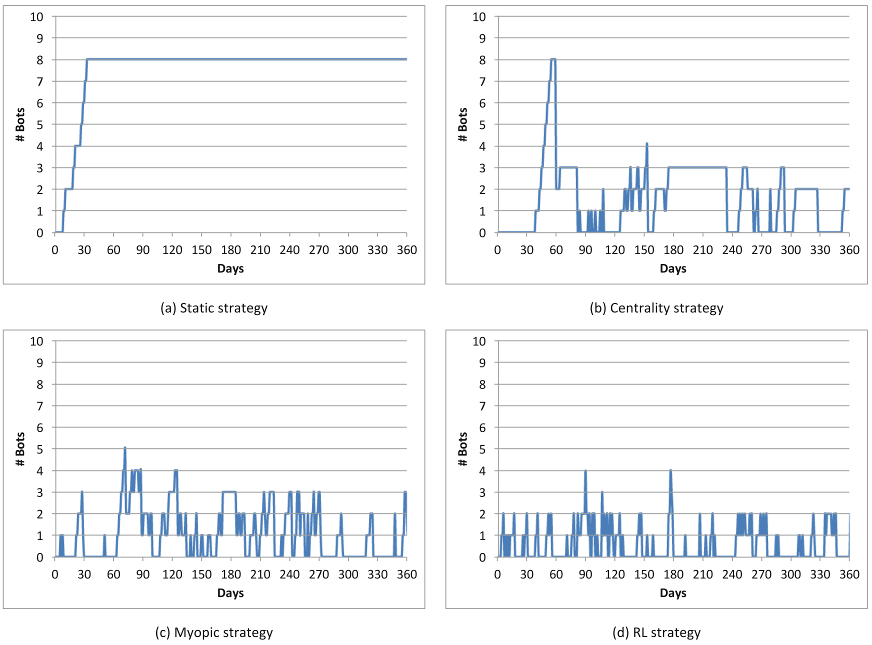


Fig. 21. Comparison between the reinforcement learning strategy and other strategies

a significant improvement over the centrality strategy in reducing the lifetime of the botnet, because it also considers information obtained from the network. Finally, among the MTD strategies, the RL approach shows the largest reduction in the botnet's lifetime.

9 Conclusions and Future Work

Stealthy botnets pose significant threats due to their ability to evade traditional defenses and persist in the target system for extended periods of time. Detecting and mitigating these threats is a multifaceted problem that calls for novel solutions to address the several interrelated challenges that stealthy bots introduce. We have shown how the problem can be decomposed into relatively simpler problems that can be tackled separately, while keeping the big picture in mind. Essentially, we need to understand where and when to monitor for potentially suspicious activity, how to look at observed traffic to identify potentially compromised machines, and how to ensure that each and every bot has been removed from the network. Moving Target Defense (MTD) has proved to be a viable and promising approach in tackling these challenges, enabling us to achieve some interesting results. Of course, the security benefits of deploying MTD techniques come at a cost for the defender, which can be measured in terms of increased overhead to maintain availability for legitimate users. The tradeoff between security and cost can generally be controlled by configuring the parameters of an MTD technique.

In this chapter, we have presented the key findings of our work on disrupting stealthy botnets through the use of a novel moving target defense approach. Specifically, we have targeted botnets that are being used for exfiltrating sensitive data from mission-critical systems. Defending against such botnets is challenging, as research has shown how they have become increasingly sophisticated and have the capability of operating in stealth mode by minimizing their footprint.

In order to defeat exfiltration attempts by modern botnets, we have proposed a moving target defense approach for placing detectors across the network – in a resource-constrained environment – and dynamically and continuously changing the placement of detectors over time. Specifically, we have proposed several strategies based on centrality measures that capture important properties of the network. Our objective is to increase the attacker's effort and likelihood of detection by creating uncertainty about the location of detectors and forcing the botmaster to perform additional actions in an attempt to create detector-free paths through the network.

We have presented two metrics to evaluate the proposed strategies – namely the *minimum detection probability* and the *attacker's uncertainty* – and an algorithm to compute the minimum detection probability. We validated our approach through simulations, and the results confirmed that the proposed solution can effectively reduce the likelihood of successful exfiltration campaigns.

As part of our work on optimal detector placement, we also proposed a Stackelberg game model that accounts for the strategic response of attackers to deployed defenses. We proposed two double-oracle based algorithms, **ORANI** and **ORABI**, to compute optimal defense strategies with respect to data uni-exfiltration and broad-exfiltration formulations, respectively. We also provided greedy heuristics to approximate the defender and the attacker best-response oracles. We conducted experiments based on both random scale-free graphs and real-world ISP network topologies, demonstrating the advantages of our game-theoretic solution over previous strategies.

To identify and remove bots, we have developed a novel network-based detection scheme, called DeBot, which can identify traffic flows potentially associated with data exfiltration attempts. The proposed solution intercepts traffic through deployed detectors and leverages differences in the network behavior of botnets and benign users to identify suspicious flows. We analyze the characteristics of traffic flows to identify suspicious hosts and use periodogram analysis to identify malicious flows. The fundamental assumption behind the use of periodogram analysis is that exfiltration traffic tends to be relatively more periodic than normal or benign traffic. This approach has been evaluated against different architecturally stealthy botnets in the CyberVAN testbed and its performance has been compared to two state-of-the-art detection techniques. The results indicate that DeBot is effective in detecting botnet activity and outperforms existing solutions with respect to false positive rates.

Finally, to achieve the defender's ultimate objective of eradicating a botnet from the network, we proposed to deploy – in a defense-in-depth approach – a mix of two classes of countermeasures, namely honeypots and network-based detectors. While honeypots are used to detect intrusion attempts – including a bot's attempt to compromise another machine – network-based detection mechanisms can identify (through behavioral analysis) bots that coexist with benign machines. Both honeypots and network-based detectors can be treated as resources available to the defender in limited supply due to cost constraints. To optimally and dynamically deploy these mechanisms in an iterative fashion, and consequently reduce the lifetime of botnets, we developed a solution based on a reinforcement learning model.

Our future plans include but are not limited to: (i) introducing a probabilistic model to account for false negatives in the deployed detectors; (ii) defining and evaluating the performance of the proposed detector placement strategies against more sophisticated attacker's strategies; and (iii) casting the model in a game-theoretic framework to study the Nash equilibria and dominant strategies.

References

1. APT1: Exposing one of China's cyber espionage units. Technical report, Mandiant, February 2013
2. Lateral movement: how do threat actors move deeper into your network? Technical report, Trend Micro (2013)
3. Alpcan, T., Başar, T.: An intrusion detection game with limited observations. In: Proceedings of the 12th International Symposium on Dynamic Games and Applications (ISDG 2006), Sophia-Antipolis, France, July 2006

4. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: OPTICS: ordering points to identify the clustering structure. In: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999), pp. 49–60. ACM, Philadelphia, May 1999
5. Beigi, E.B., Jazi, H.H., Stakhanova, N., Ghorbani, A.A.: Towards effective feature selection in machine learning-based botnet detection approaches. In: Proceedings of the IEEE Conference on Communications and Network Security (IEEE CNS 2014), pp. 247–255. IEEE, San Francisco, October 2014
6. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
7. Chadha, R., et al.: CyberVAN: a cyber security virtual assured network testbed. In: Proceedings of the 2016 IEEE Military Communications Conference (MILCOM 2016), pp. 1125–1130. IEEE, Baltimore, November 2016
8. Collins, M.P., Shimeall, T.J., Faber, S., Janies, J., Weaver, R., Shon, M.D., Kadane, J.B.: Using uncleanliness to predict future botnet addresses. In: Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC 2007), pp. 93–104. ACM, San Diego, October 2007
9. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the Internet topology. ACM SIGCOMM Comput. Commun. Rev. **29**(4), 251–262 (1999)
10. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM (JACM) **34**(3), 596–615 (1987)
11. Frescura, F.A.M., Engelbrecht, C.A., Frank, B.S.: Significance tests for periodogram peaks, June 2007. https://arxiv.org/abs/0706.2225
12. Ganesan, R., Jajodia, S., Shah, A., Cam, H.: Dynamic scheduling of cybersecurity analysts for minimizing risk using reinforcement learning. ACM Trans. Intell. Syst. Technol. **8**(1) (2016)
13. García, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Comput. Secur. **45**, 100–123 (2014)
14. Gosavi, A.: Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, Operations Research/Computer Science Interfaces, vol. 55, 2nd edn. Springer, New York (2003)
15. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th USENIX Security Symposium (USENIX Security 2008), pp. 139–154. USENIX Association, San Jose, July 2008
16. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: detecting malware infection through IDS-driven dialog correlation. In: Proceedings of the 16th USENIX Security Symposium (USENIX Security 2007), pp. 167–182. USENIX Association, August 2007
17. Jain, M., Korzhyk, D., Vaněk, O., Conitzer, V., Pěchouček, M., Tambe, M.: A double oracle algorithm for zero-sum security games on graphs. In: Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2011), pp. 327–334. IFAAMAS, Taipei, May 2011
18. Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S. (eds.): Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, Advances in Information Security, vol. 54. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-0977-9
19. Kaspersky Labs: Kaspersky lab and ITU research reveals new advanced cyber threat, May 2012. http://usa.kaspersky.com/about-us/press-center/press-releases/kaspersky-lab-and-itu-research-reveals-new-advanced-cyber-threat

20. Khalil, K., Qian, Z., Yu, P., Krishnamurthy, S., Swam, A.: Optimal monitor placement for detection of persistent threats. In: Proceedings of the IEEE Global Communications Conference (IEEE GLOBECOM 2016). IEEE, Washington, DC, December 2016
21. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., Tambe, M.: Computing optimal randomized resource allocations for massive security games. In: Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 689–696. IFAAMAS, Budapest, May 2009
22. Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. Nash in security games: an extended investigation of interchangeability, equivalence, and uniqueness. J. Artif. Intell. Res. **41**(2), 297–327 (2011)
23. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Trans. Softw. Eng. **37**(3), 371–386 (2011)
24. Marschalek, M., Kimayong, P., Gong, F.: POS malware revisited - look what we found inside your cashdesk. Cyphort labs special report, Cyphort, Inc. (2014)
25. McMahan, H.B., Gordon, G.J., Blum, A.: Planning in the presence of cost functions controlled by an adversary. In: Proceedings of the 20th International Conference on Machine Learning (ICML 2003), pp. 536–543. AAAI Press, Washington DC, August 2003
26. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: an approach to universal topology generation. In: Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 346–353. IEEE, Cincinnati, August 2001
27. Merritt, E.: New POS malware emerges - Punkey, April 2015. https://www.trustwave.com/Resources/SpiderLabs-Blog/New-POS-Malware-Emerges--Punkey/
28. Moreira Moura, G.C.: Internet Bad Neighborhoods. Ph.D. thesis, University of Twente, The Netherlands, March 2013
29. Nascimento, J.M., Powell, W.B.: An optimal approximate dynamic programming algorithm for the lagged asset acquisition problem. Math. Oper. Res. **34**(1), 210–237 (2009)
30. Nguyen, T.H., Wellman, M.P., Singh, S.: A stackelberg game model for botnet data exfiltration. In: Rass, S., An, B., Kiekintveld, C., Fang, F., Schauer, S. (eds.) GameSec 2017. LNCS, vol. 10575, pp. 151–170. Springer, Vienna (2017). https://doi.org/10.1007/978-3-319-68711-7_9
31. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd edn. Wiley, Hoboken (2011)
32. Rossow, C., Andriesse, D., Werner, T., Stone-Gross, B., Plohmann, D., Dietrich, C.J., Bos, H.: 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), pp. 97–111. IEEE, Berkeley (2013)
33. Scargle, J.D.: Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. Astrophys. J. **263**, 835–853 (1982)
34. Schmidt, S., Alpcan, T., Albayrak, Ş., Başar, T., Mueller, A.: A malware detector placement game for intrusion detection. In: Lopez, J., Hämmerli, B.M. (eds.) CRITIS 2007. LNCS, vol. 5141, pp. 311–326. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89173-4_26
35. Shinoda, Y., Ikai, K., Itoh, M.: Vulnerabilities of passive internet threat monitors. In: Proceedings of the 14th USENIX Security Symposium (USENIX Security 2005), pp. 209–224. USENIX Association, Baltimore, August 2005

36. Shmatikov, V., Wang, M.H.: Security against probe-response attacks in collaborative intrusion detection. In: Proceedings of the 2007 Workshop on Large Scale Attack Defense, pp. 129–136. ACM, Kyoto, August 2007
37. Spring, N., Mahajan, R., Wetherall, D., Anderson, T.: Measuring ISP topologies with Rocketfuel. IEEE/ACM Trans. Netw. **12**(1), 2–16 (2004)
38. Stinson, E., Mitchell, J.C.: Towards systematic evaluation of the evadability of bot/botnet detection methods. In: Proceedings of the 2nd USENIX Workshop on Offensive Technologies. USENIX Association, San Jose, July 2008
39. Stoica, P., Moses, R.L.: Introduction to Spectral Analysis, 1st edn. Prentice Hall, Upper Saddle River (1997)
40. Sweeney, P.J.: Designing effective and stealthy botnets for cyber espionage and interdiction: finding the cyber high ground. Ph.D. thesis, Thayer School of Engineering, Darthmouth College, August 2014
41. Venkatesan, S., Albanese, M., Cybenko, G., Jajodia, S.: A moving target defense approach to disrupting stealthy botnets. In: Proceedings of the 3rd ACM Workshop on Moving Target Defense (MTD 2016), pp. 37–46. ACM, Vienna, October 2016
42. Venkatesan, S., Albanese, M., Jajodia, S.: Disrupting stealthy botnets through strategic placement of detectors. In: Proceedings of the 3rd IEEE Conference on Communications and Network Security (IEEE CNS 2015), pp. 55–63. IEEE, Florence, September 2015. Best Paper Runner-up Award
43. Venkatesan, S., Albanese, M., Shah, A., Ganesan, R., Jajodia, S.: Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In: Proceedings of the 4th ACM Workshop on Moving Target Defense (MTD 2017), pp. 75–85. ACM, Dallas, October 2017
44. Vlachos, M., Yu, P., Castelli, V.: On periodicity detection and structural periodic similarity. In: Proceedings of the 5th SIAM International Conference on Data Mining (SDM 2005), pp. 449–460. SIAM, Newport Beach, April 2005
45. Wang, Y., Wen, S., Xiang, Y., Zhou, W.: Modeling the propagation of worms in networks: a survey. IEEE Commun. Surv. Tutorials **16**(2), 942–960 (2014)
46. Wellman, M.P., Prakash, A.: Empirical game-theoretic analysis of an adaptive cyber-defense scenario (preliminary report). In: Poovendran, R., Saad, W. (eds.) GameSec 2014. LNCS, vol. 8840, pp. 43–58. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12601-2_3
47. West, M.: Preventing system intrusions. In: Network and System Security, pp. 29–56, , 2nd edn. Syngress (2014)
48. Zeng, Y., Hu, X., Shin, K.G.: Detection of botnets using combined host- and network-level information. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), pp. 291–300. IEEE, Chicago, June 2010
49. Zhang, J., Perdisci, R., Lee, W., Luo, X., Sarfraz, U.: Building a scalable system for stealthy P2P-botnet detection. IEEE Trans. Inf. Forensics Secur. **9**(1), 27–38 (2014)