ADAPTIVE CYBER DEFENSES TO MITIGATE BOTNET-BORNE THREATS

by

Sridhar Venkatesan
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Dr. Massimiliano Albanese, Dissertation Director
_____	Dr. Sushil Jajodia, Committee Member
_____	Dr. Rajesh Ganesan, Committee Member
_____	Dr. Kris Gaj, Committee Member
_____	Dr. Stephen Nash, Senior Associate Dean, Volgenau School of Engineering
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Summer Semester 2017
George Mason University
Fairfax, VA

Adaptive Cyber Defenses to Mitigate Botnet-Borne Threats

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Sridhar Venkatesan
Master of Science
PSG College of Technology, India, 2012

Director: Dr. Massimiliano Albanese, Associate Professor
Department of Information Sciences and Technology

Summer Semester 2017
George Mason University
Fairfax, VA

Acknowledgments

I am indebted to my advisor Dr. Massimiliano Albanese for his continuous support and encouragement throughout my PhD. I would like to express my sincerest gratitude to Dr. Sushil Jajodia for his invaluable insights and criticisms that have helped me improve the quality of my thesis. I would also like to thank my dissertation committee members Dr. Rajesh Ganesan and Dr. Kris Gaj for being helpful, encouraging and accommodating throughout the process. I would also like to thank the MURI team members: Dr. George Cybenko, Dr. Michael Wellman, Dr. Demosthenis Teneketzis, Dr. Satinder Singh, Dr. Peng Liu, Dr. Kareem Amin, Mason Wright, Erik Miehling, Mohammad Rasouli and Kate Farris for the intellectually simulating discussions which have helped me to identify new directions for my thesis.

I would also like to thank my colleagues at the Center for Secure Information Systems and my friends who have stuck by me during the times of joy and despair. Last but not the least, I am eternally indebted to my family members who have always been supportive of my every endeavor.

This work was partially supported by the Army Research Office under the grants W911NF-13-1-0421 and W911NF-13-1-0317, and by the Office of Naval Research under the grants N00014-12-1-0461 and N00014-13-1-0703.

Table of Contents

List of Tables

List of Figures

Abstract

ADAPTIVE CYBER DEFENSES TO MITIGATE BOTNET-BORNE THREATS

Sridhar Venkatesan, PhD

George Mason University, 2017

Dissertation Director: Dr. Massimiliano Albanese

A botnet is a network of compromised machines remotely controlled by an attacker. Over the past two decades, the modus operandi of botnets has evolved to facilitate a wide array of attacks ranging from stealthy exfiltration of sensitive data to large-scale Distributed Denial-of-Service (DDoS) attacks. As a result, botnets have emerged as one of the biggest threats to the Internet ecosystem.

Despite several successful efforts at shutting down botnet operations, attackers quickly reconstruct them while switching to more stealthy and resilient architectures. To address this ceaseless arms race between the defenders and the attackers, a new paradigm known as Adaptive Cyber Defense (ACD) has emerged as a promising approach. ACD techniques mitigate attack campaigns by increasing the cost and complexity for a malicious actor to successfully execute an attack. In this thesis, I present ACD models and techniques to address two significant threats: data exfiltration and DDoS.

Botnets that facilitate data exfiltration persist in the target network for an extended period of time and operate in a stealthy manner. In the recent past, attackers that employ botnets to exfiltrate data have implemented a new form of stealth – known as architectural stealth – in which the attacker constructs a communication architecture that reduces the exposure of malicious traffic to the detectors. To disrupt such botnets in a resource-constrained

environment,we first develop two dynamic monitoring strategies that significantly increases the likelihood of intercepting bot traffic. Next, we design a detection mechanism, namely DeBot, that processes the intercepted traffic and exploits known intrinsic behaviors of bots to detect them. Finally, to minimize the persistence of such botnets within a network, we develop a Reinforcement Learning (RL) model that combines the proactive capability of honeypots to prevent lateral movement and the reactive nature of detection mechanisms to detect persistent bots within the network. We provide a proof-of-concept of the proposed techniques, and study their performance in a simulated environment. The results show that the proposed approaches are promising in protecting a network against long-term exfiltration campaigns by stealthy botnets.

On the other end of the attack spectrum, adversaries also employ botnets to launch large-scale volumetric DDoS attacks. To mitigate the impact of DDoS attacks, organizations are increasingly adopting proxy-based architectures. These architectures introduce a well-provisioned intermediate layer of secret proxies between end users and target services and reduce the impact of a DDoS attack by migrating the clients to new proxies and shuffling the clients across proxies so as to isolate malicious clients. However, the reactive nature of these solutions presents a weakness that we leverage to develop a new attack – the proxy harvesting attack – which enables malicious clients to collect information about a large number of proxies before launching a DDoS attack. We show that current solutions are vulnerable to this attack, and propose PROTAG – a moving target defense technique consisting in periodically and proactively replacing one or more proxies and remapping clients to proxies. Our primary goal is to disrupt the attacker's reconnaissance effort. Additionally, to mitigate ongoing attacks, we develop a new client-to-proxy assignment strategy to isolate compromised clients, thereby reducing the impact of subsequent attacks. We validate our approach both theoretically and through simulation, and show that the proposed solution can effectively limit the number of proxies an attacker can discover and isolate malicious clients.

Chapter 1: Introduction and Motivation

Confidentiality, Integrity and Availability (CIA triad) are at the core of information security. In the past decade, botnets have emerged as an attractive toolkit for cyber-criminals to plague an organization's information security. A botnet is a network of compromised machines which are remotely controlled by an attacker. Ever since its first appearance, the modus operandi of botnets has evolved to enable attackers to engage in a plethora of malicious activities. In the beginning, botnets were created and maintained for large-scale overt activities such as DDoS attacks and spam. Such botnets required the coordination of a large number of bots in order to be effective, i.e., the success of these attacks relied on the *size* of the botnet. However, over the past decade, attackers have identified new uses for botnets, such as data theft. These botnets need to *persist* in the target network for an extended period of time in order to operate effectively.

Due to the risks posed by botnets to the Internet ecosystem, law enforcements continually conduct botnet takedown operations by shutting down the Command and Control (C&C) servers [4]. However, attackers restore botnet operations immediately and adopt resilient (such a peer-to-peer (P2P) networks [5]) and stealthier communication architecture (such as HTTP [6]) to withstand future takedown attempts. As a result, there is an arms race between the attacker and defender. In order to address this arms race, in this dissertation, I develop Adaptive Cyber Defense (ACD) models and techniques to tip the balance in favor of the defender.

ACD techniques mitigate attack campaigns by increasing the cost and complexity for the attacker to successfully execute the attack [7, 8]. Among different manifestations of ACD techniques, Moving Target Defense (MTD) techniques have emerged as a game-changing paradigm and advocates proactive re-configuration of a system so as to continually shift the

system's attack surface [7, 8]. Continual re-configuration of a system reduces an attacker's certainty of the system's attack surface and consequently, increases the effort to successfully execute an attack. Inspired by this paradigm, my ultimate objective is to increase an attacker's effort and disincentivize them from performing botnet-assisted malicious activities. In particular, in this dissertation, I address two major threats due to botnets: long-term data exfiltration and DDoS attacks.

1.1 Data Exfiltration by Stealthy Botnets

Data exfiltration is an unauthorized transfer of sensitive information from a target's network to a remote location controlled by an attacker. Data exfiltration violates the confidentiality of an organization's digital assets by leaking sensitive data through an exfiltration channel such as social engineering [9]. Reports by Kaspersky labs [10] and Mandiant [11] show that the threat actors infiltrate an organization's network and persist in the system for several years, mapping out the network before exfiltrating sensitive and valuable intellectual property to a remote drop-site. Due to the wealth of information that can be acquired through such an attack vector, threat actors ranging from competitors of an organization to foreign nations invest huge amount of time and money to design such Advanced Persistent Threats (APT). With a rising trend in the use of botnets to facilitate attacks that are motivated by profit such as harvesting information and bitcoin mining, it is anticipated that, in the future, botnets will be employed to facilitate stealthy attack vectors such as APTs [12]. Botnets that engage in long-term exfiltration campaigns need to *persist* in the target network for an extended period of time to operate effectively. To persist in the network, botnets require two properties: *stealth* and *resilience*.

Stealth is critical for a botnet's survival and is achieved by implementing either one or a combination of the following types of stealth: Anti-signature stealth and Architectural stealth [13]. In a monitored environment, anti-signature stealth obfuscates a bot's observable behavior to evade detection. For instance, to evade a network-based detection mechanism, attackers can manifest anti-signature stealth by randomizing the spatiotemporal properties

of the traffic generated by a bot or a group of bots. Several existing botnets such as Zeus implement anti-signature stealth by encrypting the payload of the traffic while communicating with a peer or an external C&C server. Besides anti-signature stealth, advanced botnets have adopted a new form of stealth known as architectural stealth. The communication architecture of these botnets is constructed such that the volume of malicious traffic intercepted by detectors is reduced. For instance, malwares such as Duqu [14], Regin [15] and BlackPOS [16] – which were responsible for several exfiltration campaigns from corporate and government networks – construct a communication architecture between the bots such that one of the bots aggregates data from the remaining bots and acts as a proxy to relay the data to an external C&C server. Such a communication architecture was designed to limit the volume of suspicious traffic intercepted by the detectors at the network perimeter; thereby, reducing the likelihood of detection [14, 15].

In addition to stealth, a botnet's communication architecture should be resilient to bot takedown efforts by the defender. This property guarantees that the attacker can maintain a foothold in the target network and continuously communicate with all the bots in the botnet. Several advanced botnets such as Zeus GameOver, Waledac – which steal data – switch to a fallback channel (such as DGA) when the bots are unable to contact the C&C server using their main communication channel (e.g., when all the peers of a bot in a P2P botnet is down) [17].

Defending a network against attack campaigns by such stealthy botnets calls for a large-scale network-wide monitoring solution. However, monitoring a network in a resource-constrained setting presents three challenges.

First, to detect architectural stealthy botnets that minimize the interception of suspicious traffic, it is crucial to identify the locations within a network that can potentially intercept large volumes of bot traffic. A straightforward approach to ensure that all bot traffic is intercepted would be to simultaneously mirror traffic generated from all segments of the network to a central location for analysis. However, due to the poor scalability of existing detection mechanisms coupled with an ever-growing internal benign traffic volume,

detecting bots in a reasonable amount of time becomes infeasible. Such a setting gives rise to the following research problem – In a resource-constrained environment, what monitoring strategy must be employed to guarantee a high likelihood of intercepting bot traffic?

Second, existing state-of-the-art network-based detection mechanisms [18–22] were designed to detect bots that communicate with their peers or the C&C server located outside the network. These mechanisms either drop internal traffic [18], ignore traffic in a predefined whitelist [20–22] or use a biased traffic sampling algorithm [19] to significantly reduce the false positive rate and the volume of traffic processed by the respective mechanisms. However, bots that operate under architectural stealth, generate traffic that is largely internal to the network and also, blend with the benign traffic. Modifying the operations of existing mechanisms result in a trade-off between the false positive rate and the detection accuracy. On the one hand, considering packets from a subset of the network segments (the choice of segments could be guided by a monitoring strategy) and processing all the traffic through them (without dropping any internal traffic) would result in a high false positive rate. While on the other hand, dropping internal traffic would reduce the detection rate. This trade-off paves the way to the following research problem – What intrinsic characteristics of bots can be leveraged to distinguish benign traffic from the traffic generated by bots that operate under architectural stealth?

Finally, as detection mechanisms are imperfect, bots may remain undetected for an extended period of time. Studies [23] show that the median time to detect a breach in an organization is 205 days ($\approx$ 7 months). Therefore, to reduce the lifetime of a botnet, we consider a defense-in-depth approach that combines honeypots and network-based detection mechanisms to detect and remove bots. In particular, honeypots detect bots during lateral movement while the detection mechanism detects bots that persist in the network. However, in a resource-constrained environment, the defender can deploy only a limited number of honeypots and monitor only a subset of network segments at a given time. Such a setting introduces the following research problem – In a resource-constrained environment, what strategy must be employed to deploy honeypots and monitors within a network to reduce

the lifetime of a botnet?

In this dissertation, I will address these three challenges and propose solutions that incorporate the underlying principles of ACD.

1.2 Distributed Denial of Service Attack

On the other end of attack spectrum, botnets are commonly employed to enable massive-scale attacks such as DDoS. Studies have shown that a single DDoS attack cost a victim more than $400,000 [24]. Volumetric or bandwidth-based DDoS attacks disrupt the availability of a service to its legitimate users by flooding the target with large volumes of traffic; thereby, congesting the target's network infrastructure. While attackers are constantly developing new types of DDoS attacks (such as multi-vector attacks and application-layer attacks), volumetric attacks continue to be amongst the most common type of attack [25]. These attacks leverage the fact that the target's location can be easily identified (e.g., through DNS lookup), allowing the attackers to directly reach the target.

A possible approach for preventing an attacker from directly reaching a target is to introduce indirection layers between users and target services [26, 27]. In such an architecture, in order to access a protected service, users need to authenticate with an intermediate layer of systems which act as proxies between the clients and the services, and relay incoming requests to the servers. Additionally, to protect against Internet-wide scans, the target service accepts traffic only from the designated systems in the indirection layer. As a result, the architecture shifts the attack surface from the target service to the proxy servers i.e., to launch a successful a DDoS attack, the attacker must overwhelm *all* the proxies with illegitimate requests simultaneously.

One of the main benefits of this solution is that it can mitigate DDoS attacks against an enterprise without requiring an Internet-wide adoption. However, the indirection layer needs to be well-provisioned to effectively out-muscle the attacks. As we cannot indefinitely add resources to the indirection layer and keep adding proxies, effective mitigation of DDoS attacks in a proxy-based architecture raises the following research challenge – Can we develop

a client-to-proxy assignment strategy that increases the complexity for the attacker to launch an effective DDoS attack? Additionally, in order to reduce the impact of a wave of DDoS attacks, can we develop a strategy that identifies insiders who outsource the location of active proxies to botnets that, in turn, to launch the DDoS attack?

1.3 Contributions and Organization

As mentioned earlier, in this thesis, I present ACD models and techniques to address the challenges posed in detecting and mitigating two major botnet-borne threats: long-term data exfiltration and DDoS attacks. To this end, in Chapter 2, I will first discuss related work and the current state-of-the-art approaches towards addressing these challenges.

To disrupt stealthy botnets that exfiltrate data from a network, I will first address the challenge of developing a monitoring strategy to intercept bot traffic in Chapter 3. Here, I model the defender's resource-constrain by bounding the number of network segments that can be monitored at any given time and adopt a graph-theoretic approach to develop two dynamic monitor placement strategies which continuously change the monitored segments of the network. Specifically, I developed several strategies based on centrality measures that capture important properties of the network. The objective of these strategies is to increase the likelihood of intercepting bot traffic by creating uncertainty about the location of the monitors. Furthermore, the uncertainty introduced by the proposed strategies increases an attacker's effort and forces them to perform additional actions in an attempt to exfiltrate data through a monitor-free path.

Next, I will leverage the monitoring strategies proposed in Chapter 3, to develop a scalable detection mechanism, namely DeBot, to detect botnets that operate under architectural stealth in Chapter 4. DeBot processes traffic snapshots from different network segments and leverages intrinsic characteristics of bots to detect them. In particular, DeBot leverages the difference in the flow statistics of the traffic generated by bots and benign applications and, the periodicity in the communication between a bot and its peers to identify suspicious host pairs within the network. Finally, in Chapter 5, I will address the challenge of developing a

strategy for placing a limited number of honeypots and monitors in a network. To this end, I model the problem as a sequential decision-making problem and develop a Reinforcement Learning (RL) model to learn and thus, control the evolution of a botnet that propagate and persist in the target network.

To mitigate DDoS attacks, several MTD-based approaches have been proposed that continuously re-assign the clients to proxies during a wave of DDoS attacks [28–30]. In Chapter 6, I will first present a new type of reconnaissance-based attack – the proxy harvesting attack – which significantly diminishes the effectiveness of existing proxy-based architectures that employ the principles of MTD. To counter such an attack, I developed a lightweight defense technique, PROTAG, which proactively reconfigures the attack surface by periodically replacing one or more proxies and reassigning clients to new proxies. The main objective of the proposed technique is to disrupt the attacker's reconnaissance efforts to launch a significant DDoS attack. Finally, I will combine PROTAG with a simple attacker isolation technique to minimize the impact of a wave of DDoS attacks. Finally, in Chapter 7, I will present some possible directions for future work.

Chapter 2: Related Work

2.1 Evolution of Botnet Communication Architecture

The communication architecture of a botnet plays a crucial role in accomplishing its malicious attack campaign. Ever since the first appearance of a botnet, its communication architecture has evolved in response to the increasing shutdown efforts by the network defenders. A timeline of significant events in the evolution of botnet C&C infrastructure is shown in Fig. 2.1. In the beginning, attackers developed centralized architectures in which the bots setup a direct communication channel with the C&C server; thereby, minimize the latency in the propagation of commands from the C&C server to the bots. However, due to single-point failure of centralized architecture, attackers began adopting resilient and stealthy communication architectures such as P2P, Domain and IP fluxing.

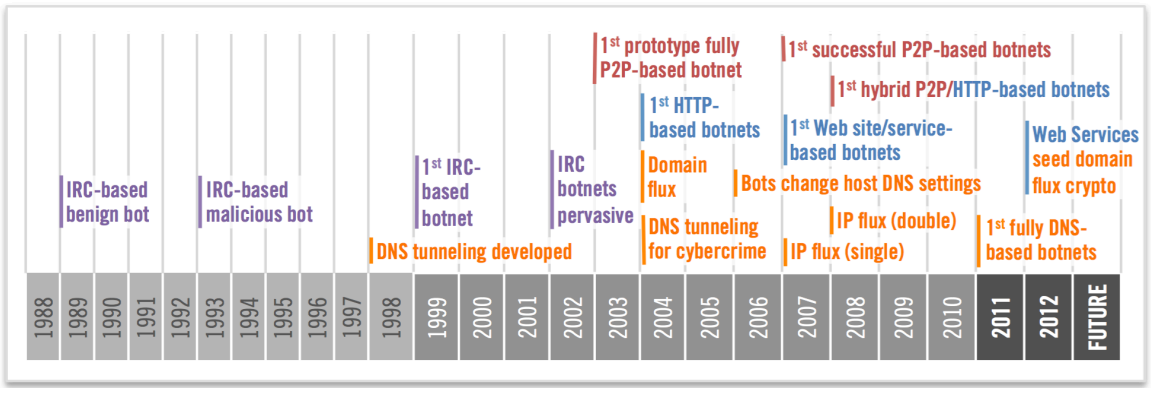


Figure 2.1: Evolution of Botnet C&C Infrastructure [2]

In P2P botnets, the bots form a distributed C&C topology to remain resilient to takedown attempts by law enforcements and security researchers. Such P2P-based C&C architectures have become a popular choice for attackers e.g., Sality for spamming [31], Zeroaccess for Bitcoin mining [32].

The operations of existing P2P botnets differ in three aspects: the communication protocol, the message propagation method and the direction of communication [33]. There are two types of communication protocols: structured and unstructured. Structured P2P botnets such as Storm [34] are build on existing P2P networks (such as Gnutella, Overnet) where the bots co-exist with benign users. Each bot maintains a Distributed Hash Table (DHT) to store the IDs of its peer bots. To propagate a message, structured P2P botnets first identify the peers and use a DHT lookup to route commands to the chosen peers. On the other hand, messages in an unstructured P2P botnet is propagated from a bot to its peers through a gossip protocol; a protocol in which the bot sends the commands to all its peers simultaneously. Finally, a bot in a P2P botnet may either *pull* updates or *push* commands to its peers. The choice of direction of communication depends on the location of bots; pushing commands from a bot to its peers reduces communication latency, while pulling updates from its peers enables non-routable bots (such the ones behind a NAT) to join the botnet.

Besides P2P-based C&C infrastructure, domain and IP fluxing techniques enable an attacker to circumvent large-scale takedown attempts by continuously changing the location of the C&C server. In domain fluxing, the botnet operator uses a Domain Generation Algorithm (DGA) to generate a large number of domain names in a pseudorandom fashion [35]; a subset of these are registered to host the C&C servers. Bots in such botnets use the DGA to generate the domain names (with the same seed) and query the DNS server until one of the domain names resolve to the C&C server [36]. As the domain names are generated in a pseudorandom manner, network defenders cannot accurately predict the location of the C&C server. On the other hand, in IP fluxing, attackers register a domain name for the C&C server while continuously change the set of IP addresses associated with the domain

[36]. In order to implement this, the attackers exploit the time-to-live (TTL) value in the DNS resource record; a record that associates the domain name with the IP address. To evade takedown attempts, attackers set a small TTL value thereby, enabling them to change the IP address of the C&C at a high frequency.

In addition to resilience, attackers have also begun adopting stealthier communication channels to evade detection at network perimeter devices. Channels such as HTTP, DNS are typically whitelisted by network perimeter devices. As a result, bots communicate with the C&C server by encoding their messages in the protocol headers of these communication channels. For instance HTTP-based botnets such as Zeus, SpyEye and Citadel communicate with the C&C server through encrypted HTTP GET/POST message requests [6].

2.1.1 Centralized Botnet Detection

Over the past decade, several tools and techniques have been developed to detect different classes of botnets. At the outset, researchers treated botnets as malwares and developed signatures by reverse engineering bot malware samples and analyzing their behavior in a sandboxed environment. Based on this analysis, researchers developed mechanisms that exploit known IRC bot nicknames [37], known C&C domains to identify compromised machines within a network. While signature-based solutions captured known botnets, attackers evaded them by altering the spatiotemporal characteristics of their bots and adopting resilient communication architectures. Such an evolution of botnets prompted the need to develop behavior-based detection mechanisms. To this end, detection mechanisms such as BotSniffer [38], BotGAD [39] were designed to detect centralized botnets by correlating network activities of hosts and identifying synchronized behavior across different hosts while BotHunter [40] exploited the ordered sequence of messages between a bot and a C&C server to detect compromised machines.

2.1.2 Graph-theoretic Approaches to Disrupt Botnets

With the emergence P2P-based botnets, several detection mechanisms have been developed for different types of networks. For large-scale detection and removal of bots, research in [41, 42] study the graph-theoretic properties of a P2P botnet to identify bots that need to be removed to disrupt the botnet. These techniques implement a *crawler* that infiltrates the botnet as a bot and requests a list of peer bots. Upon obtaining a list of neighboring bots, the infiltrating bot iteratively requests every bot in its neighbor list for addition bots. In order to enlist the bots that are behind a NAT, Kang et al. [43] introduced *sensors* that act as superpeers – a responsive bot with a large uptime. These sensors wait for the bots in non-routable regions of the Internet to connect to it. Once all bots have been enumerated, the connectivity of the botnet can be obtained. By studying the graph-theoretic properties such as degree, and clustering coefficient, they enable in identification of bots whose removal would partition the network; thereby disrupt their operation. A formal treatment of existing graph-based disruption methods is provided in [33].

At the ISP-level, BotGrep [44] was proposed as a scalable graph-theoretic approach to detect P2P botnets. It exploited the highly structured communication between bots to isolate P2P botnet's communication subgraph from the ISP communication graph. To improve its accuracy, BotGrep needs C&C traffic samples observed across several vantage points in the network. These vantage points in the network are determined by considering graph-theoretic properties of the communication graph.

2.1.3 P2P Botnet Detection in Enterprise Networks

For enterprise network, techniques in [18] (scalable version in [19]) and [20] are the state-of-the-art mechanisms that were designed to detect P2P botnets.

In [18], Gu et al. proposed BotMiner to detect botnets independent of its protocol (such as IRC) or communication architecture (such as centralized, P2P). BotMiner analyzes network traffic traversing the network gateway and groups hosts that exhibit similar communication patterns (based on features extracted from capture traffic) and similar activity

(such as spamming, scanning etc). In order to scale BotMiner for high speed networks, Zhang et al. in [19] developed a packet sampling technique known as *BSampling* which reduces the volume of traffic that needs to be analyzed by BotMiner. The BSampling algorithm leverages a bot's intrinsic characteristic to persistently communicate with its peers or C&C and that the corresponding communication pattern is similar across different bots. Given an upper bound on the capacity of the monitoring device, the BSampling algorithm periodically updates the sampling rate for each host so that the packets corresponding to hosts that exhibit similar communication patterns are sampled at a higher rate than other packets. As a result, hosts that persistently exhibit similarity in communication pattern will be identified as suspicious hosts. Subsequently, only traffic from the suspicious hosts are analyzed by BotMiner.

Zhang et al. in [20] developed a scalable technique to detect stealthy P2P bots within a network. In this work, the authors consider P2P botnets that implement anti-signature stealth by encrypting its traffic and co-existing with benign P2P applications (such as Bittorrent, Skype) to blend malicious traffic with the background traffic. Originally, the detector was developed based on three properties exhibited by such stealthy P2P bots: (i) bots communicate with *several* distinct peers outside the network, (ii) the active time of a bot is *close* to the active time of the underlying system on which they are running and (iii) the traffic generated by the bots have flow characteristics that are different from benign P2P applications. The detection mechanism leverages these properties to derive statistical fingerprints of different P2P communications to differentiate between hosts belonging to a legitimate P2P network from those belonging to a P2P botnet.

2.1.4 Advanced Botnet Designs

With the evolution of botnet C&C architectures from simple static centralized C&C to advanced P2P-based C&C, researchers have investigated the possibility of other resilient botnet communication structures [45]. Such studies assist in proactively designing detection mechanisms against them. Recently, authors in [46] designed OnionBot, a resilient and

stealthy P2P botnet whose communication infrastructure is build using the Tor's hidden services. In OnionBot, a bot's IP address is hidden and can be reached through a .onion address. A bot's peer list only contains the .onion address of its peers and uses the Tor network to communicate with its peers and C&C. Therefore, a defender cannot learn the IP address from the peer list of a captured bot, resulting in a highly resilient botnet. Additionally, OnionBot minimizes the network footprint by maintaining a low degree and a low diameter of the resulting botnet, thereby remaining stealthy. Along similar lines, Sweeney and Cybenko in [12] emphasize the importance of physical location of bots in a network (called cyber high ground) to perform stealthy missions such as data exfiltration. To this end, they design a P2P botnet whose objective is to exfiltrate data traversing a mission-critical communication channel in a stealthy manner. For a given network topology, they design a P2P overlay communication structure which maximizes effectiveness (to exfiltrate data) while maintaining stealth (by avoiding detectors and reducing network footprint).

2.1.5 Scalability Issues with Botnet Detection Mechanisms

One of the challenges of existing fine-grained detection mechanisms is that they scale poorly with traffic volume. Existing techniques attempt to improve scalability by employing either one or a combination of the following methods: (i) filtering traffic and reducing the volume of analyzed traffic, or (ii) using a biased traffic sampling algorithm, or (iii) analyzing with a low-dimensional feature space. For instance, BotMiner [18] filters any traffic between hosts within the network and all traffic to/from whitelisted destinations. Similarly, P2P botnet detection mechanism developed by Zhang et al. in [20] does not consider any network flow that was previously resolved in DNS responses. Ramachandran et al. [47] and Zhang et al. [19], on the other hand, propose traffic sampling algorithms to reduce the volume of traffic analyzed at the network gateway. These algorithms bias the sampling rates such that low-volume traffic with bot-like characteristics are sampled at a higher rate than other packets. Besides reducing the traffic volume, techniques such as BAYWATCH [22] and PsyBoG [21] detect bots by testing for periodicity in the connection patterns between internal hosts and

external destinations. In the presence of periodic benign traffic, detecting stealthy botnets would require a rich set of features to reduce false positive rates.

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 [48–51] in which the defender's objective is to optimally place a limited number of monitors to protect a set of target servers. The game-theoretic models in [48, 50] develop optimal placement strategies to detect intrusion attempts by considering all possible routes through which the attack can reach the target server from a given set of entry points. While the models in [49, 51] develop optimal placement strategies to minimize the attacker's control over the target server.

2.2 Existing Approaches to Mitigate DDoS Attack

Since the first appearance of DDoS attacks in 2000 [52], several defense mechanisms based on filtering attack traffic or limiting a client's share of bandwidth have been proposed to mitigate attacks at the Internet scale [53]. An in-depth survey of DDoS flooding attacks and existing approaches to mitigate them is provided in [54]. However, these solutions rely on large-scale adoption and coordination among different network elements such as routers and ISPs in order to be effective. These limitations have paved the way to overlay-based defense approaches which hide the location of target servers behind a well-provisioned, distributed overlay network [55].

MOVE [56] is one such overlay-based architectures which protects services against attackers who control and disrupt only a subset of network elements. In MOVE, the target service accepts traffic only from a subset of overlay nodes (called secret servlets) and when a DoS attack is detected, the target service is migrated to a new hosting site to alleviate the impact of the attack. Early overlay-based architectures – such as MOVE – did not account for a threat model in which an attacker can persist in the system and repeatedly launch DoS attacks, thereby nullifying the advantage of migrating the service to a new location. To this

end, overlay-based architectures such as MOTAG [28, 29], DoSE [30] leverage the principles of MTD to modify the client-to-overlay node association after a DoS attack, in the hope that insiders – who persist after the attack – will be eventually isolated from innocent clients.

Wang et al. [28] designed MOTAG to mitigate the persistence of DoS attacks against online services. MOTAG protects applications by hiding the application server's location behind a large pool of proxy servers. The addresses of these proxy servers are not available to unauthorized clients. After authentication, an authorized client is provided with the IP address of a random proxy server which forwards incoming request to the application server. This ensures that unauthorized clients (such as bots) who do not know the address of a proxy will not be able to disrupt client-server communication. In addition, MOTAG provides a mechanism to combat insiders who outsource the address of these proxies to external attackers (e.g., botmasters). When proxy servers are discovered and attacked, MOTAG *moves* the clients connected to attacked proxies to new proxies and *shuffles* client assignment to these new proxies such that innocent clients are isolated from insiders. This architecture was extended in [29] to protect Internet services that support both authenticated and anonymous users. The proposed architecture leverages the resource elasticity in a cloud environment to instantiate new replica servers when existing servers are under attack. The load balancer identifies each client with its IP address and assigns new clients to an active replica server. Similar to the original design, when a subset of replica servers are under attack, a coordination server instantiates new replica servers followed by moving and shuffling clients connected to the attacked servers to the new replica servers.

The shuffling process to isolate insiders in MOTAG-based architectures does not consider the cost-overhead to instantiate and maintain new proxies. To address this issue, Wood et al. [30] proposed DoSE, a cloud-based architecture that provides a cost-effective mechanism to isolate insiders and confine the attack to a few proxies. For each client, DoSE assigns a risk value which captures the likelihood that the client will indulge in an attack. DoSE also assigns an upper bound on the risk that any proxy node can tolerate. Based on these risk parameters, DoSE assigns clients to proxies and in the face of an attack, the risk

value of clients associated with attacked proxies are updated. To absorb the impact of the DDoS attack, new proxies are spawned and clients associated with the attacked proxies are reassigned to new proxy based on their updated risk values. By maintaining a state for each client (through risk), DoSE limits the number of proxies needed to identify insiders, thereby reducing the cost to maintain such an architecture.

Current moving-target based DDoS defense mechanisms are reactive in nature. To this end, Duan et al. [57] proposed a proactive DoS mitigation technique for wired networks to tolerate attacks on a small portion of the network. Given a source and destination on a wired network, they derive several optimal routing paths which minimize overlap with the recently used routing path and, simultaneously, meet given QoS constraints. One of these routing paths is chosen at random for forwarding packets from the source to the destination. Inspired by the frequency hopping in wireless networks, Mittal et al. [58] proposed Mirage, a moving target defense architecture in which the victim server hops to an IP address chosen at random from a pre-defined set of IP addresses. When the victim server is under attack, it signals the new IP address to a puzzle server and the upstream routers. Clients connected to the victim server are challenged with cryptographic puzzles; the solution of which contains the new IP address at which the victim server is hosted. As the attacker spends extra computational resources to solve the puzzle, the proposed approach is expected to delay the attack. In addition to be being reactive to an attack, Mirage requires a large pool of IP addresses and hence, it is effective only in IPv6 networks.

Chapter 3: A Moving Target Defense Approach to Intercept Stealthy Botnet Traffic

3.1 Introduction

Consider a network that is hosting sensitive data. An attacker's objective is to exfiltrate the sensitive data from the network. To this end, the attacker constructs a architectural stealthy botnet within the target network to steal the data and subsequently, relay it to a remote C&C location. The defender, on the other hand, analyzes traffic from different segments of a network to detect exfiltration activity by such botnets. However, due to the poor scalability of existing detection tools, traffic from all network segments cannot be simultaneously analyzed for the presence of botnet activity. To this end, in this chapter, we develop a strategy that dynamically monitors different segments of a network to increase the likelihood of intercepting bot traffic.

3.1.1 Preliminaries

Let $G = (V, E)$ be a graph representing the physical topology of the network, where V is a set of network elements – such as 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.

Typically, traffic between any two nodes is routed using a routing algorithm/policy. Let Π_G denote the set of all simple paths $\pi(v_i, v_j)$ between any pair of nodes $(v_i, v_j) \in V \times V$. Then, a routing algorithm can be formally defined as a mapping R_A from the set $V \times V$ of all pairs of nodes in the connectivity graph to Π_G, such that

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

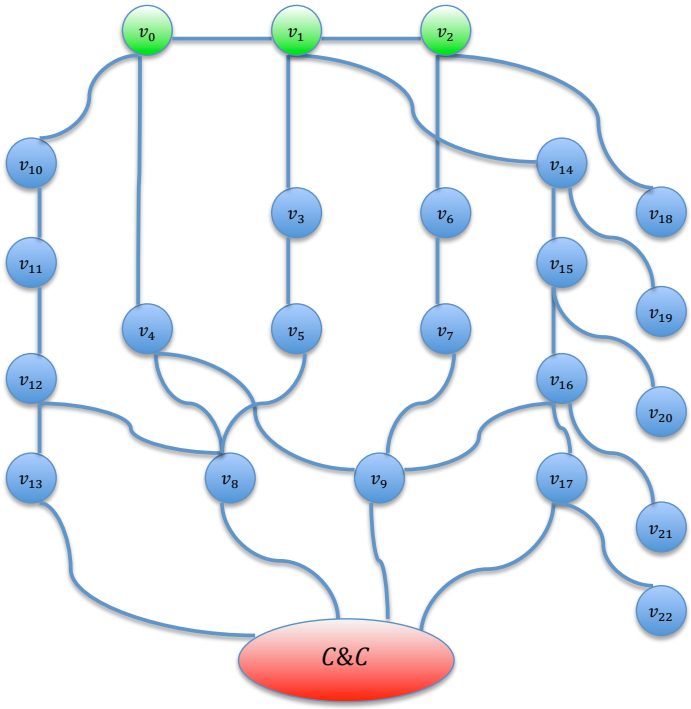


Figure 3.1: Example of network graph

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, as determined by the routing protocol/policy. 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/policy 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.

Example 3.1.1. *In the example of Figure 3.1, the set of mission-critical channels is* $C =$

$\{(v_0, v_1)\}$ *and the set of mission-critical nodes is* $N = \{v_0, v_1, v_2\}$.

To exfiltrate traffic from a set N of mission-critical nodes, the attacker compromises a set $B \subset V$ of network nodes and creates an overlay network – consisting of bots installed on nodes in B – to forward eavesdropped traffic to the remote C&C server. Although the defender may not know the exact location of the remote C&C, we assume that the defender is aware of potential C&C locations. In theory, any destination that is located outside the monitored network could be a potential C&C. However, research by César [59] and Collins et al. [60] shows that certain IP address ranges are known to participate in malicious campaigns. This information can be leveraged to identify potential C&C locations.

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

$$\pi_e(b_0, C\&C) = \langle b_0, v_1^0, v_2^0, \ldots, v_{l_0}^0, b_1, v_1^1, v_2^1, \ldots, v_{l_1}^1, b_2, \ldots, b_r, v_1^r, v_2^r, \ldots, v_{l_r}^r, C\&C\rangle \tag{3.1}$$

where $R_A(b_i, b_{i+1}) = \langle v_1^i, v_2^i, \ldots, v_{l_i}^i\rangle, \forall i \in [1, r-1]$ *and* $R_A(b_r, C\&C) = \langle b_r, v_1^r, v_2^r, \ldots, v_{l_r}^r, C\&C\rangle$.

Example 3.1.2. *In the running example given in Figure 3.1, if* $\{v_{15}, v_9\} \in B$ *are bots and the attacker chooses to exfiltrate traffic through the overlay path* $\pi_o(v_1, C\&C) = \langle v_1, v_{15}, v_9, C\&C\rangle$*, then the corresponding exfiltration path is* $\pi_E(v_1, C\&C) = \langle v_1, v_{14}, v_{15}, v_{16}, v_9, C\&C\rangle$.

For a given set of mission-critical nodes N, the defender's objective is to intercept and detect exfiltration traffic. In our work, we model the resource-constrains of the defender by bounding the number of monitors, k, that can be deployed in the network. In order to monitor the network for botnet activity, the defender chooses a subset of nodes $D \subset V$ where $|D| = k$ and inspects traffic flowing through them.

Definition 2 (Interception)**.** *Given the set N of mission-critical nodes for a network $G = (V, E)$, let $B \subset V$ be a set of bots controlled by the attacker. An exfiltration attempt over a exfiltration path $\pi_E = \langle v_0, v_1, v_2, \ldots, v_r, C\&C \rangle, v_o \in N$ is the said to be* intercepted *iff the exfiltrated traffic traverses a monitor node, that is, $\exists v_d \in V_D$ s.t. $v_d \in \pi_E$. A botnet is said to be* stealthy *with respect to N, iff no exfiltration path between nodes in $N \cap B$ and a C&C site can be intercepted.*

Several detection mechanisms have been proposed to detect botnets within a network [18, 20, 40, 61]. These detection mechanisms leverage the fact that the bots need to communicate with their peers or the C&C server to relay the captured data. Unfortunately, existing detection mechanisms suffer from false positives and false negatives, therefore exfiltration attempts may go undetected even when a monitor is placed on a node along the exfiltration path. However, for the purpose of our analysis, the accuracy of detection mechanism is not as critical as their placement. Furthermore, to operate under architectural stealth, a prudent attacker – who does not want any portion of the exfiltration traffic to be intercepted and eventually detected before reaching a C&C site – will opt for creating more bots in order to establish a monitor-free path, rather than having the traffic routed through monitors, irrespective of their false positive and false negative rate. Based on these considerations, we ignore the accuracy of detectors and assume that any exfiltration attempt going through a monitor node is detected.

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

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

In [62], we proposed an approach to deploy monitors on selected network nodes, so as to reduce the exfiltration surface by either completely disrupting communication between the bots and C&C nodes, or at least forcing the attacker to create more bots, thereby increasing the footprint of the botnet. As the monitor placement problem is intractable, we proposed heuristics based on several centrality measures. Specifically, it was shown that the *iterative mission-betweenness centrality strategy* yields the best results. In this strategy, after a node has been selected as a monitor, the mission-betweenness centrality of all non-detector nodes that belong to the exfiltration surface is recomputed, and the node with the highest centrality is chosen for placing an additional monitor. In practice, this approach prevents two or more monitors from being placed on the same high-centrality path.

Although the above 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 monitor-free path to exfiltrate data. In this work, we overcome this limitation by designing detector placement strategies that *dynamically* change the exfiltration surface by continually altering the placement of monitors.

3.1.2 Threat Model

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.

- The attacker can discover the topology of the network, and is aware of what nodes are mission-critical. Reports by Kaspersky labs [10] and Mandiant [11] 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 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 $\mathcal{T} = \langle t_1, t_2, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, i.e., at each time point t_i, the attacker transmits data segment d_i over the network to C&C. The attacker is said to have successfully exfiltrated a communication session between any two mission-critical hosts iff the attacker exfiltrates all the m data segments by time t_m.

- The attacker is aware of the monitor placement strategy employed by the defender.

3.2 Defender's Model

To detect exfiltration over a network, the defender deploys monitors on selected network nodes. Placing monitors on different vantage points in the network reduces single-point failures of monitoring traffic through network perimeter devices and increases the opportunities to intercept and detect botnet activity. In our defender's model, we consider a resource-constrained setting where the defender can only deploy k monitors. The upper bound on the number of monitors can be determined as follows: For the purposes of modeling, given the computation resources available for detection, let Q denote an upper bound on the number of packets that can be processed by a detection mechanism. Assuming that the volume of traffic across different portions of the network is uniform, say P packets/hour, then the number of segments monitored at any given time period can be approximated as $M_{max} = \lfloor \frac{Q}{P \cdot t_{obs}} \rfloor$, where t_{obs} is the length of the monitoring period in hours. Here, if there are K potential monitoring locations, then all M_{max}-sized combinations of locations are admissible solutions. It should be noted that in reality the traffic volume across different portions of the network may not be uniform; however, the average traffic volume at different locations may be known through historical data. In such situations, the problem of finding the feasible solutions for monitor placement is NP-Hard as it can be shown that the 0/1 Knapsack problem can be reduced to this problem in polynomial time. Although the problem is NP-Hard for large networks, the set of feasible solutions can be obtained in polynomial time for small-medium

scale networks that have fewer monitoring locations. In the following, we provide a formal definition of the notion of *monitor 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 on the Internet, i.e., outside the network perimeter. Similarly, for an ISP network, potential C&C sites could be located outside the administered domain. Furthermore, research by Moreira Moura [59] and Collins et al. [60] shows that certain IP address ranges are known to participate in malicious campaigns. This information can be leveraged to identify potential C&C locations. Due to the conservative estimate on the location of potential C&C sites, blacklisting traffic destined to these locations will adversely impact the utility of the network.

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

As mentioned earlier, static placement of detectors suffers from the drawback that a persistent attacker – who is aware of the network topology – can discover the location of monitors and identify monitor-free paths from mission-critical nodes to C&C sites. The resulting exfiltration surface remains static during exfiltration and, therefore, cannot intercept bot traffic. Hence, to increase the probability of interception and subsequent detection, we propose to continually shift the exfiltration surface by dynamically changing the location of detectors. In our model, we discretize time as a finite sequence of integers $\mathcal{T} = \langle t_1, t_2, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, for some $m \in \mathbb{N}$, such that for all $1 \leq i < m$, $t_i < t_{i+1}$. In the following, we slightly abuse notation and, for the sake of presentation, we treat the sequence $\mathcal{T}$ as a set when appropriate. We can now define how placements can evolve over time.

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

Intuitively, for each time point in $\mathcal{T}$, a temporal k-placement deploys monitors on k network nodes. In order to create uncertainty for the attacker with respect to the location of

monitors, we choose k-placement functions by using a probability distribution over temporal placements.

Definition 6 (Temporal probabilistic k-placement)**.** *A* temporal probabilistic k-placement *(*tp-k-*placement) is a function* $\tau : \mathcal{TP}_k \to [0,1]$ *such that* $\sum_{tp \in \mathcal{TP}_k} \tau(tp) = 1$.

Example 3.2.1. *Figure 3.2 shows an example of temporal probabilistic k-placement τ for the graph of Figure 3.1 and for $k = 2$. Each table in the figure represents a different temporal k-placement tp (for the sake of presentation, we assume those shown are the only possible temporal k-placements in $\mathcal{TP}_k$). Note that $\sum_{tp \in \mathcal{TP}_k} \tau(tp) = 1$. For any given temporal k-placement tp, the i-th column – with $i \in \{1, 2, \ldots, m\}$ – in the corresponding table 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 monitors.*

Let the indicator random variable $I^v_{t_i}$ be associated with the event that node v is chosen as a monitor 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 monitor at time t_i can be derived as

$$\begin{aligned} pr^v_{t_i} &= Pr(I^v_{t_i} = 1 \mid \tau) \\ &= \textstyle\sum_{tp \in \mathcal{TP}_k \ s.t. \ (\exists pl \in \mathcal{P}_k)(tp(t_i) = pl \wedge pl(v) = 1)} \tau(tp) \end{aligned} \tag{3.2}$$

Therefore, at time t_i, a defender chooses k nodes for monitor placement by sampling from the distribution given in Eq. 3.2. We denote such a strategy by $\mathcal{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$.

3.3 Metrics

To evaluate the performance of a defender strategy, we present two metrics: the *minimum interception probability* and the *attacker's uncertainty*. The minimum interception probability provides a theoretical lower bound on the probability that an exfiltration activity

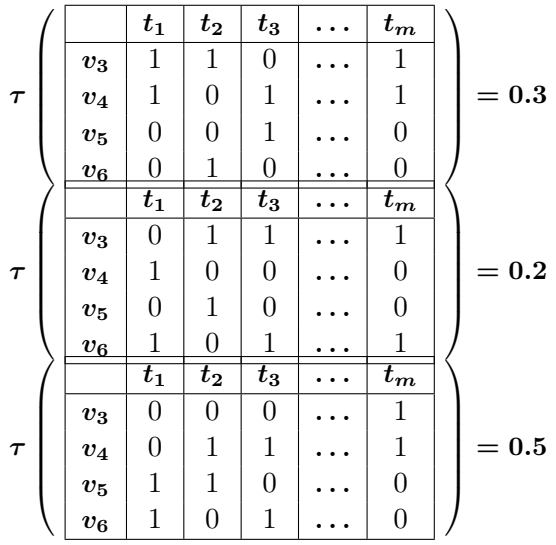


Figure 3.2: Example of temporal probabilistic k-placement

is intercepted due to the detector placement strategy. On the other hand, the attacker's uncertainty is measured as the entropy in the location of the monitors from the attacker's point of view: the higher the entropy, the higher the attacker's effort required to discover the location of monitors.

3.3.1 Minimum Interception Probability

As mentioned earlier, the attacker's objective is to exfiltrate the data segments $d_1, d_2, \ldots, d_m$ over a temporal span $\mathcal{T} = \langle t_1, t_2, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, while remaining undetected. At each time point t_i, the defender chooses a strategy, $\mathcal{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$ and samples k nodes without replacement. Let D_{t_i} denote the set of monitors at time t_i. Following the defender's placement of monitors, the attacker begins exfiltrating data segment d_i. For a chosen overlay path $\pi_o(h, C\&C)$ to transmit d_i, the traffic will traverse the corresponding exfiltration path

$\pi_e(h, C\&C) = \langle h, v_{i_1}, v_{i_2}, \ldots, v_{i_l}, C\&C\rangle$, with $h \in N$. Therefore, the probability that the attacker's exfiltration of data segment d_i is intercepted is given by:

$$interceptPr(\mathcal{D}_{t_i}, d_i, \pi_e(h, C\&C)) = 1 - \prod_{v \in \pi_e(h,C\&C) \setminus \{h,C\&C\}} \left(1 - pr_{t_i}^{v}\right) \tag{3.3}$$

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

$$\pi_e^{i^*}(h, C\&C) = \underset{\pi_e(h,C\&C)}{\operatorname{argmin}} \left(interceptPr\left(\mathcal{D}_{t_i}, d_i, \pi_e\left(h, C\&C\right)\right)\right) \tag{3.4}$$

In other words, Eq. 3.4 can be used to compute the minimum interception probability that a defender strategy $\mathcal{D}_{t_i}$ can guarantee at time t_i. Finally, an exfiltration activity is said to be *intercepted* when *any* of the m data flows is intercepted. Therefore, the minimum probability with which a strategy $\mathcal{D}_{t_i}$ intercepts an exfiltration activity is given by

$$exfInterceptPr\left(\{\mathcal{D}_{t_i}\}_{i\in[1,m]}\right) = 1 - \prod_{d_i} \left(1 - \min_{\pi_e} \left(interceptPr\left(\mathcal{D}_{t_i}, d_i, \pi_e\right)\right)\right)$$

Given a graph $G(V, E)$, the minimum interception probability of a strategy $\mathcal{D}_{t_i}$ at time t_i – that is $\min_{\pi_e} \left(interceptPr\left(\mathcal{D}_{t_i}, d_i, \pi_e\right)\right)$ – can be computed using Alg. 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 interception probability over any path $\pi_e(u, v)$ (given by Eq. 3.3) can be re-written as $1 - b^S$, where $S = \left(\sum_{x \in \pi_e(u,v)} \log_b \left(1 - pr_{t_i}^{x}\right)\right)$ 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 monitors. Therefore, each edge in E' corresponding to a node

Algorithm 1: *minimumInterceptionProb*$(G, \mathcal{D}_{t_i}, N, C\&C)$

Require: a connectivity graph $G(V, E)$, a defender strategy, $\mathcal{D}_{t_i} \sim \{pr^v_{t_i}\}$, a set $N \subseteq V$ of mission-critical nodes, a potential C&C location

Ensure: the minimum interception probability of strategy $\mathcal{D}_{t_i}$ at time t_i for graph $G(V, E)$ with respect to mission-critical nodes N and the potential 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 $\mathcal{E}(v)$ 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 $\mathcal{E}(h)$ **do**
14: **for all** c in $\mathcal{E}(C\&C)$ **do**
15: $S \leftarrow$ length of the shortest path from e to c in H
16: $interceptPr(h, e, c) \leftarrow 1 - b^S$
17: **end for**
18: **end for**
19: $interceptPr(h) \leftarrow \min_{(e,c) \in \mathcal{E}(h) \times \mathcal{E}(C\&C)} (interceptPr(h, e, c))$
20: **end for**
21: **return** $\min_{h \in N}(interceptPr(h))$

$v \in V$ 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 the edges incident on the mission-critical and C&C vertices in V. The shortest path length represents the maximum probability that the exfiltration traffic is not intercepted and the vertices in V corresponding to edges on this shortest path form the path $\pi^{i*}_e(h, C\&C)$.

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 monitor placement (Lines $3 - 10$). If a monitor 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 C&C, respectively. Finally, Line 16 computes the minimum interception probability over all the paths from a mission-critical node $h \in N$ to C&C that traverse edges e and c. Line 19 computes the minimum interception probability for each mission-critical node h by considering all the paths to C&C. Finally, the minimum interception probability with respect to all mission-critical nodes in N for graph $G(V, E)$ is computed on Line 21.

Analysis

In the worst case, Alg. 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 C&C in G, we can leverage the Fibonacci heap implementation of Dijkstra's single-source shortest path algorithm [63]. The complexity of computing the shortest path lengths for a node $h \in N$ (Lines $13-19$) is given by $O\left(\mathcal{E}(h) \cdot (|E'| + |V'| \log |V'|)\right)$. 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 interception probability is $O(|V|^6)$. However, for practical network topologies, our simulation results indicate that the running time does not exceed $O(|V|^3)$.

3.3.2 Attacker's Uncertainty

Probabilistic deployment of monitors introduces uncertainty for the attacker with respect to the location of the monitors. Depending upon the nature of the deployed monitors (either active or passive), the attacker may progressively learn the location of these monitors through

probing. For instance, if an enterprise network deploys an active IDS, a simple probing strategy could be to send malicious packets to a node suspected of hosting a monitor and depending on whether the node accepts or rejects the packets, the attacker can determine the node's monitoring state. In an ISP network, an attacker can leverage probing strategies described by Shinoda et al. [64], and by Shmatikov and Wang [65] to identify the presence of monitors in a network. Here, the attacker probes the nodes of interest with attacks whose characteristics are recognizable from the resulting alert.

The uncertainty introduced by a dynamic placement strategy can be quantified by measuring the entropy in locating the monitors 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 monitor placement. Therefore, the entropy due to a strategy $\mathcal{D}_{t_i} \sim \{pr^v_{t_i}\}_{v \in V}$ is given by:

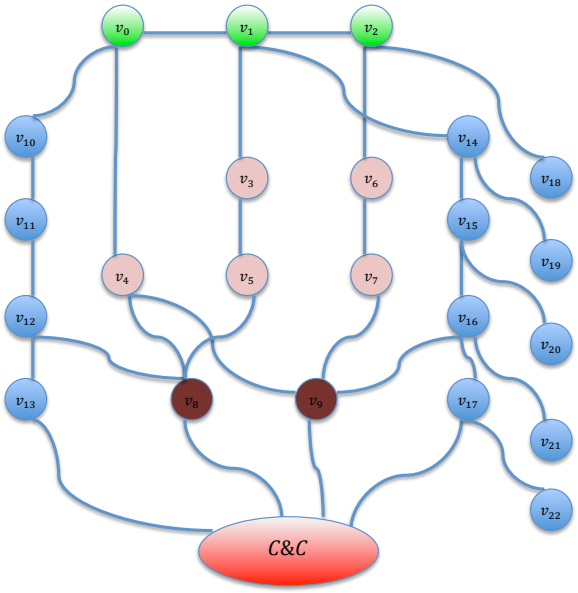
$$H(X^-_{t_i} \mid \mathcal{D}_{t_i}) = -\sum_{x \in V} P(X^-_{t_i} = x) \log(P(X^-_{t_i} = x)) \tag{3.5}$$

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

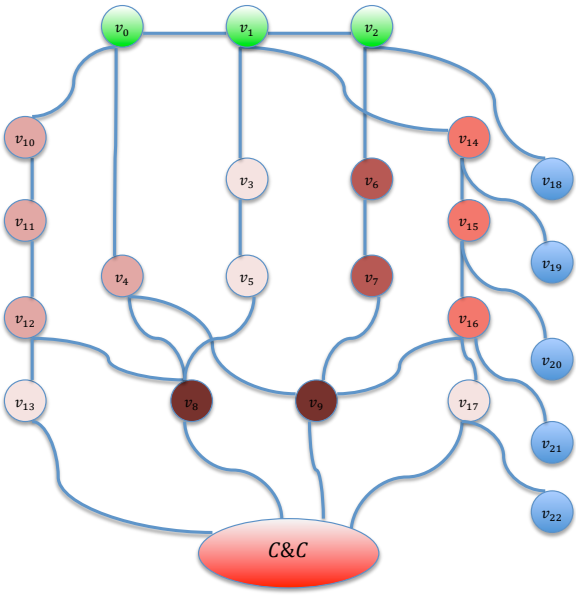
3.4 Defender's Strategies

To illustrate the effectiveness of different defender strategies, consider the network shown in Figure 3.1, which includes mission-critical nodes $N = \{v_0, v_1, v_2\}$. The attacker's objective is to relay data from any node in N to the C&C node. To protect mission-critical nodes from data exfiltration, we consider the following strategies for placing k monitors.

- *Static Iterative Centrality Placement*: In this strategy, the defender chooses nodes based on the iterative mission-betweenness centrality algorithm proposed in [62]. The defender first computes the mission-betweenness centrality of a node v as $C_M(v) = \sum_{(s,t) \in N \times C\&C \; s.t. \; v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$, 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



(a) Centrality-Weighted Strategy



(b) Expanded Centrality-Weighted Strategy

Figure 3.3: Candidate detector locations for the network of Figure 3.1, based on the (a) centrality-weighted strategy, and (b) expanded centrality-weighted strategy

mission-betweenness centrality for all the nodes, the defender chooses the node with the highest centrality for monitor placement. For each subsequent monitor placement, the centrality $C_M(v)$ of all non-monitor nodes is re-computed and the node with the highest centrality among the non-detector nodes is picked for placing the next monitor. In the example of Figure 3.1, assume that the defender can place $k = 2$ monitors. Then, node v_9 (or v_8) will be chosen to the place the first monitor followed by v_8 (or v_9) to place the second monitor.

- *Uniform Random Placement*: The static nature of the above strategy enables an attacker to pre-compute the location of monitors and compromise nodes along a monitor-free path to exfiltrate data. Therefore, in order to create uncertainty about the exact location of monitors, in this strategy, the defender chooses k nodes to place monitors uniformly at random.

- *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 interception probability. In this strategy, to improve interception guarantees, the defender places k-monitors 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 monitor placement. For the example shown in Figure 3.1, the nodes shaded in brown in Figure 3.3a are the only nodes considered for monitor placement by this strategy (the color intensity 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, the strategy considers only the nodes on the set of all shortest paths between the mission-critical nodes and C&C. Therefore, in this strategy, the coverage of the exfiltration

surface is expanded by considering all the nodes in the paths that are 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 $C_E(v) = \sum_{(s,t)\in N\times C\&C \text{ s.t. } v\neq t} \frac{\sigma'_{st}(v)}{\sigma'_{st}}$, 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 Figure 3.1, when $\delta = 0.25$, the nodes shaded with different intensities of brown in Figure 3.3b will be considered for randomizing the placement.

3.5 Simulation Results

We evaluated the proposed strategies using real ISP network topologies obtained from the Rocketfuel dataset [1] 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 3.1 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, in our simulations, we considered a worst-case scenario and assumed that all the external routers are potentially routing traffic to C&C servers.

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 [1]. In order to accurately capture the connectivity of an ISP network at the router level, the BRITE network topology generator [66] was used to generate these networks. Ten such networks were considered, with mission-critical nodes varying between 10%-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 monitor set

Table 3.1: Summary of ISP networks from [1]

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

as a fraction of the number of mission-critical nodes while, for synthetic topologies, we varied the size of the monitor 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 (either proportional to the number of nodes that need to be protected or 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 presented results using paired t-test at 95% confidence interval. Finally, to illustrate the performance of the proposed strategies, we show the results for a subset of the topologies from the Rocketfuel dataset. We observed a similar trend in the performance in the remaining networks.

3.5.1 Minimum Interception Probability

As illustrated in Figure 3.4 and Figure 3.5, the probability of intercepting exfiltration traffic increases linearly with the number of monitors. We observed that variations in the interception probability for different synthetic networks were less than 1% and hence, for sake of presentation, we only show mean values.

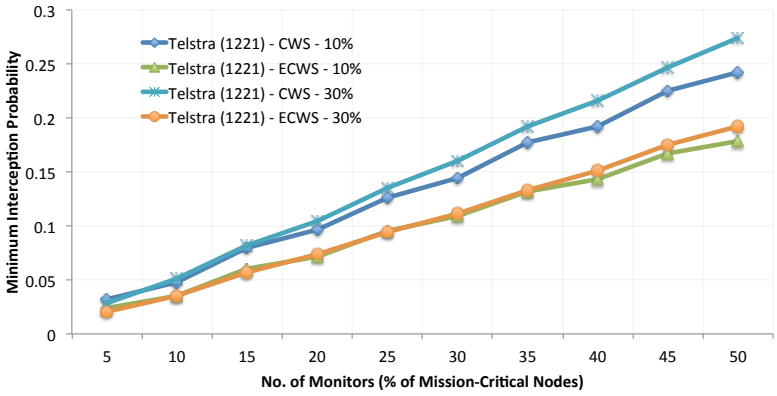
It can be observed that, independently of the number of mission-critical nodes and monitors, the centrality-weighted strategy outperforms the expanded centrality-weighted strategy. This trend can be attributed to the scale-free nature of the topology in which most of the paths traverse a small portion of the nodes. As the expanded strategy considers a larger sample space 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 intercept the exfiltration of the data segments in any of the networks as there was at least one monitor-free path between one of the mission-critical nodes and a C&C location.

3.5.2 Attacker's Uncertainty

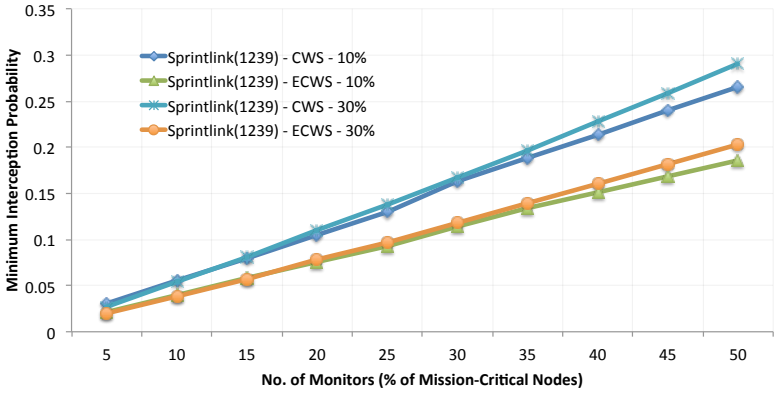
As mentioned earlier, among the monitor placement strategies, the static iterative centrality strategy does not introduce any uncertainty whilst the uniform random strategy introduces the highest uncertainty in the location of monitors to an attacker. To study the attacker's uncertainty in the location of monitors due to the proposed strategies, we computed the relative entropy introduced by the centrality-weighted and the expanded centrality-weighted strategy w.r.t. the uniform random strategy. As shown in Figure 3.6, the centrality-weighted strategy and its expanded version create an uncertainty that lies in-between the two ends of the entropy spectrum. In particular, as expanded strategy potentially considers more nodes, the number of combinations of monitor 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 potentially trades-off entropy for interception probability guarantees.

3.5.3 Running Time

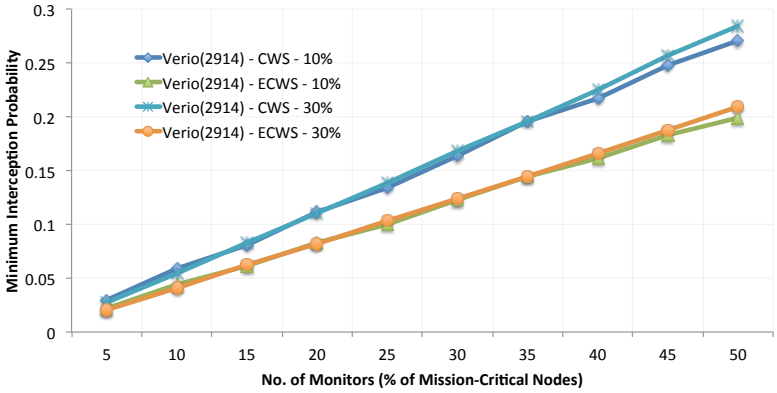
In this section, we evaluate the performance of Algorithm 1 in computing the minimum interception probability and the performance of various monitor placement strategies as a



(a) Telstra



(b) Sprintlink



(c) Verio

Figure 3.4: Minimum interception probability for different networks using centrality-weighted (CWS) and expanded centrality-weighted (ECWS) strategies

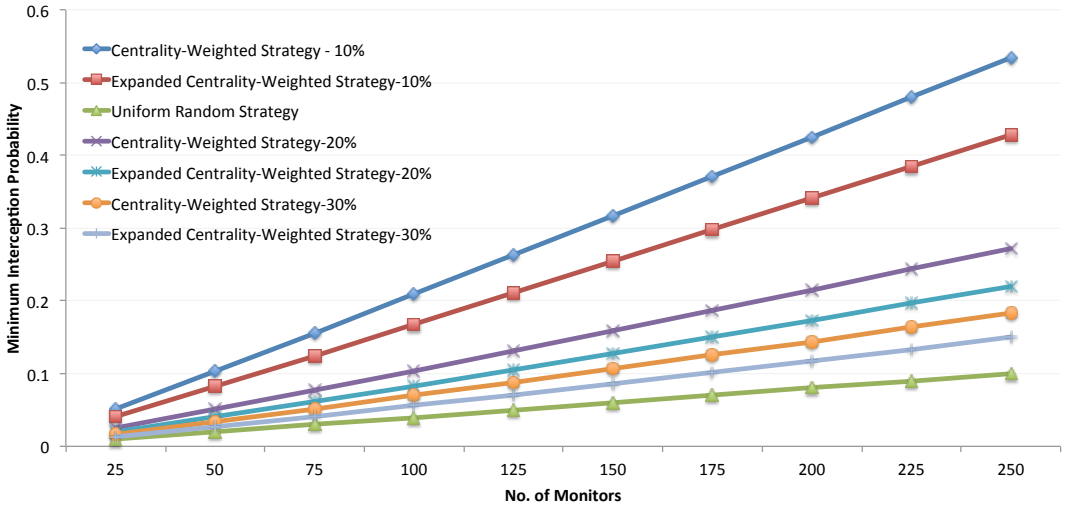
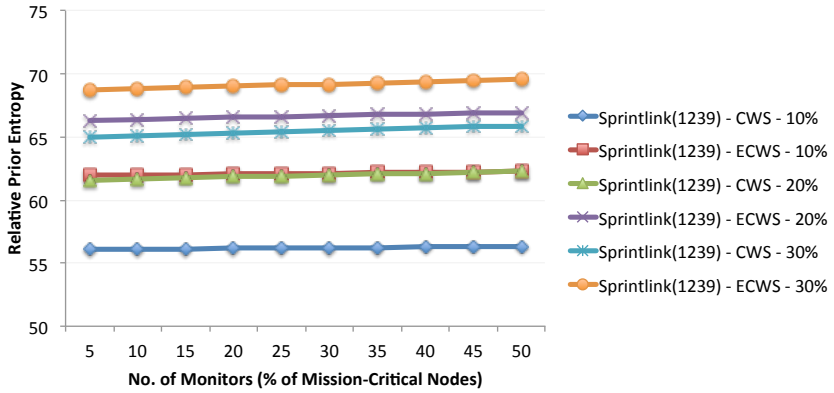


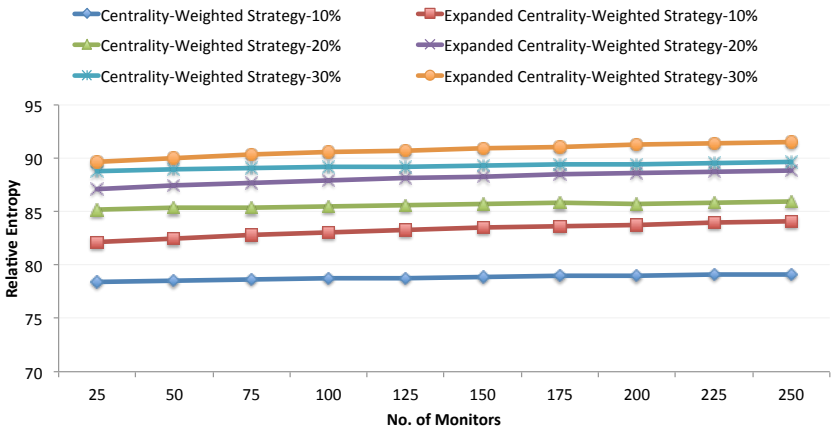
Figure 3.5: Minimum interception probability for different strategies using synthetic topologies

function of the network size. For each network size, we generated 10 ISP-type topologies, with 10% of network size as mission-critical nodes and 500 C&C locations chosen at random. We varied the number of monitors from 3% to 7% of network size and observed similar trend in the running time. For the sake of presentation, we present the results with 3% of network size for the total number of available monitors. The runtime was averaged over the 10 graph settings for each network size. All experiments were conducted on an AMD Opteron processor with 4GB memory running Ubuntu 12.04.

Although, in theory, the worst-case running time of Algorithm 1 is $O(|V|^6)$, for practical network settings, it can be seen (line marked in orange in Figure 3.7) that the execution time grows as $O(|V|^3)$ with an R^2 value of 0.99. Finally, as shown in Figure 3.8, the dynamic strategy computes the monitor 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 monitors.



(a)



(b)

Figure 3.6: Relative increase in entropy for the attacker introduced by different strategies for (a) Sprintlink network and (b) synthetic network

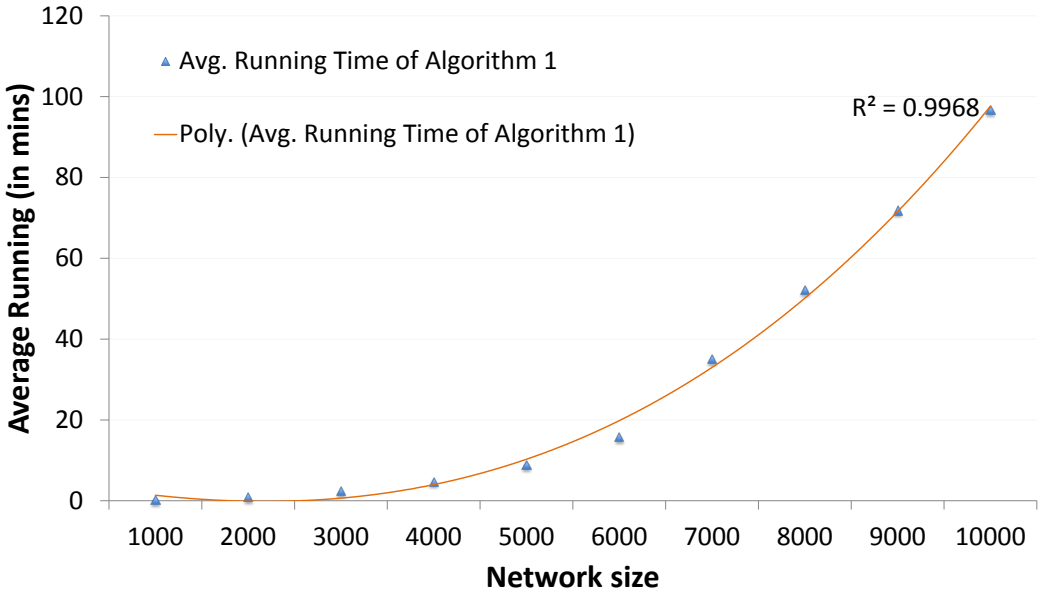


Figure 3.7: Running time for computing the minimum interception probability using Algorithm 1.

3.6 Conclusions

In this chapter, we addressed the challenge of developing a monitoring strategy to detect stealthy botnets that exfiltrate data from a network. To this end, we have proposed a moving target defense approach to dynamically and continuously changing the location of monitors 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 likelihood of detection by creating uncertainty about the location of monitors and also, increase attacker's effort by forcing the botmaster to perform additional actions in an attempt to create monitor-free paths through the network. To evaluate the proposed strategies,we have presented two metrics – namely the minimum interception probability and the attacker's uncertainty – and an algorithm to compute the minimum interception probability. We validated our approach

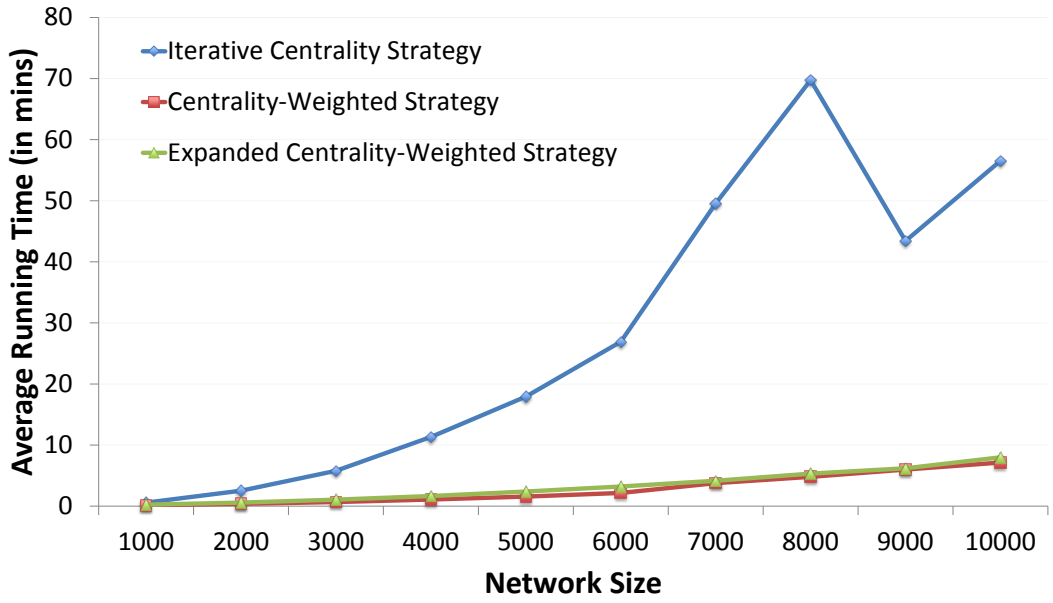


Figure 3.8: Running time for different monitor placement strategies

through simulations, and the results confirmed that the proposed solution can effectively reduce the likelihood of successful exfiltration campaigns.

Chapter 4: DeBot: A Mechanism to Detect Stealthy Botnets

4.1 Introduction

In this chapter, we propose a mechanism, namely DeBot, to detect architectural stealthy botnets by analyzing the traffic generated by different hosts within the network and subsequently, identify suspicious flows. To this end, we first leverage the monitoring strategy proposed in Chapter 3 to monitor and intercept traffic from different network segments. Additionally, we also leverage two intrinsic characteristics of botnets: (i) the difference in the traffic flow statistics between bots and benign applications and (ii) the periodicity in the connection patterns between a bot and its peers to identify potential bot flows.

4.2 Overview and Architecture

In our work, we assume that the attacker is aware of the network topology and can compromise hosts and servers within the network and setup a communication architecture to exfiltrate data to an attacker-controlled server outside the network.

The proposed approach to detect exfiltration by architectural-stealthy botnets can be divided into four phases: *preprocessing*, *observation*, *refinement*, and *analysis*. Fig. 4.1 provides an overview of the detection mechanism. First, in the *preprocessing phase*, DeBot computes the rate at which traffic snapshots should be captured at different monitoring points in order to enable subsequent analysis. Through the chapter, we refer to this rate as the *snapshot rate*.

In DeBot, the monitoring period T (say, 24 hours) is divided into smaller epochs, Δt (say, 30 mins). At each epoch in the *observation phase*, DeBot randomly chooses a monitoring point based on the snapshot rates and inspects traffic traversing it during that period.

Throughout the monitoring period, DeBot maintains a score for each host in the network and updates it 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 similarity score for each host based on $\frac{T}{\Delta t}$ snapshots of traffic observed from different vantage points in the network. The aggregated score is then used to identify suspicious hosts H_B by comparing the similarity score of each host with the similarity score of other hosts within its neighborhood.

In the *refinement phase*, DeBot identifies the set of flows that correspond to bot traffic by exploiting the periodic communication pattern between bots. For each host in H_B, DeBot uses periodogram analysis to identify flows that are relatively more periodic than other flows generated by it and marks them as suspicious for further inspection.

Finally, to confirm the existence of malicious activity, in the *analysis phase*, the suspicious flows can be further analyzed using a fine-grained analysis techniques such as Deep Packet Inspection.

4.3 Preprocessing Phase

As mentioned above, DeBot analyzes flow records from different monitoring points in order to identify malicious flows. The performance of the proposed detection mechanism is influenced by the volume of malicious flows intercepted in the analyzed traffic. To increase the likelihood of intercepting malicious flows, two challenges emerge: (i) identifying potential traffic monitoring locations in the network (Section 4.3.1); and (ii) developing a sampling technique to collect malicious flows over time and tackle scalability issues (Section 4.3.2).

4.3.1 Information Flow

The objective of an exfiltration campaign is to continually transfer sensitive data to a remote server controlled by the attacker. Typically, in an enterprise network, sensitive data is confined to a few servers which we refer to as *mission-critical* servers. The exfiltrated data traverses several intermediate forwarding devices such as switches, routers and gateways within the network before reaching the remote server. Any of these intermediate devices

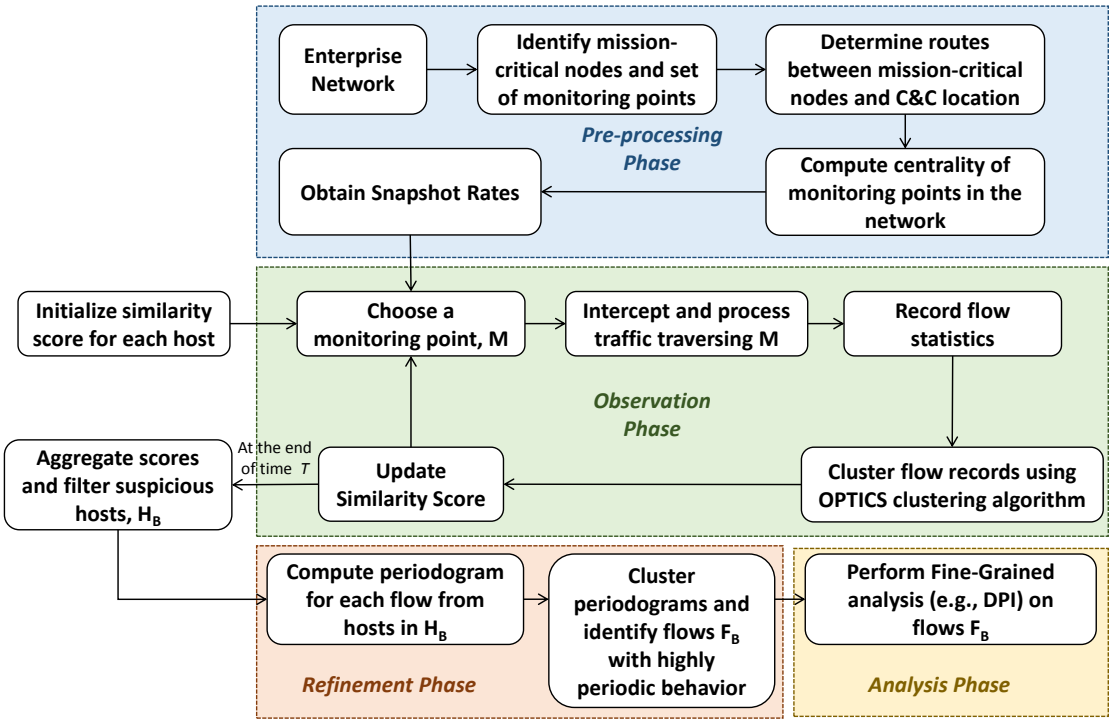


Figure 4.1: Overview of DeBot

can be used as a monitoring point. In the proposed detection scheme, traffic is mirrored from these intermediate devices to a central location for analysis. In large enterprises, such an infrastructure is already usually in place and used to remotely monitor the performance of the devices.

Given the sparseness of malicious flows, it is critical to identify monitoring points that are well-positioned to capture malicious flows. For instance, consider the enterprise network in Fig. 4.2 with the sensitive data hosted in the file servers located in Subnet-1 and Subnet-2. The file servers host redundant copies of the data. To exfiltrate this sensitive data, an attacker could choose from one of several botnet communication architectures with varying

degrees of exposure of malicious flows to the monitoring points. For example, an attacker could compromise one of the client systems in Subnet-1, say h_1 with IP 192.168.1.2, mount the share drive and directly transfer the sensitive data to a remote location. 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 exposition, we denote this communication architecture as a path: $h_1 \rightarrow M_6 \rightarrow M_1 \rightarrow C\&C$, where $C\&C$ denotes the attacker-controlled external server. Other possible exfiltration paths include, but not limited to: $g_x \rightarrow M_7 \rightarrow M_6 \rightarrow M_1 \rightarrow C\&C$ where g_x is a compromised host in Subnet-2, $h_x \rightarrow M_6 \rightarrow M_7 \rightarrow M_5 \rightarrow M_4 \rightarrow Db \rightarrow M_4 \rightarrow M_2 \rightarrow C\&C$, in which a database server (Db) was compromised and used as an intermediate relay bot to forward data outside the network. It can be observed that the percentage of malicious flows intercepted by different monitoring points depend on the internal routing policy and the attacker's choice of communication architecture.

4.3.2 Intercepting Malicious Flows

Analyzing traffic traversing all monitoring points calls for a detection mechanism that can scale with large volumes of traffic. Although capturing traffic from all monitoring points would ensure that all the exfiltration flows are intercepted, the resulting traffic volume that need to processed at the central location would be too large. Therefore, it is crucial to identify the most effective monitoring points to limit the amount of data centrally processed by the detection mechanism, whilst ensuring that a sufficient number of malicious flows are intercepted for analysis. When not enough malicious flows are intercepted, the detection mechanism may not have enough information 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 example enterprise network scenario of Fig. 4.2 in the CyberVAN testbed [67] – a testbed which has the capability to generate benign user traffic[1]. In the network of Fig. 4.2, we consider a stealthy botnet whose communication

[1]More details about CyberVAN are provided in Section 4.6.

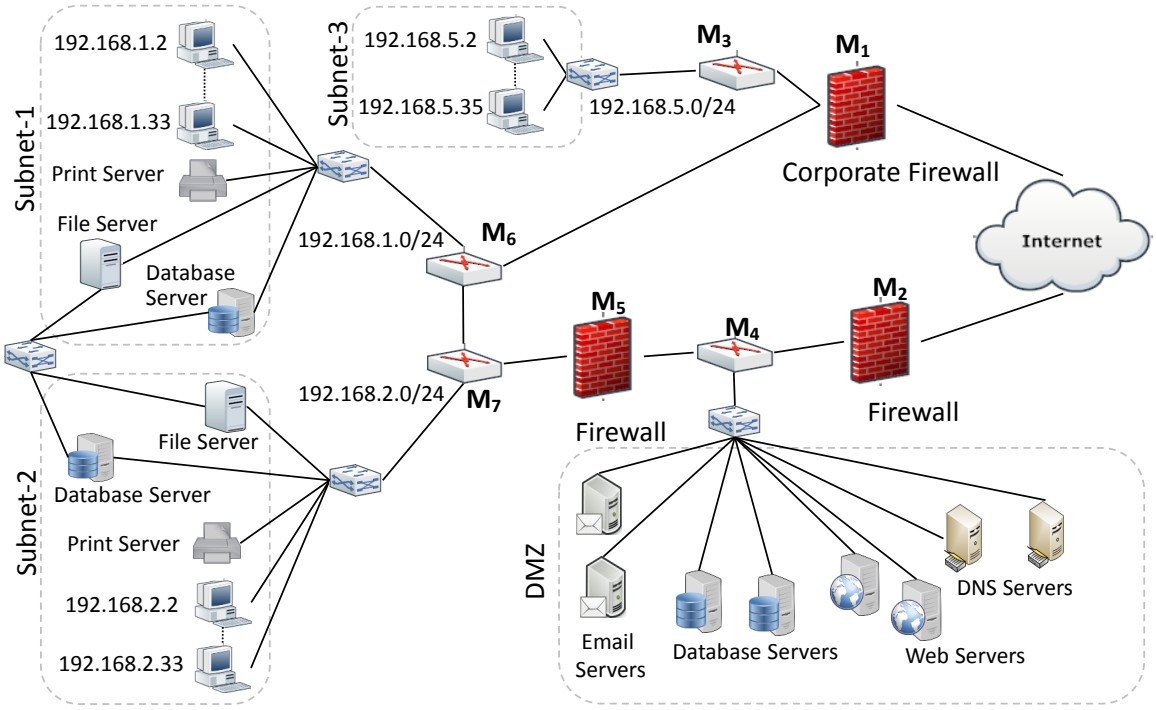


Figure 4.2: Enterprise Network

architecture is composed of a server in the DMZ and two compromised hosts, one in Subnet-1 and one 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 an external server on the Internet.

Table 4.1 shows the number of flows intercepted at different monitoring points in the example enterprise network during a 12 hour monitoring period. From the table, it can be seen that, if the monitoring point M_4 is chosen, then the detection mechanism would process 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.

Table 4.1: Percentage of bot flows intercepted at different monitoring points during a 12 hour observation period

Monitoring Point	**No. of Unique Flows**	**% of Bot Flows**
M_1	$5,248$	0
M_2	$149,392$	2.31
M_3	$106,448$	1.62
M_4	$913,680$	0.85
M_5	$690,748$	0.63
M_6	$126,156$	1.37
M_7	$149,580$	2.31
Total	$1,146,784$	0.97

To improve the chances of intercepting malicious flows in a resource-constrained environment, we leverage our work on dynamic monitoring strategies. To recall, we model the network as graph $G(V, E)$ – with a subset of nodes M_c identified as mission-critical and another subset of nodes representing $C\&C$ sites. For each potential monitoring point $m \in M \subset V$ in the network, we 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 $C\&C$ locations that traverse m. The entire time horizon is divided into smaller observation epochs and in each epoch a monitoring point m is chosen with probability $\frac{C_M(m)}{\sum_{m' \in M} C_M(m')}$. The proposed strategy only considered monitoring points on the shortest path. Therefore, to improve coverage, we proposed an alternative strategy – the expanded centrality-weighted strategy – which, in addition to considering monitoring points on shortest paths, 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 monitoring strategy mentioned above – i.e., choosing monitoring points with high centrality – and modify it by in order

to take into account internal routing policies of the network. In Chapter 3, we assumed that traffic between systems is routed through the shortest path. However, in an enterprise, the 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, network security policies etc. In our work, 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 $\mathcal{R}$ to denote the set of all routes between all systems within a network.

As mentioned earlier, stealthy botnets reduce exposure to detectors by compromising additional systems and use them as proxy servers to relay messages to a destination outside the network. In order to model this, we first determine the set of all systems that could act as proxies. In an enterprise network, most communication patterns follow a client-server model in which an end user requests a service from a server. Hence, to avoid suspicious communication patterns, compromised servers (such as Email, DNS etc.) could take the role of a proxy servers for bots within the network. However, in our work, we make a conservative assumption that all systems (both client machines and servers) can act as proxy servers for the botnet. 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.

We use algorithm *computeSnapshotRates* (Algorithm 2) to compute the snapshot rates for each monitoring point. For each mission-critical node m_c and potential proxy server s, the algorithm first determines all the paths through s by concatenating routes from m to s, i.e., $\mathcal{R}_{m,s}$, 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 $\mathcal{R}'_m$ is used for computing the mission-betweenness centrality for the monitoring points (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 time horizon, this is a one-time computation.

The snapshot rate of a monitoring point, $m \in \mathcal{M}$, is the probability that m is chosen by

Algorithm 2: *computeSnapshotRates*$(M, M_c, S, \mathcal{R})$

Require: 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

Ensure: the snapshot rate $P(m)$ for each monitoring point $m \in M$

for all $m_c \in M_c$ **do**
  $\mathcal{R}'_{m_c} \leftarrow \emptyset$
  for all $s \in S$ **do**
    $\mathcal{R}'_{m_c} \leftarrow \mathcal{R}'_{m_c} \cup \{R_1 || R_2 \mid (R_1, R_2) \in \mathcal{R}_{m,s} \times \mathcal{R}_{s,C\&C}\}$
  end for
end for
$C_B(m) \leftarrow 0, \forall m \in M$ {Initialize mission-betweenness centrality}
for all m_c in M_c **do**
  $\sigma(m) \leftarrow 0, \forall m \in M$
  for all $R \in \mathcal{R}'_{m_c}$ **do**
    for all $m \in M \cap R$ **do**
      $\sigma(m) \leftarrow \sigma(m) + 1$
    end for
  end for
  $C_B(m) \leftarrow C_B(m) + \frac{\sigma(m)}{|\mathcal{R}'_{m_c}|}, \forall m \in M$
end for
$P(m) \leftarrow \frac{C_B(m)}{\sum_{m' \in M} C_B(m')}, \forall m \in M$
return P

the detection mechanism for analyzing the traffic traversing it during an observation epoch. Choosing monitoring points randomly introduces uncertainty to the attacker and increases the cost and complexity of establishing a stealthy botnet communication architecture.

4.4 Observation Phase

DeBot identifies suspicious hosts by examining the network characteristics of hosts and comparing them 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 a malicious activity. Hosts whose characteristics deviate from their neighboring hosts are classified as suspicious. In this paper, without loss of generality, we assume the neighborhood of a host to be the set of hosts that exist in the same subnet.

Prior to the start of this phase, DeBot initializes the similarity scores of host pairs within the network. The similarity score quantifies the similarity in the network behavior of two hosts. At the beginning of each observation epoch $\Delta t_i, i \in \left[1, \frac{T}{\Delta t}\right]$, the detection

mechanism determines the monitoring point m_i based on the snapshot rates determined 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). The collected 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.

4.4.1 Extracting Flow Statistics and Clustering Flows

For each flow f, DeBot records the median number of packets sent ($pkts_{sent}(f)$), packets received ($pkts_{recv}(f)$), bytes sent ($bytes_{sent}(f)$) and bytes received ($bytes_{recv}(f)$) during the 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, the traffic snapshot may include incomplete sessions (SYN/SYN-ACK packets and/or FIN/FIN-ACK packets might not have been intercepted during the observation window). In such cases, a TCP session is included into the flow record table $\mathcal{F}_i$ if at least one of the packets and its corresponding acknowledgment is intercepted during the same epoch. For UDP packets, flows in which a request is followed by a response are only considered.

To identify flows that exhibit similar network behavior, the flow records in $\mathcal{F}_i$ are clustered using the OPTICS clustering algorithm. OPTICS [69] is a density-based clustering algorithm that can identify arbitrary shaped clusters (unlike K-means) 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) there are at least $minPts$ records in it 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 ϵ-distance. Records that do not belong to any of the clusters are labeled as noise.

As DeBot operates on traffic snapshots, selecting an appropriate value for the parameter $minPts$ is crucial to ensure that the intercepted bot flows form a cluster and not ignored as noise. If $minPts$ is set very high, the traffic snapshot might not have intercepted sufficient number of bot flows to form a cluster. However, a low $minPts$ will lead to creation of several clusters. Therefore, the choice of $minPts$ is influenced of both the frequency of bot communication (ν) and the length of the observation window (Δt). 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.

4.4.2 Update Similarity Score

As mentioned earlier, DeBot tracks the similarity in the network characteristics between hosts and is modeled as follows: Let $\mathcal{N}(h)$ denote the set of hosts in the neighborhood of host, h and let, a scoring function $sim\_score(h_i, h_j)$ quantify the similarity between the hosts h_i and h_j. Before the commencement of the observation phase, the scoring function is initialized as: $sim\_score(h_i, h_j) = 1, \forall h_j \in \mathcal{N}(h_i)$. As noted earlier, in our work all hosts that are in the same subnet as host h is considered as its neighborhood.

Let $\mathcal{C}_k$ denote the set of flow clusters at the end of an observation epoch, Δt_k and let $C_{h_i} \subseteq \mathcal{C}_k$ be the subset of clusters containing flows from/to host h_i. The scoring function is updated as follows:

$$sim\_score(h_i, h_j) = \lambda(m_k, h_i) \cdot \left(\frac{|C_{h_i} \cap C_{h_j}|}{|C_{h_i} \cup C_{h_j}|} \right) + (1 - \lambda(m_k, h_i)) \cdot sim\_score(h_i, h_j) \qquad (4.1)$$

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 always traverses the monitoring point m. For instance, in the enterprise shown in Fig. 4.2, the visibility of monitoring point 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:

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

4.4.3 Suspicious Host Identification

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

4.5 Refinement Phase

A bot that participates in an exfiltration campaign regularly communicates with its peer bots or C&C to send (or receive) updates. Such periodic exchange of messages eases the management of the C&C architecture for the bot master. For instance, Table 4.2 provides the communication frequency of different POS malwares. From the table, it can be seen that the periodicity with which bots communicate can vary depending on the malware.

Table 4.2: Communication frequency of different POS malwares

POS Malware	**Victim Institute**	**Period**
BlackPOS, [16]	Target	10 mins
FrameworkPOS, [16]	Home Depot	60 mins + random mins
Backoff, [16]	UPS	45 secs
Punkey, [70]	Suspected for CiCi's Pizza	20 mins or 45 mins

Such periodic behavior has been observed in existing botnets that are known for stealing credentials such as Storm, Waldec and Zeus [33].

In DeBot, we leverage the periodic communication nature of bots to identify host pairs that are malicious. To determine whether a host h_i is periodically communicating with another host h_j, the connection 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 PSD of remaining connections generated by host h_i to identify periodic communication pairs. The following sections provide more details for each step in this procedure

4.5.1 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 time horizon $[0, T]$ is divided into equally-spaced times, $T_{i,j} = \{t_1, t_2, ..., t_N\}$, 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 a frequency domain data by expressing the signal as a sum of sinusoidal components using the equation:

$$F_{h_i,h_j}(\omega) = \sum_{k=1}^{N} X_{h_i,h_j}(t_k) e^{-i\omega t_k} \tag{4.2}$$

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 – higher the coefficient value, 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 [71]. In this work, we use the periodogram method as it is computationally less expensive compared to other methods. The periodogram of the time series X_{h_i,h_j} is given by:

$$P_{h_i,h_j}(\omega) = \frac{1}{N} |F_{h_i,h_j}(\omega)|^2 = \frac{1}{N} \left[\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) cos\ \omega t_k \right)^2 + \left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) sin\ \omega t_k \right)^2 \right] \tag{4.3}$$

Fig. 4.3 depicts the periodogram of a sample of Zeus traffic obtained from a public repository [72]. In this network trace, the bot connected with its peer bot every 60 seconds – which, in the periodogram, is characterized by the frequency corresponding to highest power.

Employing the above approach directly in the proposed mechanism, however, suffers from several limitations:

- *Unevenly-spaced observations*: Analyzing the periodogram of a connection pattern

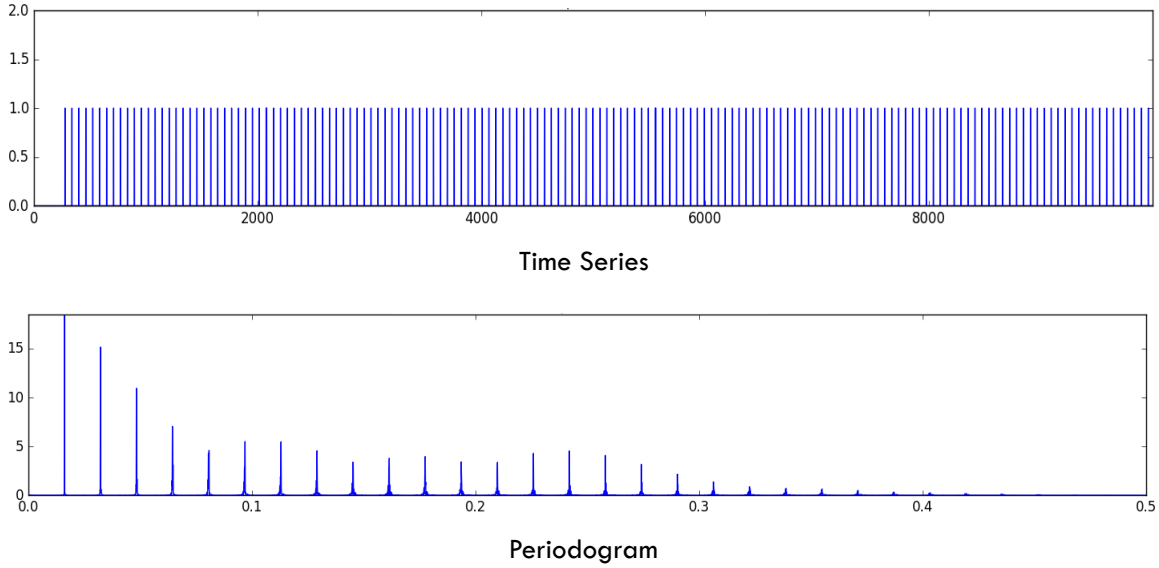


Figure 4.3: Communication pattern and corresponding periodogram of a Zeus bot

between two hosts using Eq. 4.3 assumes that the traffic was sampled at equally-spaced time intervals. However, DeBot employs a dynamic monitoring strategy which provides only snapshots of traffic from different monitoring points. Thus, there could be long periods of unobserved connection patterns between a pair of hosts. In such cases, the above periodogram analysis will 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 periodogram will lead to 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 limitations presented by the classical periodogram approach, the detection mechanism should be robust to the following:

- *Random perturbations*: As malicious flows are detected based on the periodic exchange of messages, bots can evade detection by adding random perturbations to the connection pattern.
- *Legitimate applications*: Legitimate applications such as software updates and email clients also connect to servers on a regular basis. Misclassifying the corresponding flows as malicious will lead to high false positive rates.

The above challenges are addressed in the following subsections.

4.5.2 Lomb-Scargle Periodogoram

The dynamic monitoring strategy samples the traffic between two hosts h_i, h_j irregularly. This results in a signal $X_{h_i,h_j}(t_k)$ that is defined only for times when a monitoring point could intercept traffic from h_i to h_j. In other words, the times $t_k, k = 1, 2, ..N$ are arbitrarily spaced. To study the periodicity of an unevenly-spaced discrete time series, we propose the use of Lomb-Scargle periodogram to estimate the power spectrum [73]. The Lomb-Scargle periodogram modifies the classical periodogram given in Eq. 4.3 by introducing a time translation parameter τ:

$$P_{h_i,h_j}(\omega) = \frac{1}{2}\left[\frac{\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) cos\ \omega(t_k-\tau)\right)^2}{\sum_{k=1}^{N} cos^2\omega(t_k-\tau)} + \frac{\left(\sum_{k=1}^{N} X_{h_i,h_j}(t_k) sin\ \omega(t_k-\tau)\right)^2}{\sum_{k=1}^{N} sin^2\omega(t_k-\tau)}\right] \tag{4.4}$$

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 from obtained from Eq. 4.4 is shown to be statistically equivalent to the least squares fit of a sinusoidal wave applied to the discrete time series [73].

High peaks in the resulting periodogram is 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 a result of noise artifact), 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 $\mathcal{N}(\mu, \sigma)$ and determines a threshold power-level z_0 below which a power is considered to be generated due to pure noise [74]. Thus, if there are no frequencies whose power is greater than the threshold power-level z_0, then the signal is considered to be non-periodic. One of the limitations of the significance tests is that the power threshold determined by the significance test is sensitive to the choice of parameters μ, σ for the Gaussian distribution [74]. Furthermore, existing non-parametric methods [75] is applicable for evenly-spaced time series and thus, cannot be directly adopted for our setting.

4.5.3 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 the connection pattern is *relatively*-periodic by comparing the corresponding periodogram with that of the connection patterns generated by the host h_i during the monitoring period. Such a modification will use the system's typical network behavior (instead of white noise) as the baseline to check for periodicity.

Let $\mathcal{P}_{h_i} = \{P_{h_i,h_j}(\omega)\}$ be the set of periodograms of connection pattern originating from host h_i (obtained using Eq. 4.4). To determine which periodogram exhibits an anomalously higher periodicity, the periodograms in the set $\mathcal{P}_{h_i}$ is clustered using agglomerative clustering method. While clustering, the difference in periodic structures between two periodograms is obtained using the power distance metric [75]. 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 the periodogram 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 $K = 1000$. Before computing the $pDist$, to ensure that the total energy is constant, the powers are normalized as follows:

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

In the proposed hierarchical cluster analysis of periodgorams, the linkage criteria between sets of periodograms was computed using the Ward's method. After building a hierarchical structure of the periodograms, clusters were formed by the set of periodograms whose pairwise distance is less than a threshold distance, 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, then the connection pattern of the corresponding host pair is considered to relatively-periodic. The rationale behind this approach is that connection patterns corresponding to the bot flows are anomalously more periodic than other connection patterns from the same hosts and hence, 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, the flows generated by the host pairs that were marked as suspiciously periodic can be analyzed using fine-grained tools such as Deep Packet Inspection or can be submitted for manual inspection to the security operations center.

4.6 Evaluation

We evaluated the performance of DeBot in a testbed environment against several stealthy botnet communication architectures.

4.6.1 Environment

Testbed. All the experiments were conducted in the Cyber Virtual Assured Network (CyberVAN) testbed [67]. CyberVAN is a state-of-the-art testbed designed to provide a realistic network environment to test and evaluate cyber security models. In CyberVAN, applications run on virtual machines and the network traffic generated by the host machines is forwarded to the destination through a simulated network. CyberVAN employs the ns-2 network simulator to simulate the underlying network topology and maps host machines to nodes in the simulated network. To simulate the behavior of packets traversing the real network, CyberVAN intercepts packets from the source machine and injects them into the corresponding node of the simulated network. The injected packets traverse the simulated network and are forwarded to the destination machine after they arrive at the corresponding destination node. The use of simulated networks enables a high-fidelity reproduction of network effects such as propagation delays and packet loss. Furthermore, the virtual machines are time synchronized with a discrete event simulator to ensure that the virtual machine's real time does not advance faster than the simulator time.

Network Topology and Traffic In this work, we considered the enterprise network shown in Fig. 4.2 to study the performance of the proposed detection mechanism. The network is composed of 114 machines with 98 client machines and 16 servers that include a range of services that are present in typical enterprise networks, such as DNS, database, email, web, FTP and printer services. Traffic generated by this network is intercepted at 7 monitoring points $(M_1, ..., M_7)$ that include routers and firewalls. Here, routers M_3, M_4, M_6, M_7 intercept traffic between machines within the subnets they are interfaced with, as well as traffic entering or exiting those subnetworks and traversing one of those routers. In this network scenario, we assumed the file servers in Subnet-1 and Subnet-2 to be mission-critical hosts and the C&C server to be located on the Internet. For this setting, the snapshot rates for different monitoring points were computed using Algorithm 2 and their values are reported in Table 4.3.

In this network, the benign user traffic is generated using ConsoleUser – a commercial tool

Table 4.3: Centrality-based snapshot rates for all the monitoring points

Monitoring point	**Snapshot Rate**
M_1	0.311
M_2	0.025
M_3	0.098
M_4	0.025
M_5	0.025
M_6	0.319
M_7	0.197

developed by Skaion to mimic a human user's interaction with network-based applications. The tool models a user's behavior as a Markov model in which states correspond to the state of the network application and the application transitions from one state to another when the user generates an event. For example, in the case of an email application, the state space includes reading an email (R), downloading an attachment (D), or forwarding an email (F). The application in a state (say, R) transitions to another state (say, D) when the user performs an action (click download button). The ConsoleUser is scripted by assigning probabilistic values to the state transitions, which are determined based on existing studies.

Botnet Traffic. Existing botnet detection systems are evaluated by observing their performance against botnet traces that were obtained either from public repositories or captured by executing the binary in a honeypot [18–20]. Typically, a botnet detection mechanism is evaluated by first capturing background user traffic from a network (either academia or enterprise) and then combining it with a botnet trace such that the IP addresses of the hosts in the trace are mapped to either an address from the IP space of the target network or to an host that is not present in the network [76]. However, this method – known as the overlay method – does not provide us with the flexibility of evaluating a detection mechanism against stealthy communication architectures. For instance, in these botnet

traces, bots communicate with several peers outside the network. In such a case, developing a stealthy communication architecture using the overlay method would require dropping packets to a subset of peers and thus may reduce the realism of the trace. Furthermore, accounting for bot behaviors that may adapt to the proposed detection mechanism would require introducing modifications to the network characteristics such as frequency of update message exchanges.

Ideally, DeBot should be evaluated by modifying the source code of existing bots to control the number of peers within the network (thereby, maintaining stealth) and augmenting additional capabilities such as aggregating data before forwarding it to a destination outside the network. Due to the lack of publicly available source code for different bots, we used a botnet that was developed by Applied Communication Sciences (ACS). In this botnet, two types of bots were designed: *data stealing* bots and *aggregation bots*. Data stealing bots were deployed on client machines in the subnets hosting mission-critical nodes (Subnet-1, Subnet-2). These bots first check if the network share drive was mounted and if so, exfiltrate files from that folder to their peer aggregation bot. The aggregation bots collected data and update messages from their peer bots and forwarded it the C&C located on the Internet. The flow characteristics of the bots were chosen to be similar to that of the Zeus bot [77] and each bot communicated with its peers at times randomly chosen between 30 and 60 seconds.

4.6.2 Scenarios

In this section, we describe the different botnet scenarios that were considered for the synthetic enterprise network. Each botnet scenario differs in the amount of malicious traffic exposed to the different monitoring points.

1. **Aggregation Point I (BlackPOS Malware)**. In this scenario, we considered two hosts in Subnet-3 and Subnet-2 respectively as data stealing bots and an aggregation bot as a server located in the DMZ. This scenario was inspired by the BlackPOS malware variant [16]. The aggregation bot accumulated the data from all the data

stealing bots and relayed the data to an attacker-controlled server located outside the network.

2. **Botnet Without Stealth**. This scenario captures the communication architecture of several existing botnets such as Zeus and Storm. In this scenario, the compromised hosts directly communicate with their peers outside the network. For this scenario, we considered two hosts in Subnet-1 and Subnet-2 as data stealing bots for generating the malicious traffic.

3. **Aggregation Point - II**. This scenario is similar to the *Aggregation Point I* scenario, with the exception of the location of the data stealing bots and the aggregation bot. Here, two bots are located in Subnet-1 and Subnet-2 respectively, while the aggregation server is a host in Subnet-3.

4. **Local Aggregation Point**. In this scenario, the data stealing bots forward data to an aggregation bot which is located within the respective subnet and the aggregation bots exfiltrate the data to an external server. Such a communication architecture can potentially evade detection using the mechanism proposed in [20]. For this scenario, we considered subnets Subnet-1,Subnet-2 and for each subnet a compromised host aggregated data from two bots within the subnet before relaying it to the external server.

4.6.3 Existing state-of-the-art techniques

In addition to studying the performance of the proposed detection mechanism against different botnet scenarios, we compared its performance against two state-of-the-art detection techniques developed in [20] (referred to as *Stealthy P2P Detector*) and [19]. Details regarding these detection mechanisms are provided in Chapter 2.

In our experiments, we set the threshold values for different parameters of the existing mechanisms in a conservative manner to improve the chances of detecting architectural stealthy botnets. For instance, in the above botnet scenarios, the number of peers for any

data stealing bot was one and hence, we set the corresponding threshold (θ_{bgp} in [20]) to one thereby, reducing false negatives. Furthermore, existing detectors were proposed to monitor traffic between internal and external network at the network gateway. However, as argued earlier, in the case of architectural stealthy botnets, the volume of bot traffic intercepted at the network gateway is significantly smaller than the volume of traffic intercepted at internal monitoring points. Therefore, to handle such botnets, we mirrored traffic from all monitoring points to the detector for analysis and considered the internal traffic for analysis. Finally, for [19], we considered only the BSampling portion of the detection technique and used its output (i.e., the set of suspicious hosts) for comparison.

4.6.4 Evaluation Overview

In this work, we evaluated the performance of DeBot against the four scenarios across three dimensions: percentage of traffic intercepted, detection rate and processing time. For each performance dimension, we considered two observation epochs of 30 and 60 minutes and set $\epsilon = 100, minPts = 10$ to initialize the OPTICS clustering algorithm. Furthermore, to study the performance of the centrality-based snapshot rate method, we compared its performance against uniform strategy in which, at each epoch, a monitoring point is chosen uniformly at random. All experiments were repeated 30 times with different sequences of monitoring points and the performance of DeBot was compared by averaging the results over different sequences.

4.6.5 Traffic Interception

Fig. 4.4 shows the average number of flows intercepted over different scenarios. In each scenario, we observed that, independent of the observation epoch, the uniform-based strategy intercepted and processed more flows than the centrality-based strategy. Such an observation is expected as centrality strategy sampled traffic more often from monitoring points (M_1, M_6 and M_7) which carried far less traffic than the monitoring points near the DMZ region (M_2, M_4, M_5). However, as shown in Fig. 4.5, the centrality-based strategy intercepted a

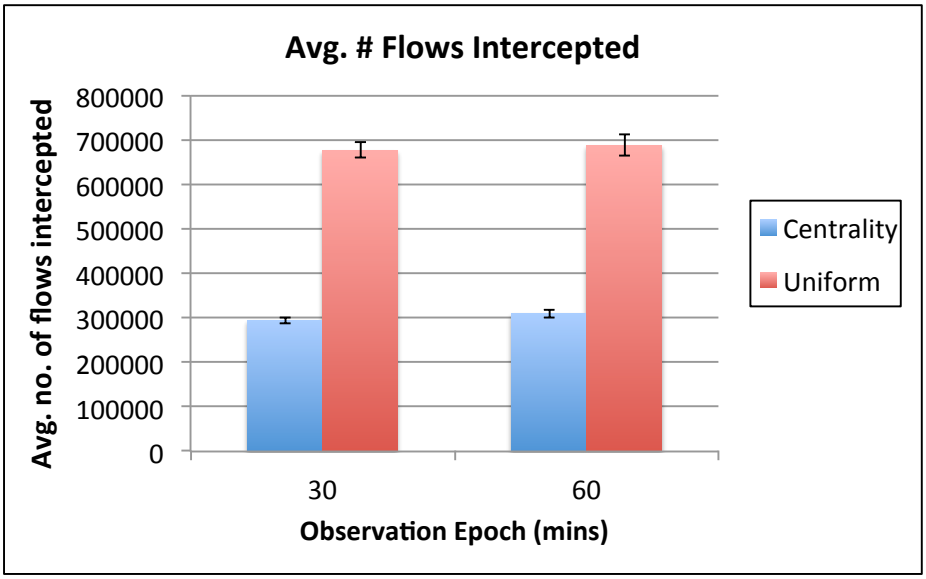


Figure 4.4: Average number of flows intercepted by different monitoring strategies

higher proportion of the bot flows than the uniform strategy. From Fig. 4.5, it can be seen that the increase in the percentage of intercepted bot flows depends on the number of monitoring points traversed by the bot traffic. For instance, In Scenario 1, the bot traffic traversed all the monitoring points (with varying volumes) while in Scenario 2 and Scenario 4, the bot traffic traversed only three monitoring points namely, M_6, M_7, M_1. Thus, as the number of opportunities to intercept bot traffic in Scenario 1 is higher, the proportion of bot flows intercepted by both uniform and centrality strategy were not significantly different.

4.6.6 Detection Rate

Detection rate is defined as the percentage of bot communication pairs that were identified as suspicious at the end of the refinement phase while false positive rate is the percentage of benign host communication pairs that were identified as suspicious. As shown in Fig. 4.6, the proposed mechanism detects more than 90% of the bot communication pairs in Scenario

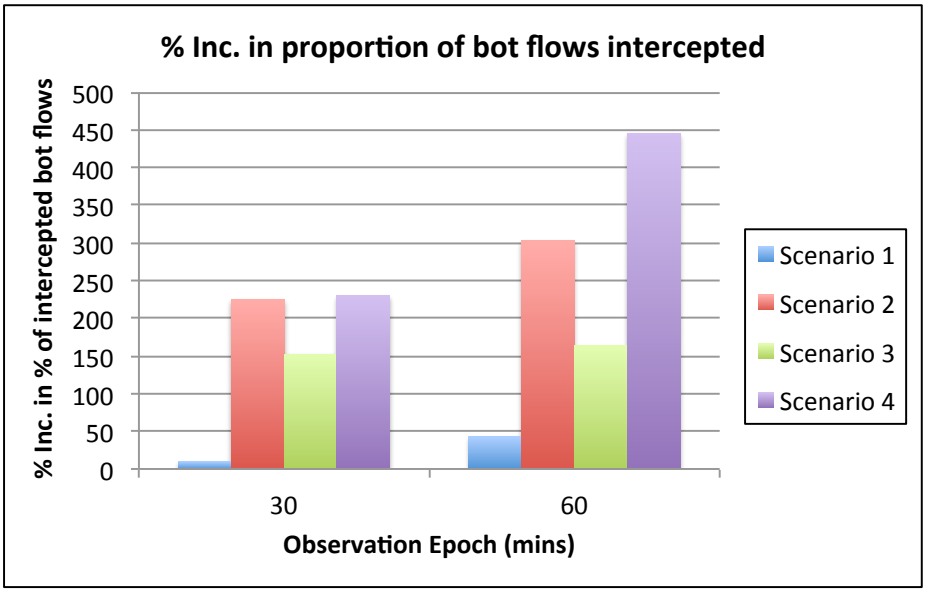
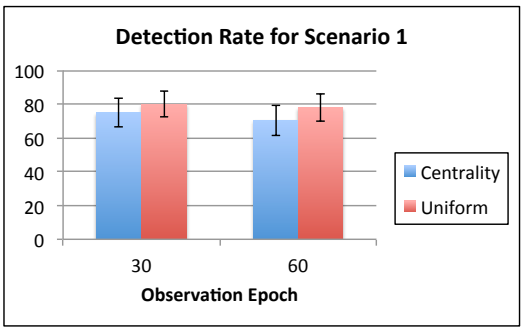


Figure 4.5: Increase in percentage of bot flows intercepted

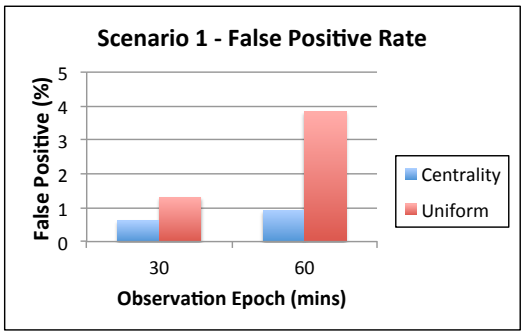
2 and Scenario 4 and, on an average, 70% of the bot communication pairs in Scenario 1 and Scenario 3. In other words, the botnet communication architectures in Scenario 1 and Scenario 3 are more stealthy than the other two architectures against our detection mechanism. While there is no statistical difference in the detection rates of centrality-based and uniform-based strategies, the false positive rates of centrality-based strategy ($< 1\%$) is much lesser than the uniform-based strategy. This significantly reduces the time and effort needed to perform fine-grained analysis on the captured flows using tools such as DPI.

4.6.7 Comparison with existing techniques

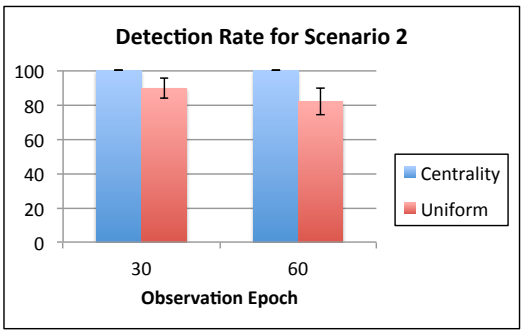
The output of the refinement phase in DeBot is pairs of hosts that belong to a botnet while, on the other hand, existing techniques – Stealthy P2P Detector (in [20]) and BSampling (in [19]) – output a set of hosts that are potentially bots. Therefore, to provide a normalized



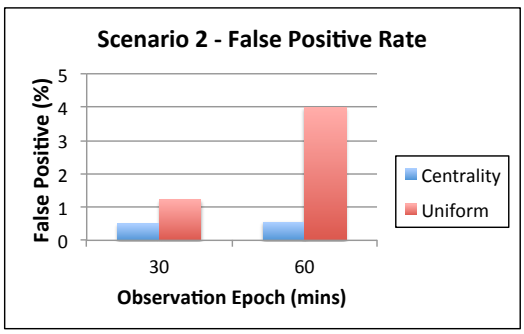
(a) Detection Rate for Scenario 1



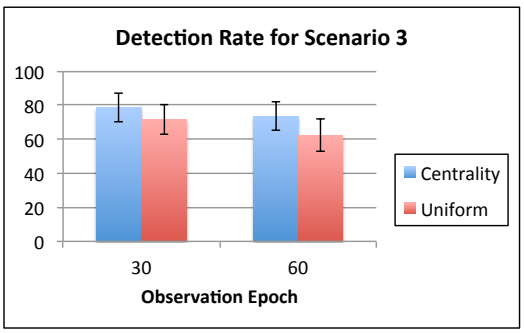
(b) False Positive Rate for Scenario 1



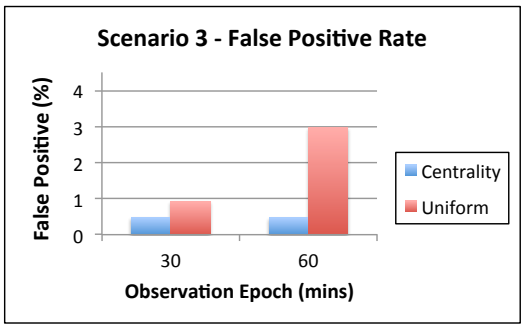
(c) Detection Rate for Scenario 2



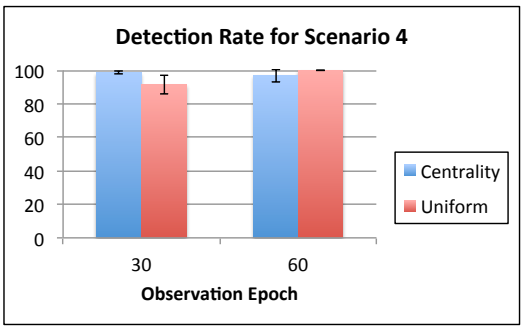
(d) False Positive Rate for Scenario 2



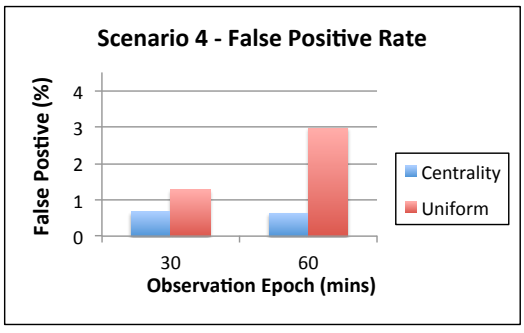
(e) Detection Rate for Scenario 3



(f) False Positive Rate for Scenario 3

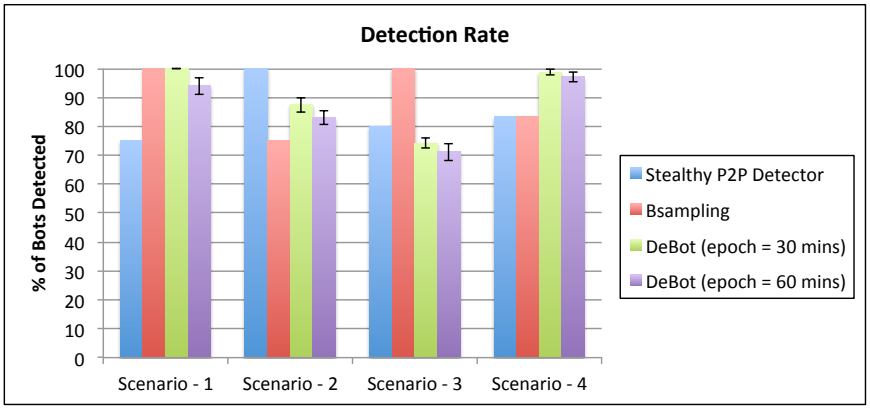


(g) Detection Rate for Scenario 4

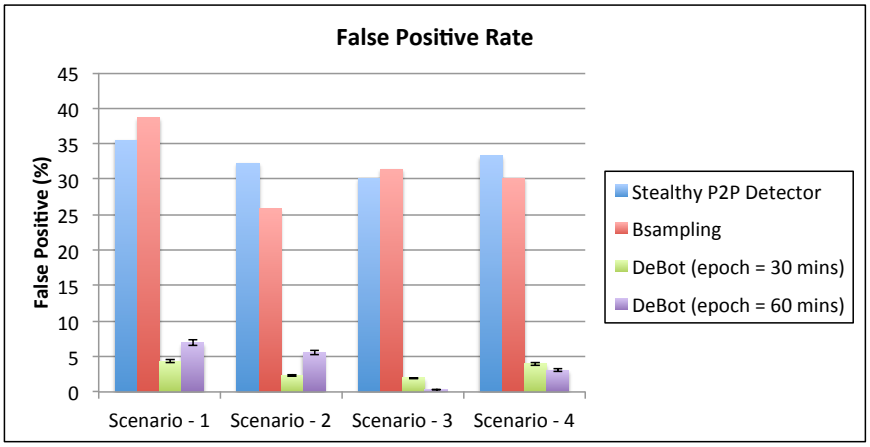


(h) False Positive Rate for Scenario 4

Figure 4.6: Detection rate and false positive rate for different scenarios



(a)



(b)

Figure 4.7: Detection rate and false positive rate comparison with existing work

comparison, for DeBot, we considered a host h_i as a potential bot if the following conditions were satisfied: (i) it was identified as suspicious during the observation phase and (ii) at least one connection pattern (to any host h_j) was identified to be relatively more periodic than other connection patterns generated by it during the refinement phase.

Fig. 4.7a presents the comparison in performance of DeBot against existing state-of-the-art techniques. It can be observed that DeBot (with observation epoch of 30 mins) has a

detection rate that is comparable with that of existing techniques in Scenario 1, Scenario 2 and Scenario 4. Although existing techniques have a higher detection rate in Scenario 3, the false positive rates (shown in Fig. 4.7b) are significantly higher than DeBot.

4.6.8 Processing Time

DeBot was implemented in Python and its processing time was recorded by running it on a Intel Xeon processor machine with 16 GB memory running Ubuntu 14.04. To compare the performance of DeBot with existing techniques, we implemented the Stealthy P2P detector (in [20]) and BSampling (in [19]) in Python. For the the purposes of normalized comparison, all techniques were run on the same environment.

Fig. 4.8 provides a comparison of processing times of DeBot (for different observation epochs) against existing state-of-the-art techniques. It can be observed that the Stealthy P2P detector performs worse than both DeBot and BSampling. This is because, Stealthy P2P Detector extracts several fine-grained features from the captured traffic and clusters flows based on these features. However, BSampling algorithm, samples only traffic that exhibit characteristics of a persistent bot. Therefore, the BSampling technique scales well with traffic.

As shown in Fig. 4.8, DeBot also executes in a time comparable with that of BSampling. To further understand the operation that contributes to its processing time, we segmented the total processing time into (i) data processing time – average time taken to extract the flow records from the captured packets for clustering and recording the connection initiation time for periodogram analysis, (ii) clustering time – average time taken for the OPTICS clustering algorithm to process the flow records to identify clusters and update the similarity score, and (iii) periodogram time – average time taken to identify suspicious host pairs in the refinement phase. Here, we considered the total data processing times and the total clustering times over the entire monitoring period of 12 hours and the observed times were averaged over different sequences of monitoring points.

From Fig. 4.9, it can be observed that the time taken to process the packets and extract

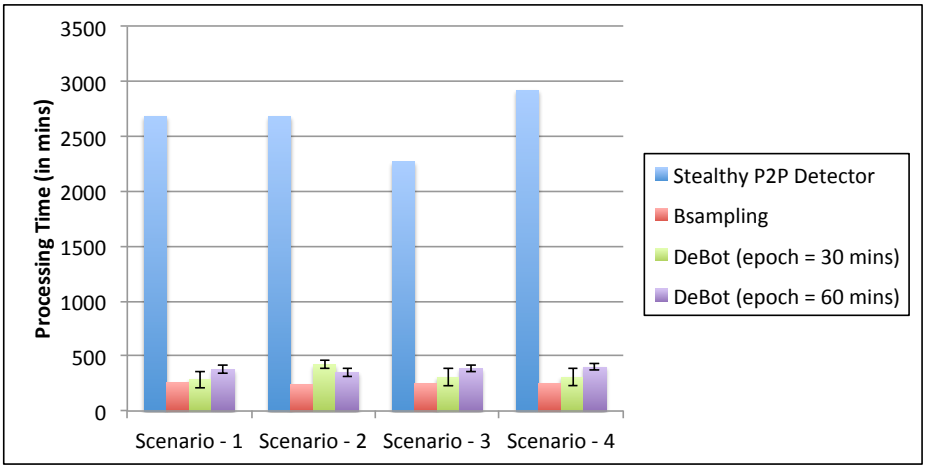


Figure 4.8: Processing time comparison of DeBot with existing state-of-the-art mechanisms

the flow statistics consumed most of the processing time. As for clustering, most of the time is taken to order the input flow records using a priority queue by the OPTICS algorithm. Finally, the time taken to determine the suspicious host pairs in the refinement phase was observed to be negligible.

4.7 Discussion

4.7.1 DNS-based botnets

In addition to centralized and P2P-based architectures, existing botnets also exploit DNS infrastructure to communicate with the external C&C server. In such botnets, the bots generate a series of domain names using an Domain Generation Algorithm (DGA) in a pseudo-random manner. A subset of these domains are registered by the attacker and act as rendezvous points between the bots and the C&C server. As the location of the C&C server changes in an unpredictable manner, the effort to takedown the botnet increases substantially.

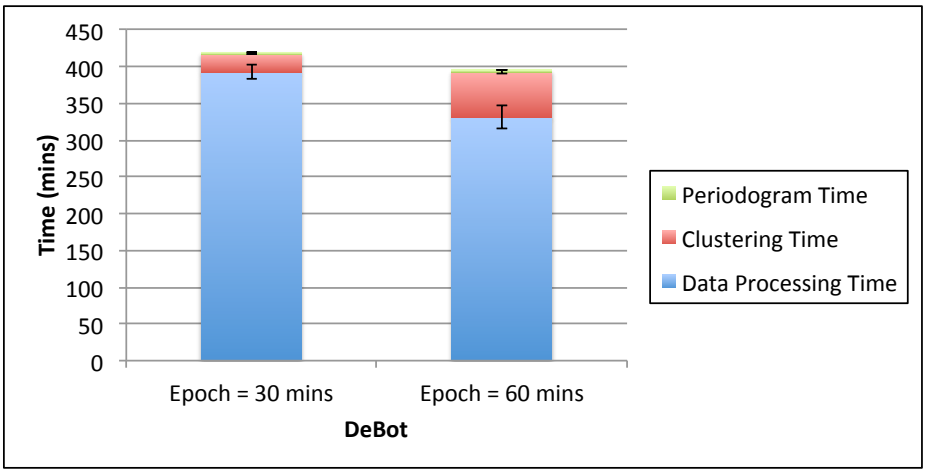


Figure 4.9: Processing time of different computationally-intensive operations of DeBot

Although the proposed detection mechanism is designed for detecting architectural stealth botnet, we argue that it is capable of detecting DGA-based botnets as well. Existing studies have shown that the *length* of the domain names (generated by a DGA) is longer than the benign domain names and can be leveraged to detect these bots [78]. Thus, the number of bytes sent per DNS request (one of the features that is used for clustering in the observation phase) by a bot will be different from that of benign systems. Hence, the system hosting the bot will potentially have a similarity score lesser than that of the other systems within its neighborhood. As a result, the system hosting the bot will be added to the suspicious hosts list. In addition to above feature, it is known that existing DGA-based bots generate domain names and resolve them *periodically*. The periodic querying behavior was exploited by Kwon et al. [21] to develop a technique that detected DGA-based botnets. Typically, enterprise networks employ a local DNS server to resolve domain name requests initiated by hosts within the network before recursively forwarding them to an upstream DNS server. If sufficient number of requests are intercepted, then the proposed detection mechanism will

be able to detect periodic querying pattern and hence, identify the corresponding bots. We intend to investigate such botnets as part of our future work.

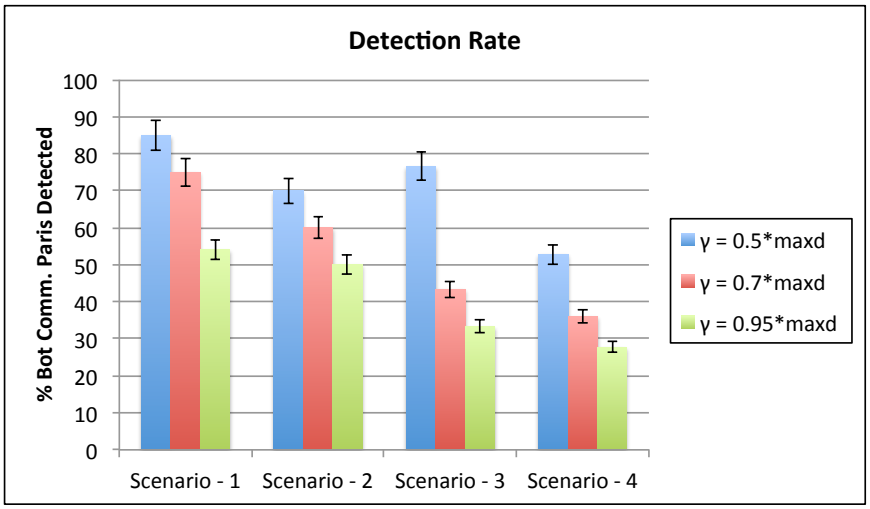
4.7.2 Host identification and neighborhood

In this work, each host is uniquely identified by its IP address. In an enterprise network, IP addresses are dynamically allocated by the DHCP server. To effectively detect bot flows, the DHCP server must be configured such that the leasing period of an IP address is equal to the monitoring period T. This would ensure that the IP address of the hosts do not change during the observation phase.

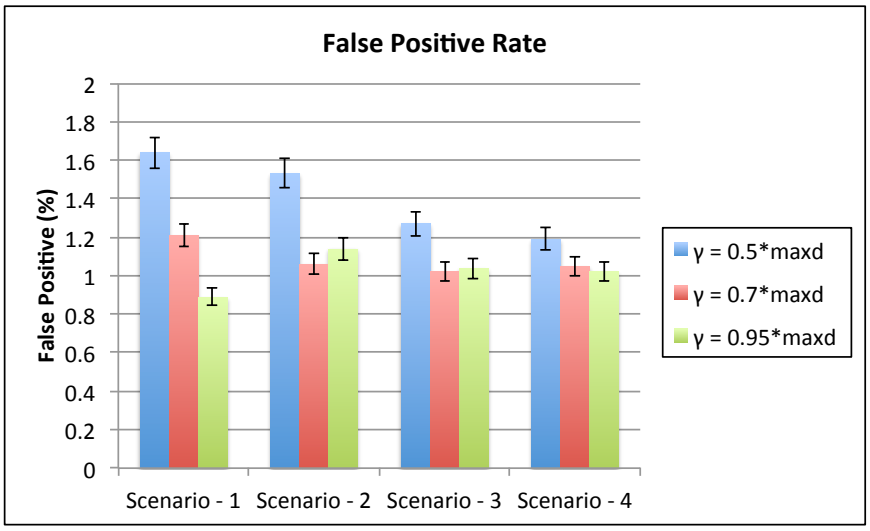
In this work, for the sake of simplicity, the neighborhood of a host was considered to be the set of hosts that were physically co-located in its subnet. Besides grouping hosts based on their physical location, hosts can also be grouped based on logical partitions within the network. For instance, network administrators use virtual LAN (VLAN) to improve security and ease network management by grouping hosts across different network segments. In such environments, each VLAN is mapped to an IP subnet with each host receiving a local-link address or an IP address from the DHCP server.

4.7.3 Evasion

One of the techniques with which attackers can evade detection is by reducing the frequency with which bots exchange update messages with its peers or the external server. The resulting connection patterns may not be detected as suspicious in the refinement phase. It should be noted that while such a communication architecture increases the complexity of managing the botnet, such an advanced botnet design might appear in the future. To understand the impact of infrequent communication between the bots, we modified the botnet to exchange messages at times randomly chosen between 0 and 600 seconds and studied the performance of the different snapshot rate methods (centrality and uniform) with an observation epoch of 30 mins. As shown in Fig. 4.10a, the detection rates for the centrality-based snapshot rate method decreases significantly (by approx. 50%).



(a)



(b)

Figure 4.10: Detection rate and false positive rate for infrequent bot communication

One way to prevent evasion through infrequent communication is by lowering the threshold distance, γ, which is used to identify connection patterns corresponding to a suspicious host that are exhibit anomalous high periodicity. Lowering the value of γ would potentially identify more host pairs (h_i, h_j) whose connection patterns are relatively more periodic than the underlying host h_i or h_j. Fig. 4.10a shows the improvement in the detection rate for different thresholds, γ. From the figure, it can be observed that low threshold values increase the detection rate as expected. Although, the improvement in detection rate increases the false positive rate (shown in Fig. 4.10b), the increase in false positive rate is less than 1%.

4.8 Conclusions

In this chapter, we addressed the problem of detecting botnets that operate under architectural stealth. To this end, we proposed DeBot, a network-based botnet detection mechanism that leverages two intrinsic behaviors of bots: (i) difference in traffic flow statistics between bots and benign applications and (ii) the periodicity in the connection patterns between a bot and its peers. DeBot was evaluated using a testbed environment against different botnet scenarios and the results indicate that while the detection rate of DeBot is similar to that of existing techniques, its false positive rate and the processing times are significantly lower making it an attractive candidate tool for detecting botnets.

Chapter 5: A Reinforcement Learning Approach to Detect Stealthy Botnets

5.1 Introduction

Defending a network against attack campaigns by stealthy botnets calls for a large-scale network monitoring solution. Due to the sophisticated nature of the attack campaigns, no single defense mechanism can prevent or detect it. Thus, motivated by the principle behind defense-in-depth that advocates the use of multiple security countermeasures to minimize an attacker's success, we consider two classes of countermeasures: Honeypots and Network-based botnet detection mechanisms. While honeypots are used to detect intrusion attempts [79], network-based botnet detection mechanisms identify bots that co-exists with benign machines through behavioral analysis [80].

Effective implementation of multiple countermeasures requires substantial monetary (to operate and maintain honeypots), storage and computation resources (to analyze large traffic volume). Furthermore, with the emergence of advanced botnets, an enterprise-scale network protection requires a mechanism that is both proactive in impeding the propagation of a botnet and reactive in its ability to detect bots residing within the network. To this end, in the chapter, we develop a reinforcement-learning model that deploys the defense mechanisms (here, honeypots and botnet detection mechanism) in an adaptive and a dynamic manner to reduce the lifetime of stealthy botnets in a resource-constrained environment.

Reinforcement learning (RL) is an algorithmic method for solving sequential decision-making problems wherein an agent (or decision-maker) interacts with the given environment to learn *how* to respond under different conditions. Formally, the agent seeks to discover a policy that maps the system state to an optimal action. In this work, through RL, the agent

learns a policy that maximizes the total number of bots detected over a long-run. As the location of bots are unknown prior to the deployment of defense mechanisms, in this work, the agent *estimates* the system state and the immediate reward of an action by monitoring network activity at different network segments. In particular, the agent tracks the scanning and the outgoing sessions behavior of hosts at different subnets to guide the placement of defense mechanism within the network.

Our reinforcement learning model differs from the existing game-theoretic approaches in several ways. First, the RL model does not consider a mission-centric approach to guide placement decision. Rather, the objective of the RL model is to maximize the number of compromised machines detected within the network; thereby, recommending placement strategies that secure the overall network. Next, although the RL model learns over a well-defined action space of the attacker, unlike existing stochastic game-theoretic models, it is oblivious to attacker's incentives in terms of the attacker's reward function for controlling machines within the network. Although, similar to the models in [49, 51], the RL model attempts to reduce an attacker's persistence in the network, the RL model considers a rich set of countermeasures (honeypots and botnet detection mechanism) to detect and remove bots during different phases of a cyber kill chain.

5.2 Threat Model

The objective of the attacker is to persist in the target network for an extended period of time and continuously exfiltrate data from the target network to an external server controlled by the attacker. In order to accomplish his/her mission, the attacker should compromise machines within the target network and establish a communication architecture between the compromised machines. The resulting network of compromised machines, or the botnet, continuously exchange messages with one another and relay stolen data to an external C&C server.

In this work, we model the lifecycle of a bot as shown in Fig. 5.1. The lifecycle of a bot begins when a benign system within the the target network is compromised by either

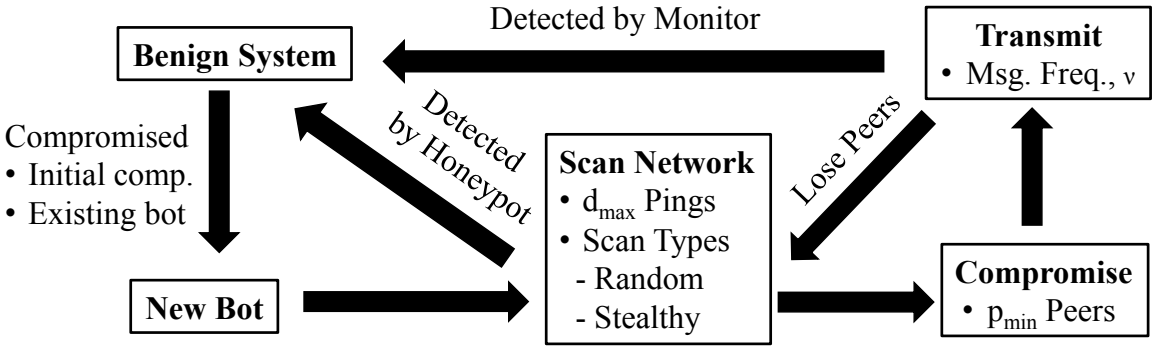


Figure 5.1: Lifecycle of a bot

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 new bot can perform two types of scans: worm-like scan 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 [81].

These discovery probes include ICMP ping packets, incomplete TCP handshake etc. to determine whether a system is hosted at the chosen IP address and also to learn the running configuration of the system such as OS version, services etc. Due to the randomness in these scans, the bots may send discovery probes to machines that may raise suspicion. For example, if a bot on a client machine sends discover probes to another client machine, then the scanning activity may be detected as anomalous in an enterprise network. In the stealthy scanning strategy, on the other hand, the bots first enumerate the active connections of the underlying host and send discovery probes only to these machines. Several existing malwares employ this mechanism to move laterally through the network [82]. Independent

of the scanning strategy, we set an upper bound on the number of discovery probes, d_{max}, that can be sent by a bot over a period of time (described later) to further improve stealth.

After enumerating the victim machines, the bot compromises these machines and adds them to its list of peers. As mentioned above, the botnet model assumes that all machines are vulnerable and can be successfully exploited. In order to build a resilient botnet, we parameterize the minimum number of machines, p_{min}, that the bot must compromise. Upon recruiting new machines, the bot begins transmitting update messages with its peers. These messages inform the attacker 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 pristine/benign state. In the considered bot lifecycle, if the number of active peers of a bot drops below the pre-defined minimum number of peers, p_{min}, then corresponding 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.

5.3 Defense Mechanisms - Overview

In this work, we consider two countermeasures: Honeypots and Network-based detection mechanism. A brief overview of these mechanisms is provided below.

Honeypots: Honeypots are systems that co-exist with other machines within the network and are deliberately configured with vulnerabilities to lure an attacker to scan and compromise them. As the only functionality of honeypots is to observe intrusion attempts, they have a low false positive rate since any interaction with a honeypot can be directly attributed to an intrusion attempt [79]. Honeypots log all intrusion attempts and are used to study the nature of the attacks against a target organization. Depending on the level of maintenance and realism, honeypots are broadly classified as either low-interaction or high-interaction.

Low-interaction honeypots emulate vulnerable services without exposing a full operating

system functionality [79]. They limit the number of operations that an attacker can execute and hence, are easy to maintain and monitor. However, due to their limited capability, low-interaction honeypots can only capture initial intrusion *attempts* such as login attempts and are unable to analyze an attacker's subsequent objectives. Moreover, the presence of low-interaction honeypots can be detected through side channel timing analysis [83]; thereby, further reducing their utility in preventing sophisticated attacks. High-interaction honeypots, on the other hand, are dedicated machines that run a fully-functional operating system with vulnerable services and hence, can provide additional insights into an attacker's intentions and techniques. Although the data collected by high-interaction honeypots provide a rich source of information, they incur a high maintenance and operational cost. Therefore, in this work, we bound the number of high-interaction honeypots, H_{max}, that can be hosted by a network defender.

Typically, honeypots are deployed in the DMZ region to monitor attack attempts on the production servers. However, many known attack campaigns establish their initial foothold by first compromising client machines through client-side attacks, social engineering etc. Once established, an attacker moves laterally through the network by compromising additional machines and escalating privileges. To detect the lateral movement of an attacker through insecure portions of the network, we advocate the need to deploy high-interaction honeypots in the internal network. With an increasing adoption of Software-Defined Networks (SDNs) to control the flow of traffic through a network, these honeypots can be physically located in a different zoned region of the network while individual honeypots can be made discoverable from different network segments by installing the corresponding flow rules on all the corresponding switches using the centralized controller [84].

Network-based detection mechanism: While honeypots assist in preventing lateral movement of an attacker, detection mechanisms, on the other hand, enable a defender to locate compromised machines that persist in the network and operate in a stealthy manner. Existing network-based detection techniques monitor traffic at the network gateway as they assume that all the peers and C&C server of all the bots are located outside the network.

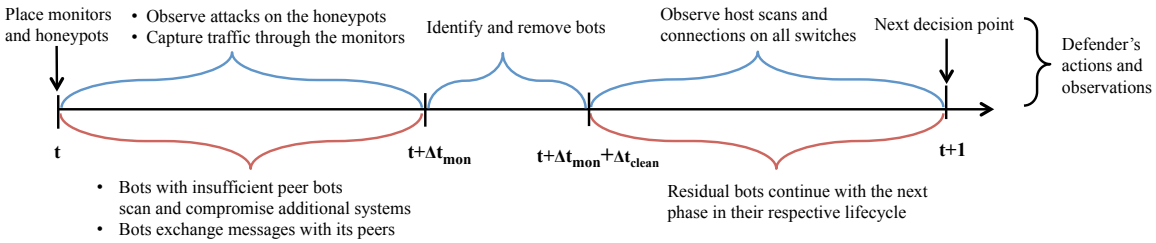


Figure 5.2: Timeline of defender's and attacker's actions and observations

However, in the presence of a botnet that operates under architectural stealth, the volume of observable bot traffic at the network gateway is reduced; resulting in a low detection rate. A trivial countermeasure to capture a higher volume of bot traffic is to monitor traffic through all internal routers/switches by mirroring traffic from the corresponding portions of the to a central location. However, due to the poor scalability of these detection mechanisms coupled with an ever-growing internal benign traffic volume, detecting bots in a reasonable amount of time becomes infeasible. Hence, in this work, we only consider traffic from M_{max} network segments at any given time and monitor these segments for bot-related activity. Furthermore, to improve network coverage, we dynamically select different segments of the network to monitor traffic. As presented in Chapter 3, continuously changing the monitored portion of the network improves the chances of intercepting botnet traffic.

5.4 Reinforcement Learning Model

In this work, the defender's objective is to maximize the number of bots detected and removed using a limited number of resources (here, 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 the bots

can propagate through the network, the locations of these defense mechanisms must also dynamically change to detect bots in different segments of the network. Due to the evolving nature of the threat, we propose a reinforcement learning (RL) approach to guide the defender's sequential decision-making process of placing monitors and honeypots over the time horizon.

In our model, we consider an infinite horizon wherein the agent makes decisions on a periodic basis; the time between decisions is referred to as an epoch. A timeline with the sequence of events that occur between consecutive decisions is shown in Fig. 5.2. At each decision point, the agent determines the network segments that will be monitored during the next epoch. At the beginning of an epoch, bots perform one of two detectable activities depending upon the stage in their respective lifecycle: (i) scan and subsequently, compromise machines within the network (referred as scanning bots) or (ii) transmit/exchange update messages with their peers and the C&C server (referred as transmission bots). The agent observes the network activity for a time period Δt_{mon} during which (i) honeypots may be scanned and compromised by the scanning bots, and (ii) traffic through the monitors is captured for analysis by centralized bot detection mechanism. At $t+\Delta t_{mon}$, the bot detection mechanism processes the captured traffic and outputs a set of potential bots within the network. It must be noted that the network-based detection mechanism is considered to be imperfect with a known true positive rate (< 1) while inferences based on network activity on honeypots is considered to be perfect with a true positive rate of 1.

After identifying the potential bots within the network, the defender removes them by restoring the corresponding machines to their pristine state. Let Δt_{clean} be the time taken to process the captured traffic by the detection mechanism 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, the bots that remain undetected continue with the next stage in their respective lifecycle; bots with insufficient peers will scan the network while bots with sufficient peers will exchange messages. The basic elements of the model are defined below.

Decision Variable: Given N monitoring points, the agent may choose to either passively monitor, m, traffic traversing a monitoring point, or place honeypot, h on a monitoring point or place both, b, types of defense mechanisms. Moreover, due to resource constraints, the agent may not be able to place either type of detector, e. The set of feasible decisions/actions at time t is represented as a vector: $x_t = (x_1^t, x_2^t, ..., x_N^t)$ where $x_i^t \in \{e, m, h, b\}$. It should be noted 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 the bots within the network. However, as the location of the bots are unknown prior to the placement of defense mechanisms, we derive the state of the system by observing the attack indicators in different segments of the network. Anomalous behaviors such as large number of unsuccessful login attempts, increase in number of host scans and large number of outgoing sessions are some of the most common symptoms of an ongoing attack [85]. Thus, in this model, we determine the potential locations of bots in a network by observing anomalous behaviors in different segments of the network. In particular, to estimate the number of bots at different segments of the network, we track the total number of host scans and 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. 5.2. It must be noted that these features can be observed at all monitoring points (i.e., switches) with very low overhead.

For N monitoring points in a network, the state of the system, S_t, at any time t, is defined as a $2 \cdot N$-dimensional vector $(\psi_1^h, \psi_1^s, \psi_2^h, \psi_2^s, ..., \psi_N^h, \psi_N^s)$ where ψ_i^h, ψ_i^s are, respectively, the host scans state and the sessions state of the monitoring point $i, i \in [1, N]$. 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 instance, if μ_i^f and σ_i^f are the mean and standard deviation of each feature $f \in \{h, s\}$ at

monitoring point i, then the state at any given time t can be defined as:

$$\psi_i^f = \begin{cases} HIGH, & Total_i^f(t) \geq \mu_i^f + \sigma_i^f \\ MED, & Total_i^f(t) \in (\mu_i^f - \sigma_i^f, \mu_i^f + \sigma_i^f) \\ LOW, & Total_i^f(t) \leq \mu_i^f - \sigma_i^f \end{cases} \tag{5.1}$$

where, $Total_i^f(t)$ is total number of observations of feature f that was recorded during the time period $[t + \Delta t_{mon} + \Delta t_{clean}, t + 1)$. The intuition behind the Eq. 5.1 is that any large deviations from the expected behavior is considered to be anomalous. It must be noted that the objective of this work is *not* to design a botnet detection system rather a strategy for placing defense mechanisms. While fine-tuning the definition of ψ_i^f will yield accurate results, it is beyond the scope of this work.

Reward Function: In a 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 is 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 the recent observations. Such problems occur in real world such as when travel and hotel reservation decisions are done today for a future date whose value is unknown until the date has occurred [86–88].

In this work, the number of bots in a network segment is estimated 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 for each machine across all monitoring points. Let $\mu_{mc,i}^f$ and $\sigma_{mc,i}^f$ be the mean and standard deviation of the feature f for the machine mc when observed from monitoring point i. We consider a simple threshold scheme to check whether a machine is a

potential bot: Given any machine mc and a monitoring point i, if $Total^f_{mc,i}(t)$ is total number of observations of a feature f that was recorded during the time period $[t+\Delta t_{mon}+\Delta t_{clean}, t+1)$, then the machine mc is considered as a bot if and only if $Total^f_{mc,i}(t) \geq \mu^f_{mc,i} + 3 \cdot \sigma^f_{mc,i}$. It should be noted that this rule to identify suspicious machines can be modified based on the specific settings of the target network and does not limit the generality of the proposed RL model.

Post-decision System State: The post-decision system state, S^x_t, is the immediate transition in the state of the system 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; here at time $t+1$. 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 inspecting the behavior of the machines that exhibit anomalous behavior. In particular, suppose the machines $mc_1, mc_2, ..., mc_k$ are identified as potential bots from the monitoring point i (due to their deviations in the feature, say, f), then placing a defense mechanism (honeypot if f is the host scans count, otherwise monitor if f is the sessions count) is expected to restore the machines $mc_j, j \in [1,k]$ to their pristine state. As a result of the cleaning process, the agent expects to record $\mu^f_{mc_j,i}, \forall j \in [1,k]$ 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 (at $[t-1+\Delta t_{mon}+\Delta t_{clean}, t)$), then the new post-decision state of monitoring point i for a feature f can be obtained by computing using Eq. 5.1 where the estimated total number of observations of feature f, $\widehat{Total}^f_i(t+1) = \sum_{j \in [1,k]} \hat{\mu}^f_{mc_j,i} + \sum_{j \notin [1,k]} \mu^f_{mc_j,i}$ where $\hat{\mu}^f_{mc_j,i}$ is the estimated behavior of the 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 pre-processing 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 arrives 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 within a network. In the RL model, the uncertainty is captured by observing the network activity and extracting features from different monitoring points.

State transition function: The state transition function, $S_{t+1} = \tau(S_t, x_t, B_{t+1})$, defines the next evolution of the system state at time $t+1$. 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 measured as the long-run total discounted value of the states $V^j(S)$ as the the iteration index $j \to \infty$, which is derived using the recursive Bellman's optimality Equation (Eq. 5.2) shown below [89]. Here, $V^j(S)$ is the cumulative sum of discounted $R(S_t, x_t)$ rewards for the learning phase whose iterations are index 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 $V^j(S)$; higher the value of $V^j(S)$, the better the system state. The model strives to transition from one good state to another by making a decision that is guided by the highest value of the estimated future states that are reachable at any given time t.

5.4.1 Phases of Reinforcement Learning

RL achieves the objective by processing in three phases, namely, exploration, learning and learned. The recursive Bellman's optimality equation that updates the value of the states

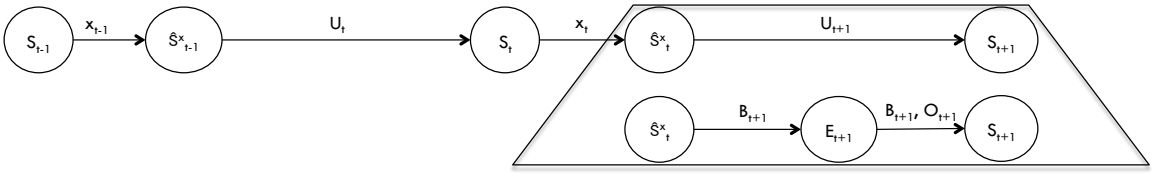


Figure 5.3: State transition diagram

is given as follows:

$$V^j(\hat{S}^x_{t-1}) = (1-\alpha^j)V^j(\hat{S}^x_{t-1}) + \alpha^j \nu^j \tag{5.2}$$

$$\nu^j = \left[\max_{x_t \in \mathcal{X}} \left\{ R(S_t, x_t) + \beta V^j(\hat{S}^x_t) \right\}\right] \tag{5.3}$$

where $\hat{S}^x_t$ is the estimated post-decision state reached by the system at state S_t under the action x_t, α^j is the learning parameter that is decayed gradually, $\mathcal{X}$ is the set of all feasible decisions from which the model will choose a decision at every iteration and β is the fixed discount factor that allows the state values to converge in a long-run. It should be noted that the value of the estimated post-decision state $\hat{S}^x_{t-1}$ is updated at time t (in Eq. 5.3) using the estimated reward function and the value of the estimated post-decision states that can be reached under different actions. In a classical RL formulation, the immediate real rewards and the immediate value of the post-decision states at time t are known; hence, the value of the post-decision state at time $t-1$ can be updated using Eq. 5.2 and Eq. 5.3. However, as both the rewards and post-decision states are estimated, we update the value of the post-decision state only after the real rewards for an action is observed.

A snapshot of the state-transition diagram is shown in Fig. 5.3 in which U_t denotes the uncertainty (before and after removing bots) after taking an action and O_{t+1} denotes the

features observed at different monitoring points after removing the bots; the bot 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} unfurls, transitioning the system to the state S_{t+1}, As the uncertainty U_{t+1} unfurls (shown in trapezoidal box in Fig. 5.3), bots are removed at stage E_{t+1} and the agent observes the real rewards which is then used to update the value of the estimated post-decision state $\hat{S}^x_{t-1}$. The three phases of learning are described below:

Exploration Phase: In this phase, the RL algorithm would explore several non-optimal decisions and acquire the value of the system states that are visited. As described in Alg. 3, Eq. 5.2 is used without the max operator in Eq. 5.3 by taking random decisions for placing defense mechanisms, and the value of $V^j(S^x_t)$ and $V^j(\hat{S}^x_{t-1})$ is used from the previously stored values if the state was visited or 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.

Learning Phase: In this phase, the algorithm would take (near-) optimal decisions at time t, which is obtained from Eq. 5.3 with the max operator (lines $13 - 14$ in Alg. 3). The value of the post-decision state at time $t-1$ is updated at time $t+1$ as per Eq. 5.2 with the real rewards. After several iterations, learning is stopped when convergence of the value of the states is achieved, as measured in terms of the mean-square error of the stochastic gradient [88].

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 would take optimal decisions at each time t, which is obtained from Eq. 5.3 with the max operator. The algorithm will evaluate all its feasible actions and chooses an action that takes the system to the post-decision state with the highest value.

Algorithm 3: Exploration and Learning Phase

Require: The baseline values ψ_i^f of each feature, $f \in \{h, s\}$ for each monitoring point i, baseline values $\psi_{mc,i}^f$ for each machine mc, the decision space $\mathcal{X}$, initial learning parameter $\alpha^0 = 0.8$ at time $t = 0$, the discount parameter $\beta = 0.95$, number of iterations for learning, $J = 1000$.

Ensure: State value function, $V(S), \forall S$

For all states, set $V(S) = 0$
for all $j = \{1, ..., J\}$ **do**
  if $j \leq 0.3 \cdot J$ **then**
    $Phase = Exploration$
  else
    $Phase = Learning$
  end if
  for all $t = \{1, ..., 365\}$ **do**
    Observe features from the monitoring points and determine state S_t
    if $Phase == Exploration$ **then**
      Choose a random defense placement decision, x_t
    else
      Estimate the immediate reward $R(S_t, x_t)$ and the post-decision state $\hat{S}_t^x$ as described in Sec. 5.4, $\forall x_t \in \mathcal{X}$
      Choose the action x_t' that gives the maximum value in Eq. 5.3
    end if
    if $t > 2$ **then**
      Observe the real reward at $t + \Delta t_{mon} + \Delta t_{clean}$
      Decay the learning parameter, $\alpha^j = \frac{\alpha^j}{1+e}$, where $e = \frac{j^2}{1.25 \cdot 10^{14} + j}$ (see [90])
      Update the value of the post-decision state $V^j(\hat{S}_{t-1}^x)$ using Eq. 5.2 and Eq. 5.3 with the real reward and the value of α^j.
    end if
  end for
end for

5.5 Simulation Results

In this work, we consider an enterprise network shown in Fig. 5.4 to study the performance of the RL model. The network is composed of 106 machines with 98 clients machines and 8 servers that are distributed across 4 subnets. In this network, the four switches, SW_1, SW_2, SW_3 and SW_4, act as the monitoring points for placing the monitors and honeypots. These switches, as depicted in the figure, are remotely controlled by an administrator to: (i) activate/deactivate mirroring ports that mirror traffic to the central monitor for analysis and, (ii) insert flow rules on the switches to make honeypots discoverable at different subnets in a SDN-based network. It should be noted that the physical location of the honeypots does not influence an agent's decisions to place the defense mechanisms and hence,

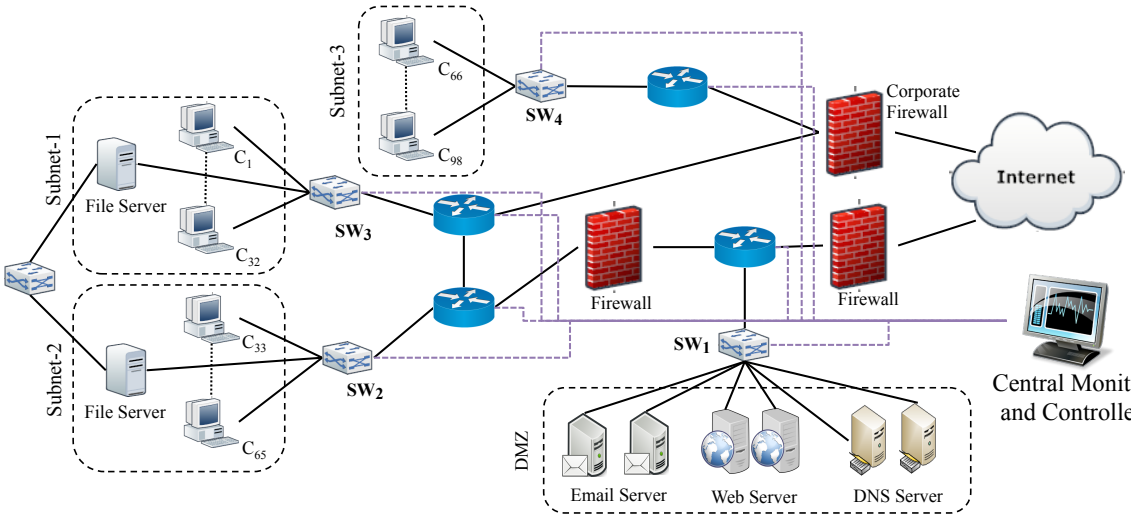


Figure 5.4: Enterprise network

their location is ignored in the above figure.

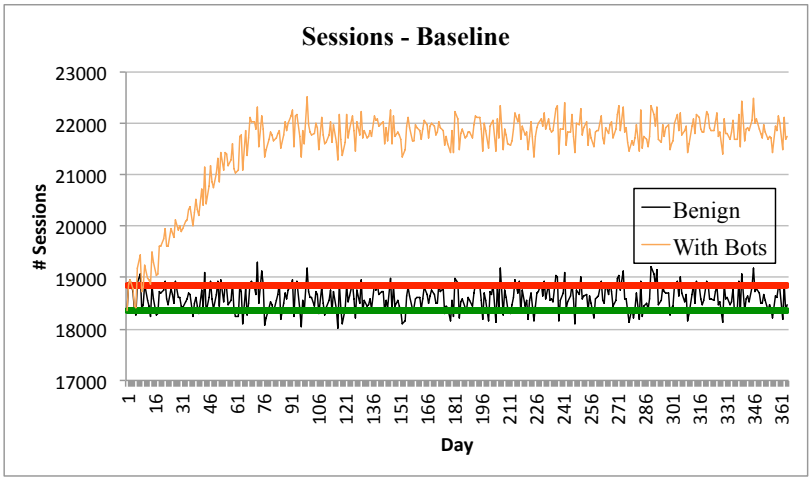
Defender's Parameters: For the considered, we set $H_{max} = 2, M_{max} = 2$ i.e., a maximum of two honeypots and monitors can be placed in the network. Additionally, existing botnet detection mechanisms [18, 20] show a true positive rate of at least 90% when evaluated against known malwares. However, if an attacker introduces perturbations to the malware's behavior, the performance of the detection mechanism may degrade. Hence, to account for the performance degradation, in this study, we conservatively estimate the true positive rate at 70%

In this work, we consider a 365-day scheduling horizon in which the agent makes decisions at the start of each day. During each epoch (1 day), for the first 12 hours i.e., $\Delta t_{mon} = 0.5$, the defender observes the network activity on the deployed honeypots and mirrors traffic from the monitors to the central location. For the purposes of this simulation-based study, we set $\Delta t_{clean} = 0$, i.e., the time taken for the detection mechanism to analyze the captured traffic and restore the identified machines to their pristine state is considered to be instantaneous.

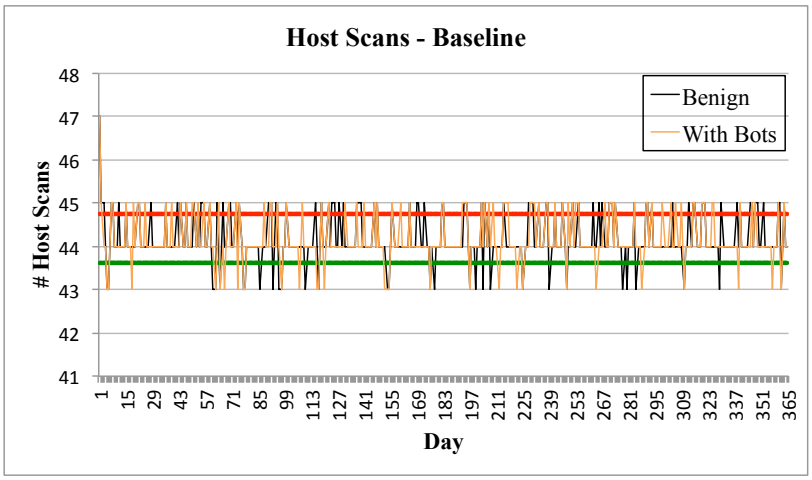
Benign Traffic Generation: The network activity of benign machines play a crucial role in setting the baseline behavior of the observable features at different monitoring points and consequently, in the defense placement decisions taken by an agent. In our simulation, the traffic characteristics were obtained from [91] in which the researchers developed traffic profiles for different types of protocols based the traffic captured in a testbed environment. In the considered network, each machine generates four types of traffic: HTTP, FTP, SMTP (for Email) and DNS with a traffic composition similar to that of [91]. Since the agent's decisions are influenced by the total number of recorded observations for each feature, the simulation only requires the distribution of the rate at which machines generate requests. A summary of the request rate distributions for different types of traffic is provided in Table 5.1. In addition to complete sessions (for HTTP, FTP and Email), we considered a 1% packet drop rate to simulate scanning-like behavior for benign machines.

In order to set the baseline behavior, we considered a 365-day simulation in which each machine generated traffic in accordance to the request rates in Table 5.1. During this simulation, we learn the feature statistics, μ_i^f, σ_i^f of each feature $f \in h, s$ for each monitoring point i at any epoch and $\mu_{mc,i}^f$ and $\sigma_{mc,i}^f$ of each feature f for each machine mc as observed from the monitoring point i. Fig. 5.5 illustrates the baseline behavior for the sessions and host scans state of switch, SW_2. In the figure, the red line and the green line provide the demarcations for the HIGH, MED and LOW states.

Botnet Parameters: In addition to the benign traffic, compromised machines generate traffic adhering to the bot lifecycle described in Sec. 6.2. In our simulation, the parameter to generate the network activity of a bot were determined either based on known behavior obtained via malware analysis reports or through a conservative estimate that would enable a stealthy operation. At the start of the simulation, machines within the network are assumed to be compromised by a client-side attack with an initial compromise probability of 0.03 i.e., on an average ≈ 3 client machines are expected to be initially infected. After establishing the initial foothold, the bots scan the network and enlist $d_{max} = 3$ machines for subsequent compromise. As described earlier, the bots can perform two types of scans: random and



(a) Sessions



(b) Host Scans

Figure 5.5: Total number of sessions and scans observed from switch SW_2

stealthy; the bots choose the scanning strategy with a probability of 0.8 and 0.2, respectively. Here, we consider a stealthy botnet in which each bot compromises only $p_{min} = 1$ additional machine. Upon compromising and recruiting additional bots, the bots begin exchanging messages with their peers every $\nu = 15$ minutes. Finally, in this simulation, we considered

Table 5.1: Benign traffic - Request rate distribution

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

a best case setting for a botnet wherein the bots were always present in the network i.e., if all the bots are removed by the defense mechanisms, then new bots are created through a client-side attack in the subsequent epoch.

Fig. 5.5 also illustrates the spike in network activity (w.r.t. no. of sessions and host scans) in the presence of such a botnet when no defense mechanisms were installed in the network. It must be noted that as there were no defense mechanisms, after the initial scan (during the first few epochs) and subsequent compromise, the bots do not return to their scanning state. Hence, no anomalous scanning activities is observed in the subsequent epochs.

5.5.1 Comparison with Alternative Strategies and Metrics

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

- **Static Strategy**: In this strategy, the defender's objective is to detect any botnet activity

in the segments of the network that hosts sensitive data. In the considered network, the file servers in Subnet-1 and Subnet-2 were assumed to host sensitive data and hence, in this strategy the defender places both the monitors and honeypots on switches SW_2 and SW_3.

- **Centrality-weighted Strategy**: This strategy – proposed in Chapter 3 – is a proactive approach to dynamically place defense mechanisms in a network. Assuming that the file servers in Subnet-1 and Subnet-2 host sensitive data, this strategy models the network as a graph and computes a centrality metric referred to as *mission-betweenness centrality* for each monitoring point in the network; a monitoring point with a high centrality value is expected to intercept a larger volume of bot traffic and hence, improve the chances of detecting botnets that operate under architectural stealth. After computing the centrality, the strategy chooses monitoring points randomly weighted by their centrality values; higher the centrality value, higher the probability of being chosen. Although the strategy was proposed to place only monitors, in our simulations, the placement of honeypots were also guided by the same policy.
- **Myopic Strategy**: This strategy is motivated by the existing practice to analyze traffic in a network segment *only* when the network operator observes a significant spike in the network activity. In particular, the defender places a honeypot (a monitor) on a network segment when the host scan state (sessions state) reaches the HIGH state. Unlike the earlier strategies, the defense placement decisions in this strategy are driven by the information acquired from the operating environment. However, dissimilar to the RL model, this strategy does not account for the long-term values of the state that the system may transition to as a result of the decision.

The following metrics were used to compare the above strategies:

- **Bot Liveprint**: Maximum number of bots present during the simulation.
- **Bot Meanprint**: Average number of bots present during the simulation.
- **Maximum Bot Lifetime**: The lifetime of a bot is the time period (in days) during which the underlying machine was controlled by an attacker. The maximum bot lifetime is the longest time period during which any machine was controlled by an attacker.

Table 5.2: Performance comparison of different strategies

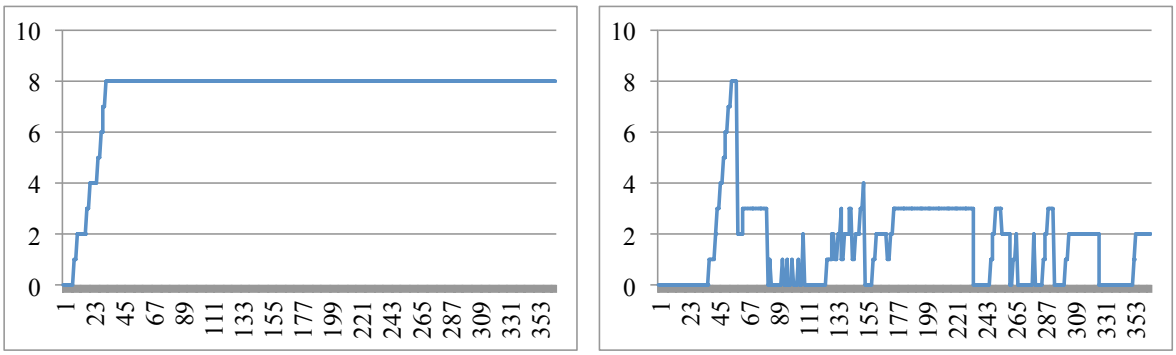
Strategy	**Avg. Bot Liveprint**	**Avg. Bot Meanprint**	**Avg. Max. Bot Lifetime (days)**	**Avg. BPT (days)**
Static	14.06	12.30	360.3	362.1
Centrality	11.28	2.89	48.94	74.86
Myopic	11.44	3.34	26.84	52.30
RL	12.90	2.97	20.0	37.40

- **Botnet Persistence Time (BPT)**: The maximum lifetime of a botnet where the lifetime of a botnet is defined as the time between the appearance of the first bot in the network to the time when all the bots were removed. As mentioned earlier, in our simulation, when all the bots are removed from the network, the simulator spawns new bots.

5.5.2 Results

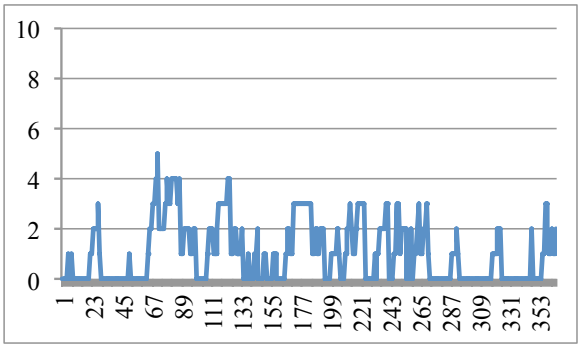
In this work, the network was simulated using PeerSim [92] – a simulator that was designed to support scalable simulation of P2P protocols. All the strategies were evaluated for a 365-day cycle with defense placement decisions made at the beginning of each epoch (or day). For each strategy, the simulation was repeated for 50 runs in which each run was initialized with a different seed that controlled the operations of a bot. The performance of each strategy was averaged across all the runs and the average value was used for comparing strategies at 95% confidence interval.

The summary of the results is provided in Table 5.2. As expected, the static placement strategy performs the worst amongst its alternatives. While the static strategy ensured that the bots in Subnet-1 and Subnet-2 were removed, it could not detect bots in Subnet-3. As a result, once a botnet was established in Subnet-3, the bots continued to persist in the network till the end of the simulation.

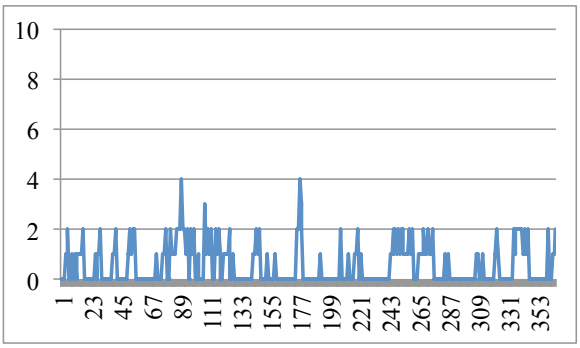


(a) Static Strategy

(b) Centrality Strategy



(c) Myopic Strategy



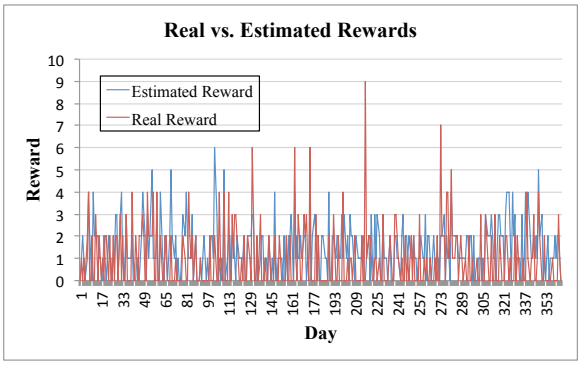
(d) RL Strategy

Figure 5.6: Total number of bots in the DMZ for one 365-day run

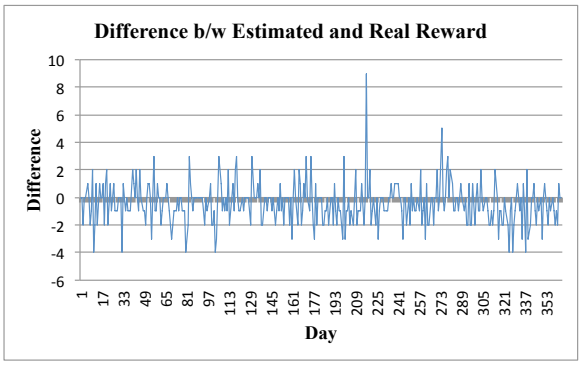
The dynamic strategies, on the other hand, provide a significantly better protection against such stealthy and persistent botnets. In particular, the Centrality-weighted and the Myopic strategies reduce the maximum lifetime of a bot by at least 85% and consequently, reduce the lifetime of a botnet (BPT) by more than 80%. The performance of these strategies strongly suggest that introducing *dynamism* while making defense placement decisions – guided by simple and/or intuitive policies – can significantly improve the security posture of the network.

Between the Centrality-weighted and Myopic strategies, we did not observe a statistically significant difference in their performance across the following metrics: bot liveprint and meanprint. However, the Myopic strategy showed a significant improvement (over Centrality-weighted strategy) in reducing the lifetime of a bot and as well as the lifetime of the botnet. This was because the decisions taken by the Centrality-weighted strategy were largely concentrated around Subnet-1 and Subnet-2 (since, the corresponding switches had high centrality values) and hence, the bots in Subnet-3 and DMZ persisted for a longer period of time. Myopic strategy, on the other hand, took decisions based on the information obtained from the network; the strategy immediately reacted to spikes in the network activity and placed defense mechanisms on those subnets. Thus, when bots propagated and operated in Subnet-3 (or DMZ), the Myopic strategy observed a spike in the network activity and removed corresponding bots quicker than the Centrality-weighted strategy (as shown in Fig. 5.6). This improvement in performance strongly suggests that the decisions that are guided by the information acquired from an environment leads to effective defense placements.

Finally, amongst the dynamic strategies, the RL model showed the largest reduction in the lifetime of a bot as well as in the lifetime of the botnet (BPT). Similar to the reasoning behind the performance of Myopic strategy, the RL model's performance stems from the fact that its decisions are based on the information acquired from the operating environment. However, the RL model synthesizes the acquired information to estimate the immediate reward (in terms of number of bots) and the post-decision state – whose long-term value is obtained through learning – to make decisions. A comparison of the estimated and real rewards (and the post-decision states) for one 365-day run is depicted in Fig. 5.7. In the model, the post-decision states capture the impact of a decision on the bots that survive a defense placement decision. As a result, the RL model is able to *control* the evolution of a botnet within the network and thus, perform better than its alternatives.

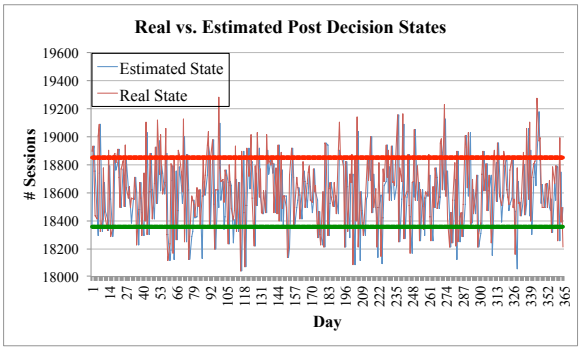


(a) Real vs. Estimated Rewards

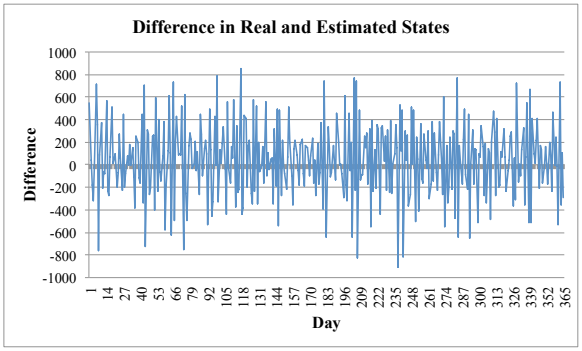


(b) Difference between Estimated and Real Rewards

Figure 5.7: Real and Estimated Rewards at switch SW_2



(a) Real vs. Estimated Post-Decision States



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

Figure 5.8: Real and Estimated Post-decision States at switch SW_2

5.6 Conclusions

In this chapter, we addressed the challenge of reducing the lifetime of stealthy botnets within a network. To this end, we adopted a defense-in-depth approach to combine the proactive nature of honeypots to detect lateral movement of bots along with the reactive nature of botnet detection mechanisms to detect persistent bots. In a resource-constrained environment, we proposed a RL-based model to place the limited number of honeypots

and monitors with an objective of maximizing the number of bots detected and removed from the network. We provided a proof-of-concept of the proposed approach, and studied its performance in a simulated environment. The results show that the RL-based approach performs better than its static or proactive strategies in protecting a network against stealthy botnets.

Chapter 6: A Moving Target Defense Approach to Mitigate DDoS Attacks in Proxy-Based Architectures

6.1 Introduction

Existing moving target-based architectures [28–30] leverage cloud environments to host a small set of active proxies. Incoming connection requests are validated by a well-provisioned lookup server which then redirects each authorized user to one of the secret active proxies to serve the user's subsequent requests. When under attack, a central server instantiates new proxies and clients associated with attacked proxies are *moved* to these newly instantiated proxies. To weed out compromised clients (assumed to be insiders), existing architectures shuffle the clients before assigning them to new proxies such that the insiders are eventually isolated from the innocent clients. Existing architectures assume that insiders persist in the system during an attack and hence, initiate the movement only *after* the attack has occurred. Such a reactive mechanism presents weaknesses that can be exploited to diminish the protection offered by these architectures.

To illustrate current limitations, we first present a new type of attack – the *proxy harvesting attack* – which can be used to collect information about a possibly large number of proxies before launching a massive DDoS attack against all known proxies. We show that state-of-the-art solutions are vulnerable to this attack. Next, we present a simple attacker isolation technique, BIND-SPLIT, to counter proxy harvesting attacks. We prove that, under the described attack model, the number of users that are affected by a DDoS attack can be minimized when a system is utilizing BIND-SPLIT. One limitation of BIND-SPLIT is that it requires the lookup server to maintain a static mapping between clients and proxies, which may lead to an uneven load distribution across proxies, thereby adversely affecting a system's performance. To address this issue while simultaneously reducing the impact

of DDoS attacks, we propose a lightweight defense technique, PROTAG, which proactively reconfigures the attack surface by periodically replacing one or more proxies and reassigning clients to new proxies. Finally, we show how BIND-SPLIT and PROTAG can be combined in a hybrid approach which integrates proactive cyber defense and attacker-isolation.

6.2 Threat Model

In this section, we briefly describe the threat model we consider in our work. We primarily focus on protecting Internet services that require client authentication, such as online banking and e-commerce portals. In our threat model, the attacker employs bandwidth-based or volumetric DoS attacks – such as SYN flood and DNS amplification – to disrupt the availability of target services. We assume a persistent attacker possessing sufficient capabilities (e.g., controlling a well-provisioned botnet) to simultaneously attack multiple proxies and any subsequent new proxy that may be spawned to mitigate the impact of the attack. Such repeated attack behavior using botnets is one of the emerging trends in DDoS attacks [25,93]. In a typical attack, bots spoof the IP addresses of traffic generated to congest network links of the target network. However, we assume that the attacker does not have sufficient capabilities to congest backbone links such as ISP networks or cloud infrastructures. In addition to leveraging a botnet, attackers may compromise the credentials of legitimate clients or eavesdrop on legitimate client's network connections and use them as *insiders*. They can then use this capability to learn the location of secret proxies and feed this information to the botnet in order to launch a DDoS attack. We assume that the number of such insiders is much smaller than the number of legitimate clients served by the target system.

6.3 Limitations of Current MTD Architectures

Existing moving target architectures [28–30] hide the location of an application server behind a layer of proxies. These proxies receive requests on behalf of the application server and

forward requests (responses) from (to) the clients. Figure 6.1 gives an overview of state-of-the-art moving target-based architectures. Essentially, they include three main components: a well-provisioned lookup server, proxy servers, and the application server. A description of each component is provided below.

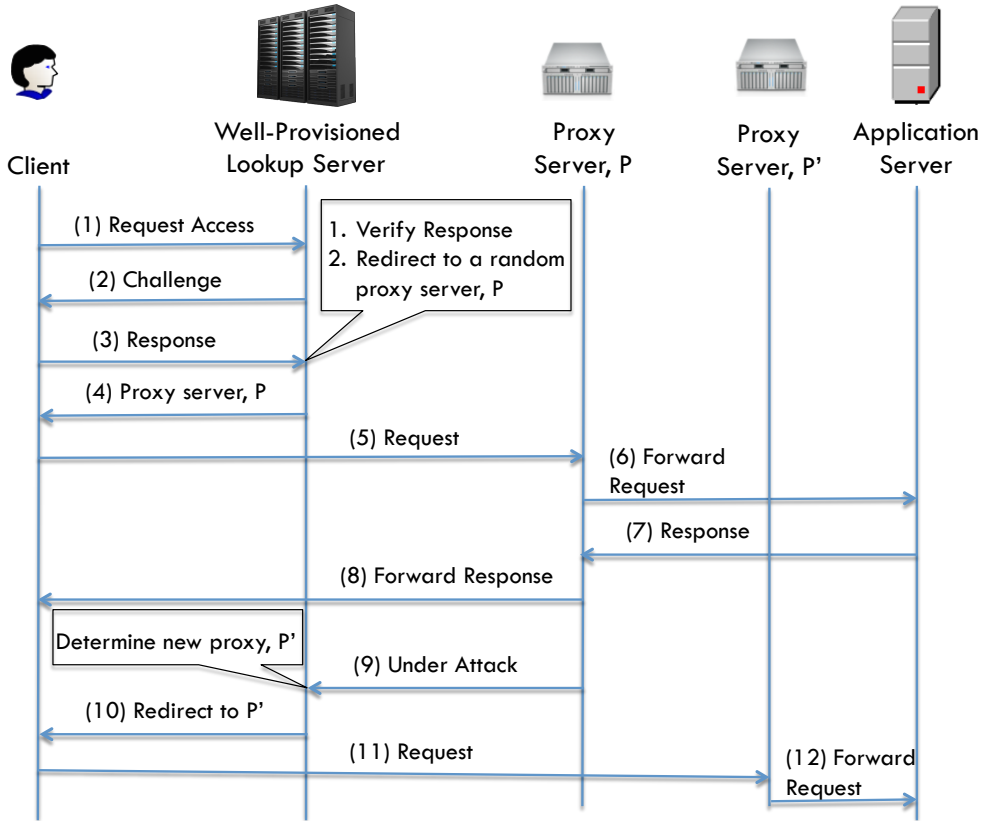


Figure 6.1: Overview of Moving Target-based Architecture

Well-Provisioned Lookup Server: To communicate with the application server, the client sends a request to a well-provisioned lookup server (step 1). The lookup server responds to the client's request by issuing a challenge (step 2) such as user credentials [28] or Proof-of-Work (PoW) [30]. These challenge-response mechanisms aim to filter illegal clients

who are either unauthorized to use the system or impersonate a legitimate client by IP spoofing. Upon solving this challenge (step 3), the client is redirected to a random active proxy server (step 4) that relays requests from the client to the application server. To ensure that the lookup server (whose IP address is public) is not susceptible to DoS attacks, existing architectures employ either a well-provisioned server – such as Content Distribution Networks [30] or a load-balancing DNS [29] – or leverage existing PoW schemes to force clients to solve cryptographic puzzles before they can consume the lookup server's resources.

Proxy Servers: Existing architectures assume the availability of a large pool of proxy servers – with only a small number of them active at any point in time. The IP addresses of active proxy servers are kept *secret*, i.e., they are not disclosed to clients who are unable to solve the challenge issued in step 2. *Authorized* clients contact the corresponding proxy server which in turn relays requests (responses) to (from) the application server (steps 5 through 8). As the location of the application server is known only to these proxy servers, the attack surface shifts from the application server to the active proxies. Existing architectures assume that proxies are equipped with a detection mechanism that enables them to detect an attack. When attacked, the proxy servers first inform the lookup server (step 9). Then, the lookup server instantiates new proxy servers and redirects clients associated with the attacked proxies to the new proxies (Step 10). Additionally, an attack on a secret proxy implies that one of the clients connected to the attacked proxy is acting as an insider and divulging the proxy address to the attacker. Therefore, before redirecting the victims to new proxies, the lookup server uses a client-to-proxy shuffling strategy to segregate innocent clients from insiders and allocates them to different proxies. As a result of this shuffling process, the number of innocent clients impacted during subsequent flooding attacks is reduced.

Application Server: The application server hosts one or more services and maintains a state for each client connected to it. As the application server stores all the session information, the proxies are lightweight and only implement a simple traffic indirection logic to relay requests/responses.

Existing moving target-based architectures are limited by their threat model, which assume that insiders (i.e., malicious clients able to solve the challenge) *persist* in the network during an attack. Such an assumption is necessary to facilitate effective isolation of innocent clients from insiders. However, under a more general threat model, in which the behavior of the insiders is unknown, we show that it is possible to circumvent the existing moving-target based defenses. To this end, we present a novel attack, called *proxy harvesting attack*, and propose an architecture to counter this attack.

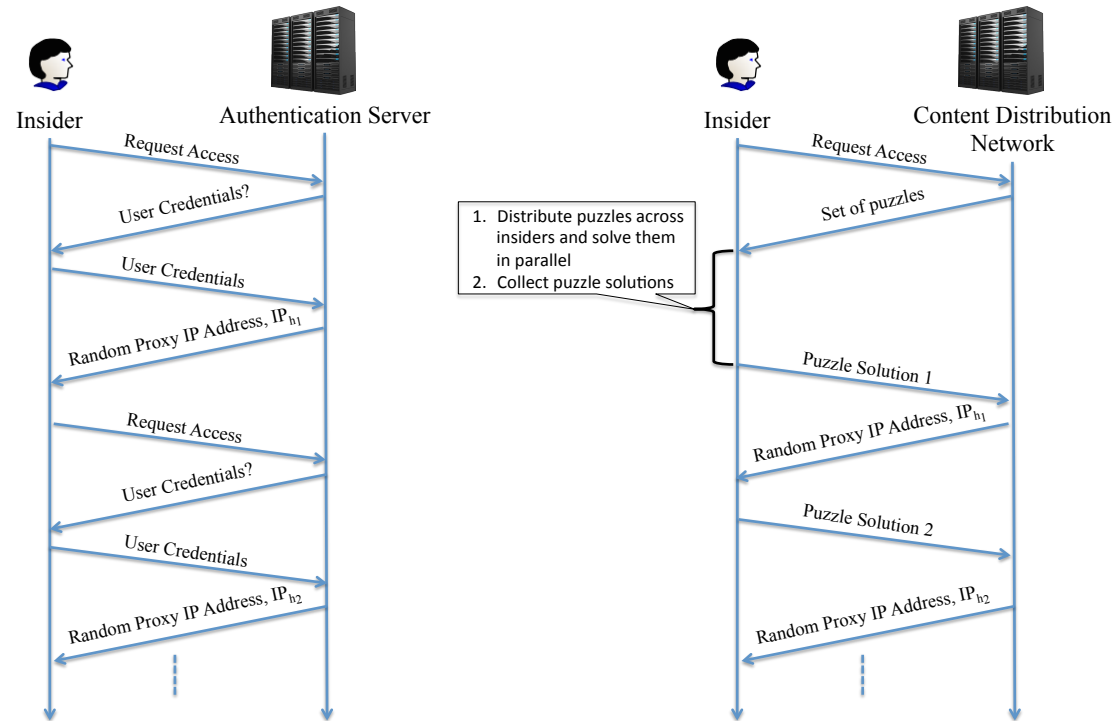
6.3.1 Proxy Harvesting Attack

In this section, we present the *proxy harvesting attack*. The attacker's goal is to collect IP addresses of as many proxy servers as possible before launching a coordinated attack against multiple proxies. We make no assumption regarding an insider's behavior. In particular, we do not assume that the insiders persist in the system during or after a DDoS attack.

In MOTAG, the proxy harvesting attack works as shown in Figure 6.2a. After an insider authenticates, the authentication server assigns it to one of the active proxy servers at random. Through this process, the insider learns the IP address of a secret active proxy. Then the insider authenticates again with the authentication server and learns a new IP address. Ideally, this process is repeated until the IP addresses of all active servers are collected.

For DoSE, the proxy harvesting attack can be modified as shown in Figure 6.2b. An insider first collects a set of puzzles from the CDN. A benign client should choose one of the puzzles at random and solve it. Upon submitting its solution to the CDN, the client will be redirected to a secret proxy. However, an insider, after collecting the set of puzzles, can distribute them to other insiders and solve multiple puzzles in parallel. The insider can then collect these solutions and submit them to learn the location of multiple secret proxies.

In both architectures, once all or a significant number of active proxies have been discovered, the insider will outsource this information to external attackers (such as botmasters)



(a) Proxy Harvesting Attack against MOTAG

(b) Proxy Harvesting Attack against DoSE

Figure 6.2: Proxy Harvesting Attack on existing moving target-based architectures

to launch a coordinated DDoS attack against all known active proxies. As mentioned earlier, when active proxies are under attack, the lookup servers will move affected clients and shuffle them among the new proxies to isolate the insider. However, if the insider is not present in the system, the shuffling operation does not provide any advantage in identifying the insider and only increases the overhead for the defender.

6.3.2 Analysis of the Proxy Harvesting Attack

In this section we analyze the proxy harvesting attack against MOTAG. For the sake of brevity, we omit the analysis for DoSE, which, however, is straightforward to conduct similarly to what we show below for MOTAG.

In MOTAG, the insider needs to interact only with the lookup server (here, authentication server) to harvest IP addresses of active proxy servers. As the lookup server assigns the clients to proxy servers randomly, the insider may be assigned to a previously known proxy server. The number of requests that must be made by an insider to discover all the IP addresses can be modeled as a classical variation of the Coupon Collector's Problem [94]. Hence, if there are N active proxy servers, the expected number of requests the insider must make to the lookup server is $N \cdot \log(N) + \gamma \cdot N + o(1)$ where $\gamma \approx 0.5772156$ is the Euler-Mascheroni constant and $o(1) \approx 0.5$. Therefore, if there are 50 active proxies, on average the insider needs to make 225 requests to learn all the IP addresses. It should be noted that, when multiple insiders collude to harvest IP addresses, the expected number of requests per insider decreases.

In practice, insiders may not know the exact number of proxies that are active at a given time, but they may know the range of IP addresses used by active proxies. Additionally, if an insider makes a large number of requests to the lookup server, an intrusion detection system may flag the situation as suspicious. Therefore, the attacker needs to determine the optimal number of authentication requests to strive a balance between stealthiness and harvesting enough proxies. We can model this problem as an optimization problem with the objective to maximize the number of active proxies discovered while simultaneously minimizing the likelihood of detection. Using the dynamic programming formulation in [95], the Bellman equation yielding the optimal stopping criterion is

$$A(r, n) = \min\{D(n+1), c + p_{r,n} \cdot A(r+1, n+1) + (1 - p_{r,n}) \cdot A(r, n+1)\} \tag{6.1}$$

where,

- $A(r, n)$ is the expected cost incurred by the insider to learn r distinct IP addresses by making n requests;
- $D(n)$ is the expected cost incurred by the insider when the attack is detected after making n requests.;
- c is the cost of a single request to the lookup server;
- $p_{r,n}$ is the conditional probability of discovering a new IP address given that r distinct IP addresses were discovered from n requests. From [95], $p_{r,n} = \frac{C(r+1,n+1)}{C(r,n)}$ where $C(r, n) = \sum_{j=0}^{\infty} \frac{(r+j)!}{j!(r+j)^n}$

As the number of IP addresses harvested increases with the number of requests to the lookup server, intuitively Eq. 6.1 maximizes the number of requests, n, made an insider subject to the constraint that the total cost of making n requests does not exceed the cost of detection, $D(n)$. Therefore, based on Eq. 6.1, the optimal probing strategy is as follows: if the insider detection cost $D(n)$ is less than the cost to discover new proxies, then make new request, otherwise stop.

6.4 BIND-SPLIT Strategy

The proxy harvesting attack exploits the lookup server's random assignment scheme – i.e., an incoming client request is randomly assigned to one of the active proxies – to discover multiple proxies before launching an attack. One of the approaches to overcome the proxy harvesting attack is to maintain a predefined mapping of clients to proxies, thus limiting the number of IP addresses that can be harvested by insiders. In this section, we present BIND-SPLIT, a simple, yet efficient, client-to-proxy mapping strategy that can be used to isolate insiders even if they do not persist in the system.

In the BIND-SPLIT strategy, the lookup server is configured to maintain a static mapping between clients and active proxies. Let $N = C_p \cdot P_0$ be the total number of clients with each one of the P_0 proxy servers initially serving C_p clients. Each client has a binding to a

particular server, which persists even if the client logs out and logs back in again. Also, let I be the number of insiders wherein each insider can only discover the IP address of the server to which it is currently assigned. If the goal of the defender is to minimize the total number of times any server can be attacked, the BIND-SPLIT strategy we propose appears to be asymptotically optimal. We assume the defender does not have sufficient server resources to preemptively assign every client its own server, which clearly would minimize attacks. Under the BIND-SPLIT strategy, clients remain bound to their assigned server until that server is attacked. When any server is attacked, the attacked server is shut down and two new servers are spawned in its place. The clients from the attacked server are split into two equal-size groups and migrated to the new servers. If the attacked server has only one client, then that client must be an insider.

We analyze BIND-SPLIT by considering a tree T. Each vertex of T represents a proxy spawned by BIND-SPLIT. Initially T consists of a root note, with edges to each of the initial P_0 proxies. When an attack occurs on some proxy p, we augment T by creating two new nodes, corresponding to the new proxies spawned by BIND-SPLIT. We make these nodes descendants of the attacked proxy. Notice that a node in T is only a leaf node if the proxy corresponding to it is never attacked. Therefore, to bound the total number of proxies attacked by I insiders, it suffices to bound the total number of non-leaf nodes in T. Now consider the proxies at some depth $d > 0$ in T. We argue that there can be only I non-leaf nodes at depth d. By definition of BIND-SPLIT, the clients served by two different proxies at depth d must be disjoint. Therefore at most I of the proxies at depth d can contain an insider, and so at most I proxies at depth d can have children. It's clear that T can also have depth at most $O(\log C_p)$, and therefore contains at most $O(I \cdot \log C_p)$ non-leaf nodes.

The maximum number of proxies used by the BIND-SPLIT strategy is $O(I \cdot \log C_p)$. This is the same order of growth as the number of attacks. The simple reason is that each time there is an attack on a server, the server is retired and two new servers spawned, increasing the server count by one. Note that, if BIND-SPLIT uses too many proxies to be practically applicable in some settings, it is possible to modify it to use fewer proxies. Specifically, the

defender can merge proxies that have not been attacked for a long time. This modification has the drawback of increasing the time to isolate an attacker on the merged proxies, in case the merged proxies did contain an insider and the new proxy is later attacked.

6.5 Proactive Proxy Migration Architecture

A limitation of BIND-SPLIT is the tight coupling of clients to proxies which may lead to sub-optimal load distribution across proxies. Furthermore, the effectiveness of existing moving target-based architecture against DDoS attack degrades in the presence of proxy harvesting attacks: moving and shuffling clients among proxies occur *only after* a DDoS attack has been detected. To strike a balance between the performance and service availability, we propose PROTAG (PROactive proxy-based moving TArGet architecture). Inspired by Fast-Flux Service Networks (FFSN) – a popular network architecture used by botnets, phishers and spammers to provide high availability for their services [96] – we periodically migrate clients to new proxies irrespective of whether an attack has been detected. Such proactive movement would disrupt the reconnaissance efforts of insiders by invalidating all or part of the information they were able to gather. In fact, as implied by our reasoning in Section 6.3.1, in order to remain stealthy, attackers will first gather IP addresses of multiple proxies over a possibly extended period of time, and only at a later stage they will use that information to launch a DDoS attack. If, by that time, many clients have been migrated to new proxies, the attack will be either disrupted or at least mitigated. Finally, after an attack, PROTAG harnesses the insider isolation principle of BIND-SPLIT strategy to partition the clients across proxy pools and eventually isolate the insiders.

Within the existing MOTAG architecture, a proactive proxy movement strategy can be defined in terms of two primary factors: proxy selection and movement frequency. Proxy selection involves determining *which* proxies to replace, whereas a strategy's movement frequency determines *when* to replace the proxies. In this work, we consider a simple, yet effective, strategy in which a subset of active proxies is chosen uniformly at random for replacement. Effectiveness of this strategy is evaluated by computing the expected number

of active proxies that can be discovered in comparison with existing architectures where no proactive action is implemented.

6.5.1 Proxy Movement

Let $\mathcal{P} = \{P_1, P_2, ..., P_K\}$ be the set of all available proxies. We assume that, at any time $t \in T$ – where T is a set of discrete time points – only k (with $k < K$) proxies are active, and we use $\mathcal{P}_t = \{P_{i_1^t}, P_{i_2^t}, ..., P_{i_k^t}\}$ – where $i_j^t \in [1, K]$ for all $j \in [1, k]$ and all $t \in T$ – to denote the set of active proxies at time t.

In PROTAG, the authentication server initiates the movement procedure every Δt time units – we refer to Δt as the *reconfiguration period* and to $f = \frac{1}{\Delta t}$ as the *reconfiguration frequency*. At that time, the authentication server chooses a subset of m active proxies $\mathcal{P}_r \subseteq \mathcal{P}_t$, with $|\mathcal{P}_r| = m$, uniformly at random. Additionally, the authentication server chooses m available (but currently not active) proxy servers $\mathcal{P}'_r$ uniformly at random from the set $\mathcal{P} \backslash \mathcal{P}_t$. All the clients associated with a proxy in $\mathcal{P}_r$ are then migrated to a proxy in $\mathcal{P}'_r$.

The authentication server maintains each client-proxy association for a period of time Δt^*, which we refer to as the *association period*. If a client assigned to a proxy P_i closes the connection with P_i and re-authenticates with the authentication server before Δt^* time units since it was originally assigned to P_i, then the authentication server assigns it to the same proxy. This mechanism imposes an upper bound on the rate at which an insider may discover new proxies. In fact, the insider may not discover more than $\frac{1}{\Delta t^*}$ proxies per time unit by doing repeated authentication requests.

The time interval Δt^* is determined by the network administrator by taking into account the desired load per proxy (in terms of number of clients) and the expected arrival rate of clients, thereby ensuring that proxies are not overloaded. The frequency with which the proxies are moved depends on the number of active proxy addresses that an insider can learn in Δt time units. In other words, if proxies are proactively moved using this strategy every Δt time units, then an insider would have discovered at most $\lceil \frac{\Delta t}{\Delta t^*} \rceil$ proxies. Determining

the value of Δt is critical to ensure that the strategy is effective: the smaller the value of Δt, the higher the insider's effort to harvest proxies. To this end, we derive the expected number of active proxies that can be discovered by multiple insiders as a function of the number of authentication requests and the reconfiguration interval Δt.

6.5.2 Analysis of PROTAG

In this section, we model the problem as a game between insiders and authentication server, such that, at each round, the insiders send multiple independent probes to discover proxies while the authentication server moves multiple proxies. Let I be the number of insiders and m be the number of active proxies moved by the authentication server at each round of the game. For the sake of analysis, we assume that the reconfiguration period Δt is a multiple of the association period Δt^*, that is $\Delta t = z \cdot \Delta t^*$ where $z \in \mathbb{Z}^+$ is a positive integer. Therefore, at each round, the authentication server first moves m of the k active proxies – using the strategy described earlier – followed by each insider making z authentications at times that are multiple of Δt^*. For the purpose of this analysis, we consider a finite number $q \in \mathbb{Z}^+$ of rounds of this game. The defender moves m proxies at times $0, \Delta t, 2 \cdot \Delta t, \ldots, (q-1) \cdot \Delta t$, and, for each round $r \in [1, q]$, each insider makes an authentication request at times $(r-1) \cdot \Delta t, (r-1) \cdot \Delta t + \Delta t^*, \ldots, (r-1) \cdot \Delta t + (z-1) \cdot \Delta t^*$. The total number of authentication requests made by all the insiders during the q rounds is then $n_{tot} = q \cdot z \cdot I$. As each probe at times $(r-1) \cdot \Delta t + x \cdot \Delta t^*, x \in [0, z-1]$ is assigned an active proxy uniformly at random – independently of the specific client – the expected number of distinct proxies discovered by I insiders at the end of q rounds by making z probes at each round is equal to the number of distinct proxies discovered by an insider making $z \cdot I$ probes at each round.

An example of the timeline of an authentication server-insider game is shown in Figure 6.3. In this example, $q = 3$, $z = 3$ and $I = 1$. The insider makes a move every $\Delta t^* = 2$ time units while the authentication server makes a move every $\Delta t = z \cdot \Delta t^* = 6$ time units.

Let X_r be a random variable representing the number of distinct active proxy IP addresses discovered at the end of round $r \in [1, q]$. The effectiveness of the strategy can be evaluated

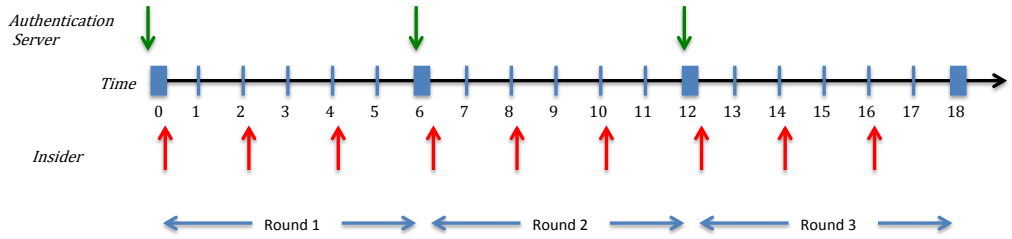


Figure 6.3: Example of authentication server-insider game

by computing the expected value of X_q, that is the expected number of distinct proxies that the insider discovers by the end of the q-th round of the game. This can be computed as $E[X_q] = \sum_{i=1}^{k} (i \cdot Pr[X_q = i])$. For the first round, the probability that the insider discovers x proxies out of k active proxies after $z \cdot I$ authentications, $Pr[X_1 = x]$, is given by the number of ways to arrange x distinct proxies from k active proxies across $z \cdot I$ authentications divided by the number $k^{z \cdot I}$ of possible outcomes. Therefore, the probability distribution for X_1 is given by

$$Pr[X_1 = x] = \frac{P_{(k,x)} \cdot S^{(x)}_{(z \cdot I)}}{k^{z \cdot I}}, \; x \in [1, \min(z \cdot I, k)] \tag{6.2}$$

where,

- $S^{(r)}_n = \sum_{j=1}^{r} \frac{(-1)^{r-j} j^n}{j!(r-j)!}$ is a Stirling number of the second kind, which is the number of ways to divide a set of n objects into r nonempty subsets. Intuitively, this captures the notion that multiple authentication requests may yield the same IP address.

- $P_{(k,x)}$ is the number of ways to arrange x-element subsets from a k-element set.

Moving proxies invalidates knowledge acquired by the insider. Let A_r be a random variable which represents the number of known proxies that are removed when the authentication server moves m proxies at round r. Also, let D_r be the maximum number of active

proxies that the insider might have already discovered by the end of round $r-1$. Such number cannot exceed the number $x + A_r$ of proxies discovered at the end round r. Therefore, the value of D_r depends on the outcome of A_r, that is

$$D_r(A_r = l) = \min\left((r-1)\cdot z, x + l, k\right) \tag{6.3}$$

Hence, for $x \in [1, \min(r \cdot z \cdot I, k)]$, the probability distribution of X_r is given by

$$Pr[X_r = x] = \sum_{y=1}^{D_r(A_r=l)} Pr[X_r = x | X_{r-1} = y] \cdot Pr[X_{r-1} = y] \tag{6.4}$$

Then, combining Eq. 6.3 and Eq. 6.4, we can write

$$Pr[X_r = x] = \sum_{l=0}^{m} \sum_{y=\max(l,1)}^{D_r(A_r=l)} \left(Pr[X_r=x|X_{r-1}=y, A_r=l] \cdot Pr[A_r=l|X_{r-1}=y] \cdot Pr[X_{r-1}=y]\right) \tag{6.5}$$

In Eq. 6.5, $Pr[A_r = l | X_{r-1} = y] = \frac{C_{(y,l)} \cdot C_{(k-y,m-l)}}{C_{(k,m)}}$. If the insider has discovered y and x proxies by the end of rounds $r-1$ and r respectively, then during the $z \cdot I$ authentications at round r, the insider should have discovered $x-(y-l)$ new proxies where $l \in [0, m]$. However, the insider may encounter s already known proxies, where $s \in [0, y-l]$. Therefore, for a given s, $Pr[X_r = x | X_{r-1} = y, A_r = l]$ is the number of ways of arranging $x-(y-l)$ distinct proxies from $k-(y-l)$ unknown active proxies and s distinct proxies from $y-l$ discovered active proxies across $z \cdot I$ authentications out of $k^{z \cdot I}$ possible outcomes. Hence,

$$Pr[X_r = x | X_{r-1} = y, A_r = l] = \frac{\sum_{s=0}^{y-l} P_{(k-(y-l),x-(y-l))} P_{(y-l,s)} C_{(x-(y-l)+s,s)} S_{z\cdot I}^{(x-(y-l)+s)}}{k^{z \cdot I}} \tag{6.6}$$

where,

- $S_n^{(r)}$ is a Stirling number of the second kind

- $P_{(n,r)}$ is the number of ways of arranging r-element subsets of an n-set
- $C_{(n,r)}$ is the number of ways of choosing r objects from a set of n objects

By substituting Eq. 6.6 in Eq. 6.5 and recursively solving for $r = Q$ with Eq. 6.2 as the base case, we derive $Pr[X_q = x], \forall x \in [1, k]$.

6.6 Insider Isolation Algorithm

PROTAG significantly disrupts an attacker's efforts to discover the IP addresses of active proxies and, hence, reduce the impact of an attack. The impact of a DDoS attack can be defined as the average number of innocent clients who suffer from a degraded quality of service during the attack. In the proposed architecture, the impact of a DDoS attack on a given number of active proxies, on average, remains constant as all the clients are distributed uniformly at random across the same pool of active proxies. To reduce the impact of subsequent DDoS attacks, we propose a simple insider isolation algorithm which incorporates the isolation principles of the BIND-SPLIT strategy to partition clients into different proxy pools and eventually, isolate insiders into proxy pools with fewer innocent clients.

In this algorithm, the lookup server maintains a mapping for each client to a set of proxies. Before the onset of an attack, all the clients are associated with a single pool of active proxies. After an attack, for each attacked proxy pool $\mathcal{P}$, the lookup server will first spawn two new proxy pools $\mathcal{P}_1$ and $\mathcal{P}_2$ with the same size as the attacked proxy pool. Next, the server will randomly partition the clients associated with the attacked proxy pool into two groups $\mathcal{C}_1$, $\mathcal{C}_2$ and re-assign clients in $\mathcal{C}_i$ to proxy pool $\mathcal{P}_i$, $i = 1, 2$. For subsequent connection requests by any client in $\mathcal{C}_i$, the lookup server will re-direct it to proxy pool $\mathcal{P}_i$, $i = 1, 2$.

6.6.1 Analysis

We analyze the proposed insider isolation algorithm and provide theoretical upper bound on the number of attacks and the number of proxies necessary to completely isolate the insiders. To analyze the algorithm, consider a tree $\mathcal{T}$ where the nodes of the tree $\{\mathcal{P},\mathcal{C}\}$ represent the identity of the proxy pool $\mathcal{P}$ and the set of clients $\mathcal{C}$ associated with it. An edge from $\{\mathcal{P},\mathcal{C}\}$ to $\{\mathcal{P}',\mathcal{C}'\}$ exists if and only if proxies in $\mathcal{P}$ were attacked and the lookup server spawned proxies in $\mathcal{P}'$ and assigned clients in $\mathcal{C}' \subset \mathcal{C}$ to proxies in $\mathcal{P}'$ where $|\mathcal{C}'| = \frac{|\mathcal{C}|}{2}$.

It can be seen that at any depth d all sets of clients at depth d are disjoint and hence, the resulting tree $\mathcal{T}$ has a total depth of $\log(N)$, where N is the total number of clients. The depth of the tree provides an upper bound on the number of DDoS attacks experienced by a client before isolating the insiders. In other words, the number of attacks experienced by any user of the system is bounded by $\log(N)$. Furthermore, the number of proxies spawned by the algorithm can be calculated by counting the number of leaf nodes in $\mathcal{T}$. As a base case, consider a scenario where there is exactly one insider in the system among N' clients associated with a proxy pool $\mathcal{P}'$. In this case, the resulting tree $\mathcal{T}'$ will be a skewed binary tree with $\log(N')$ leaf nodes. When there are I insiders, then in the worst case, after each attack, the insiders will be evenly distributed across all the proxy pools. Consider the set $\mathcal{F}$ of subtrees rooted at nodes at depth $\log(I)$. The root of each subtree $\mathcal{T}' \in \mathcal{F}$ represents a proxy pool with $\frac{N}{I}$ innocent clients and one insider. As there are at most I such subtrees, the maximum number of proxies is $\log\left(\frac{N}{I}\right) \cdot I \cdot |\mathcal{P}_i|$ where $|\mathcal{P}_i|$ is the number of proxies in the initial active proxy pool.

6.7 Simulation Results

In this section, we study the proxy harvesting attack, validate the proactive movement and the insider isolation algorithm presented in the previous section through simulations.

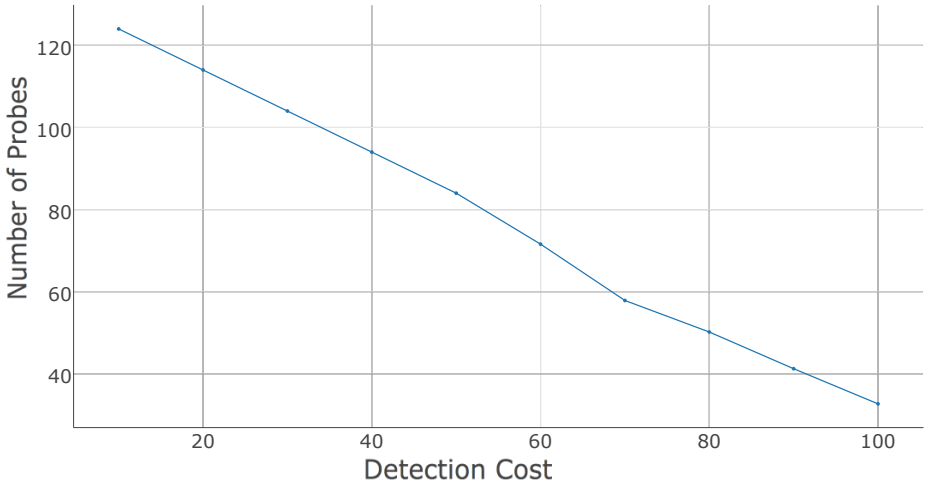


Figure 6.4: No. of probes by an insider for different detection costs

6.7.1 Proxy Harvesting Attack

In order to illustrate the trade-off in the attacker's actions in MOTAG-based architectures (whether to authenticate or not), we simulated Eq. 6.1 for various values of the detection cost and number of active proxies. In the simulation, we assumed, $D(n) = P(n) \cdot d$, where $P(n) = (1 - e^{-(\alpha \cdot n)})$ is the probability that the attacker makes n probes and d is the cost for an insider to be detected. Such exponential distributions have been used in the past [97] to model the effort required by an attacker to successfully mount an attack. In our simulations, we set $\alpha = 0.05$. The cost of detection, d, and the cost of probing, c, were normalized by setting $d = z \cdot c, z \in \mathbb{Z}^+$. In our simulations, we assumed $c = 1$. Figure 6.4 shows the number of authentication probes as a function of the detection cost d. As expected, the number of authentication probes made by an insider decreases as the detection cost increases. The rationale behind this trend is that, as the detection cost d increases, the expected cost of insider detection $D(n)$ increases for every subsequent probe. This restricts the number of authentication probes for the insider. As shown in Figure 6.5, for lower detection costs, an

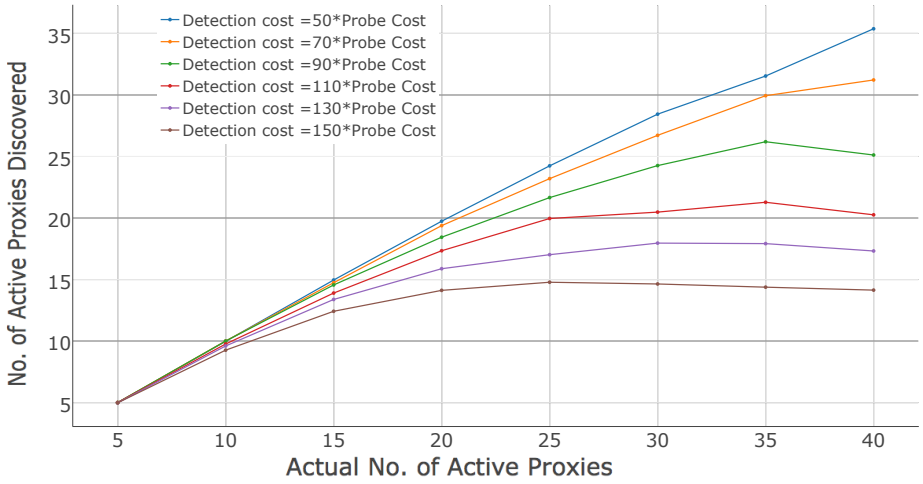


Figure 6.5: No. of discovered proxies vs. number of active proxies

insider can discover at least 70% of the active proxies and hence, can launch a *significant* DDoS attack against known proxies.

6.7.2 PROTAG

To assess the performance of PROTAG, we simulated Eq. 6.5 for 40 active proxies with different numbers of insiders. In our simulation, the insiders made a total of 100 probes. As discussed earlier, the number of proxies discovered by an attacker using I insiders is equivalent to the number of proxies discovered by one insider sending I probes. Figure 6.6 and Figure 6.7 depict the probability distribution of the number of proxies discovered by multiple insiders with and without applying PROTAG, where z is the number of authentications between consecutive proxy movements by the authentication server. As shown in the figures, when there is no proactive movement, there is a high probability of discovering most of the proxies with an expected value of 37. However, with proactive movement, the probability distribution *shifts* such that the probability of discovering more than 80% of the

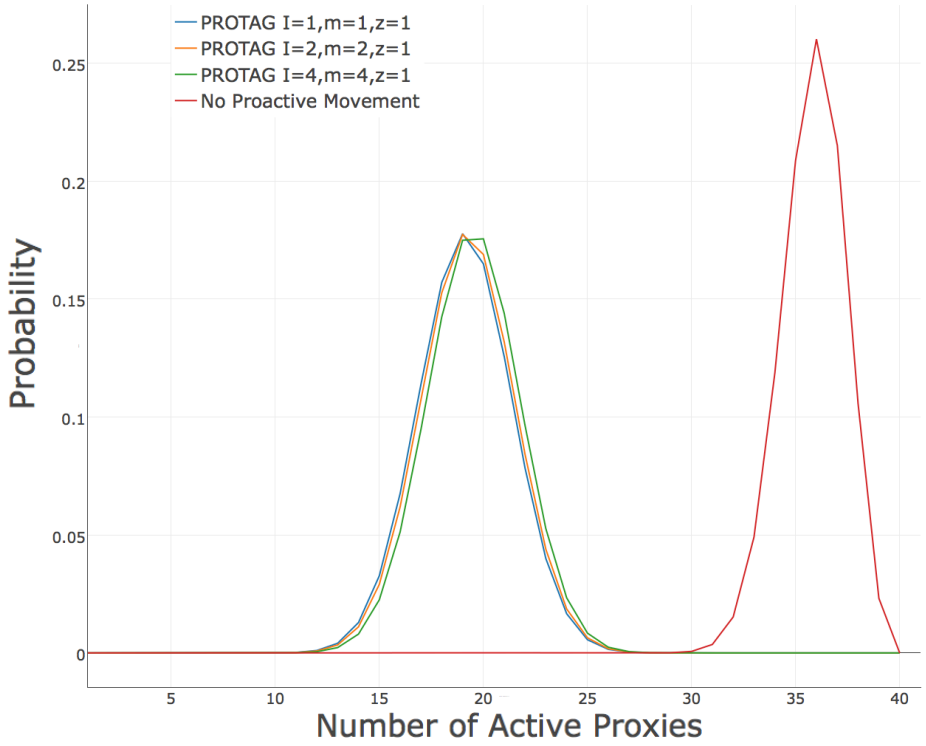


Figure 6.6: Probability distribution of the number of discovered proxies when authentication server moves $m = I$ proxies

proxies reduces significantly. In particular, for proactive movement, the expected number of proxies discovered is 20 and 26 respectively, reducing the number of discovered proxies by 46% and 30% respectively. Furthermore, we observed that by increasing the number of probes made by the insider, the probability distribution of the number of proxies discovered by an insider did not change.

On the other hand, if the authentication server moves multiple proxies ($m > 1$), the expected number of discovered proxies can be reduced. Figure 6.6 and Figure 6.7 depict the probability distribution of discovered proxies when the authentication server moves $m = I$ and $m = 0.5 \cdot I$ proxies, respectively. It can be seen that the expected number of discovered

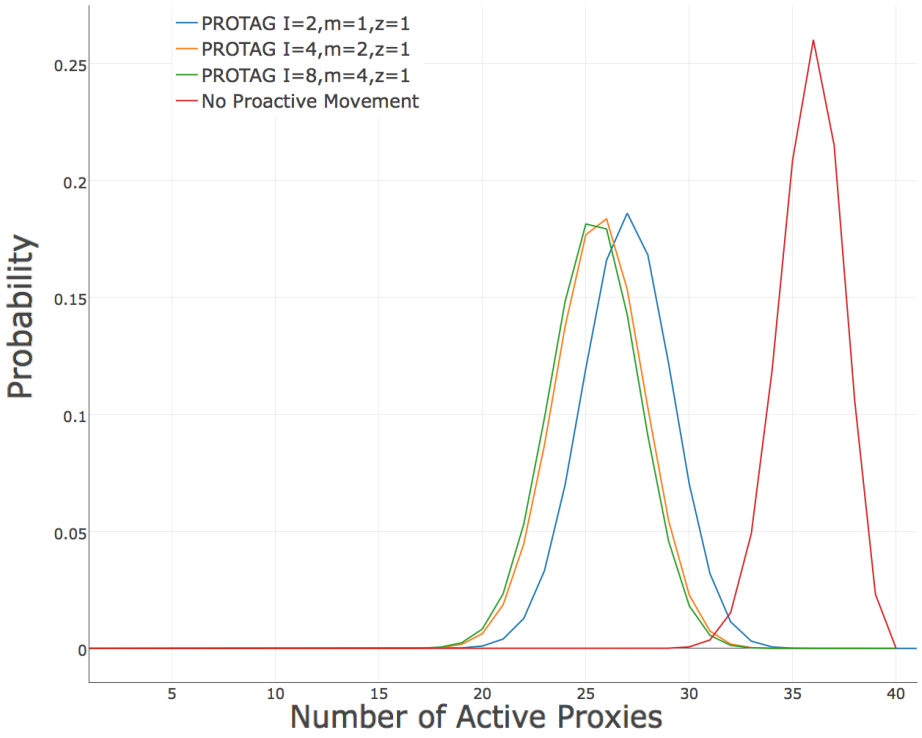


Figure 6.7: Probability distribution of the number of discovered proxies when authentication server moves $m = 0.5 \cdot I$ proxies

proxies can be restricted to the case when the authentication server moved only one proxy. These results also show that, if the cost of frequently moving a proxy is high, then the authentication server can move at a lesser frequency and compensate for the additional information gained by an attacker by moving multiple proxies (proportional to the number of authentication probes) at a later time.

6.7.3 Insider Isolation Algorithm

To study the performance of the proposed insider isolation algorithm, we considered a system serving 10,000 users with a varying number of insiders. In order to simulate the user's

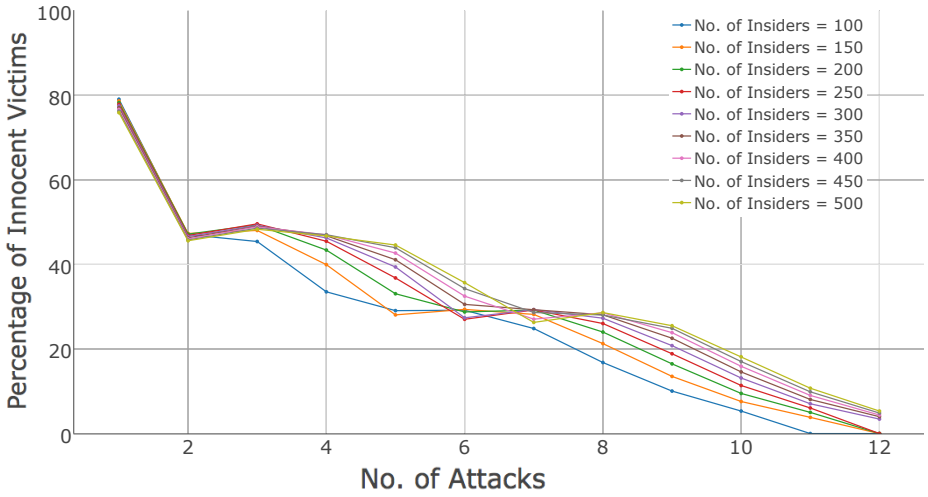


Figure 6.8: Percentage of innocent users affected during a sequence of attacks

behavior, in our environment, after every hour, the users (if not connected) log into the system with a probability of 0.8 and if connected, they disconnect with a probability of 0.5. These simulations were conducted for a period of 500 hours ($\sim$ 3 weeks of uptime) starting with a proxy pool of 2 active proxies. In our simulation, we assumed that the attacker's objective is to discover at least 50% of the active proxies before launching the attack. Against such a threat model, we studied the performance of the proposed algorithm to save at least 90% of innocent users with and without proactive movement. All simulations were repeated 30 times and the results were averaged over these independent trials.

As expected, the percentage of innocent users who are affected due to repeated DDoS attacks decreases with every subsequent attack (shown in Figure 6.8). As described in section 6.6, the insider isolation algorithm spawns new proxies and partitions clients across these proxies after an attack. As the number of insiders increases, the average number of proxies instantiated also increases as depicted in Figure 6.9. When deployed in a cloud environment such as Amazon EC2, the cost of on-demand instances to host the proxies

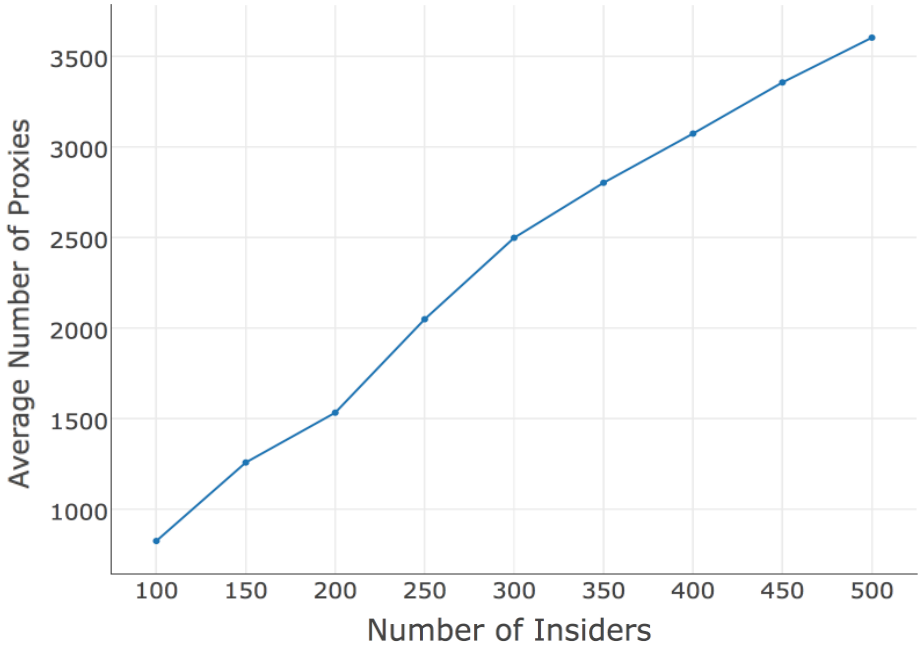


Figure 6.9: Average number of proxies required to isolate 90% of innocent users from insiders

(which is about $0.007 per hour for EC2 instances based on their spot pricing) would cost the enterprise less than $150 dollars (for 500 insiders) to mitigate attacks lasting for 100 hours and protect more than 90% of innocent users from subsequent attacks. To understand the impact of initial proxy pool size, we repeated the experiment with 10, 25 and 50 proxies and observed less than 5% decrease in the number of saved innocent users. In addition to the cost incurred from deploying the proxies, the users of the application incur an overhead cost due to frequent redirection to new proxies. As studied by researchers in [29], the clients are expected to experience less than 3 seconds overhead as a result of the redirection operation.

We also studied the impact of coupling PROTAG with the insider isolation algorithm. Here, we considered a movement strategy in which, from each active proxy pool, an active proxy was replaced on an hourly basis. As illustrated in Figure 6.10, PROTAG, on average,

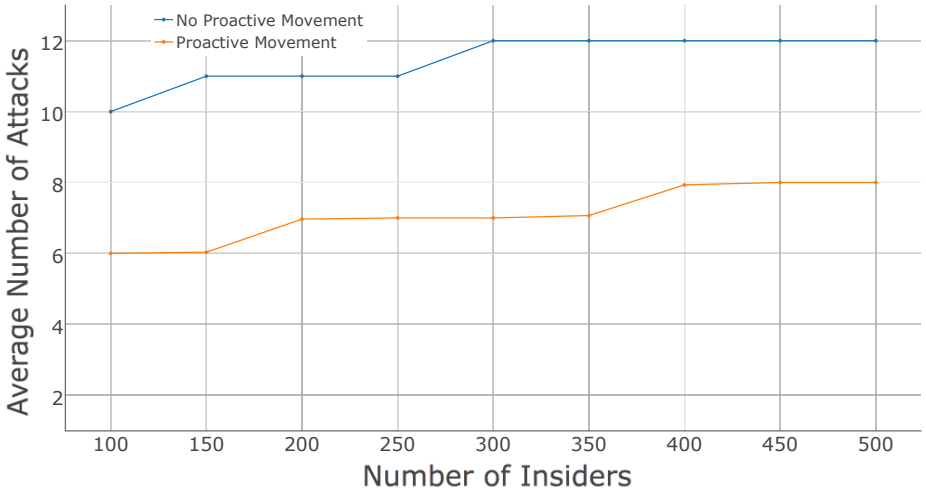


Figure 6.10: Average number of attacks with and without proactive movement strategy

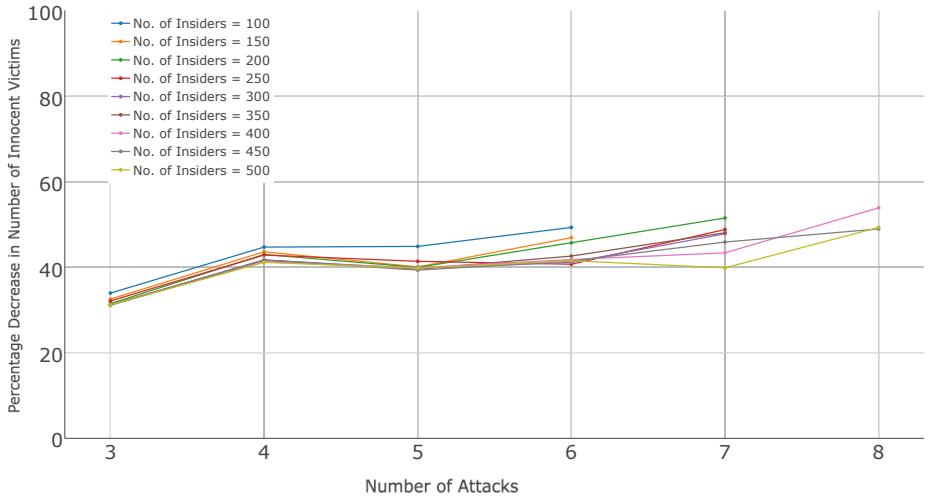


Figure 6.11: Percentage decrease in number of innocent victims

reduced the number of attacks observed during an experiment's lifetime by 4. As a result, it was observed that the average number of innocent users who are affected due to an attack decreased by 40% in comparison to the setting without PROTAG (shown in Figure 6.11).

6.8 Conclusions

In this chapter, we first presented a new type of attack, the proxy harvesting attack, and showed that state-of-the art proxy-based architectures are vulnerable to such a reconnaissance strategy. Next, to overcome the limitations of current approaches, we proposed a proactive strategy – namely PROTAG – to periodically replace one or more proxies and remap clients to proxies, with the goal of disrupting the attacker's reconnaissance efforts, and an insider isolation approach to mitigate ongoing attacks. We validated our solution in a simulated environment, and simulation results confirmed that PROTAG is effective in reducing the number of proxies an attacker can discover in a given amount of time as well as reduce the impact of a wave of DDoS attacks.

Chapter 7: Future Work

In this thesis, I have demonstrated the potential for ACD models and techniques to reverse the asymmetric advantage of an attacker – performing botnet-assisted malicious activities – in favor of the defender. In particular, we develop ACD techniques to detect long-term exfiltration campaigns and mitigate DDoS attacks.

While the proposed techniques have proved to be effective in a simulated environment, as part of our future work, we intend to deploy these techniques in a live network environment and test their effectiveness. In addition to this, there are several interesting directions for future work.

7.1 Advanced Adversarial Capabilities

In our threat model, we do not consider an adaptive adversary who takes actions in response to the monitoring strategy of the defender. For instance, the attacker may have the capability to discover which network nodes are being monitored by *probing* nodes in the network. In an enterprise network, a simple probing strategy could be to send malicious packets to a node and depending on whether or not the packet successfully reached the destination, the attacker can determine the node's monitoring state. At an ISP network level, an attacker can leverage probing strategies described by Shinoda et al. [64], and by Shmatikov and Wang [65] to identify the presence of passive detectors in a network. In these probing attacks, the attacker sends unique probes to nodes that are suspected of being monitored by a detector. If a node is monitored by a detector, then the corresponding alert will be published in a report that is publicly accessible. Once the attacker learns the location of the monitors, the attacker can relay the exfiltration traffic through a sequence of bots such the total volume intercepted by the monitors is minimized.

While probing reveals the location of monitors, it also generates alerts that can be observed by the defender. Therefore, developing a monitor placement strategy that considers information from the alerts can be casted in a game-theoretic framework through which we can study the Nash equilibria and the dominant strategies for the defender and the attacker. Along similar lines, McCarthy et al. [98] considered the problem of placing imperfect sensors on nodes within a network to detect malicious domains that are used in a DNS exfiltration campaign. To this end, they propose a POMDP model that accumulates noisy information from the sensors to guide the placement decision of sensors subject to cost constraints and routing policy constrains within the network.

7.2 Adversarial Machine Learning

With the rapid growth in the capabilities of a malware, defense mechanisms based on static analysis (ranging from simple firewall rules to static binary analysis) have become obsolete. This has paved the way to the rise of defense mechanisms that leverage machine learning algorithms to detect malwares based on behaviorial-analysis. For instance, Chapter 4 presented a detection mechanism that leveraged intrinsic characteristics in the network behavior of bots and Chapter 5 leveraged the spike in the network activity (i.e., number of host scans and number of outgoing sessions) to guide the placement of the defense mechanisms (honeypots and monitors).

In an environment where adversaries are aware of the learning algorithm, they could intentionally introduce noise to either cause a malicious flow be misclassified as a benign flow or bias the learning processing. For instance, to evade detection by DeBot, the bots could append random bytes of data to the update messages so that the corresponding malicious flows and the benign flows are clustered together. As a result, the similarity score of the compromised host and a benign host (in its neighborhood) will remain unaffected by the clustering processing. Such noises that increase the false positive and negative rate of machine learning algorithm are known as adversarial examples [99].

Similarly, the policy learnt by the RL model is dependent on the accuracy of baseline behaviors. Typically, the baseline behavior of a network is established prior to learning by analyzing the network traffic captured in the past. However, if the hosts were compromised during the baseline determination phase, the bots could *poison* the data by introducing network activity that would result in high baseline values [100]. Thus, during the learning and learnt phases of the reinforcement learning model, the bots could continue to operate below the established baselines. Developing learning algorithms that are robust to the noise or bias introduced by malwares is a potential direction for future work.

7.3 Combining Different ACD Techniques

In Chapter 5, we developed a reinforcement learning model to reduce the lifetime of botnets. The results indicated that combining honeypots and network-based detection mechanisms improves the overall security posture of the network. Building on this work, one of the potential directions of future work is to include the cost of cleaning and restoring a machine to its pristine state. In particular, as the cleansing operation for different machines may vary based on their availability or mission-criticality, the reinforcement learning model must be tuned to learn the optimal time to clean by performing cost-benefit analysis.

Furthermore, as individual ACD techniques can only partially protect a network, we plan to investigate the possibility of combining existing MTD mechanisms such as OS rotation with ACD mechanisms proposed in this thesis (such as dynamic monitoring). To this end, we developed a framework in [101] that takes a principled approach to study the effectiveness of combining different MTD techniques. Inspired by the idea of behind attack graphs [102,103], the framework first captures the relationship between a given set of MTD techniques and their impact on the information required for an attacker to successfully exploit a weakness. Based on this relationship, subject to cost constraints, the framework determines the optimal setting for the given set of MTDs that reduces the overall probability of a successful attack. As part of our future work, we intend to extend this framework and include network-wide ACD techniques to provide an effective network protection.

Bibliography

Bibliography

[1] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[2] OpenDNS, "The role of dns in botnet command & control," 2013.

[3] M. Fabian and M. A. Terzis, "My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging," in *Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets, Cambridge, USA*, 2007.

[4] J. Meisner, "Microsoft and financial services industry leaders target cybercriminal operations from zeus botnets," http://blogs.microsoft.com/blog/2012/03/25/microsoft-and-financial-services-industry-leaders-target-cybercriminal-operations-from-zeus-botnets/, March 2012.

[5] B. Labs, "Gameover zeus variants targeting ukraine, us," http://labs.bitdefender.com/2014/08/gameover-zeus-variants-targeting-ukraine-us/, August 2014.

[6] A. K. Sood, S. Zeadally, and R. J. Enbody, "An empirical study of http-based financial botnets," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 236–251, 2016.

[7] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed., ser. Advances in Information Security. Springer, 2011, vol. 54.

[8] S. Jajodia, A. K. Ghosh, V. S. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, 1st ed., ser. Advances in Information Security. Springer, 2013, vol. 100.

[9] A. Giani, V. H. Berk, and G. V. Cybenko, "Data exfiltration and covert channels," in *Defense and Security Symposium*. International Society for Optics and Photonics, 2006, pp. 620 103–620 103.

[10] K. Labs, "Kaspersky lab and itu research reveals new advanced cyber threat," http://usa.kaspersky.com/about-us/press-center/press-releases/kaspersky-lab-and-itu-research-reveals-new-advanced-cyber-threa, May 2012.

[11] D. Mcwhorter, "APT1: Exposing one of china's cyber espionage units," http://intelreport.mandiant.com/, 2013.

[12] P. Sweeney and G. Cybenko, "Identifying and exploiting the cyber high ground for botnets," in *Cyber Warfare*, ser. Advances in Information Security. Springer International Publishing, 2015, vol. 56, pp. 37–56.

[13] P. J. Sweeney, "Designing effective and stealthy botnets for cybet espionage and interdiction - finding the cyber high ground," Ph.D. dissertation, Thayer School of Engineering, Darthmouth College, 2014.

[14] S. S. Response, "W32. Duqu the precursor to the next stuxnet," 2011.

[15] K. L. Report, "The regin platform nation-state ownage of gsm networks," 2014.

[16] M. Marschalek, P. Kimayong, and F. Gong, "POS malware revisited - look what we found inside your cashdesk," 2014.

[17] D. SecureWorks, "Banking Botnets: The Battle Continues," https://www.secureworks.com/research/banking-botnets-the-battle-continues, February 2016.

[18] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection." in *Proceedings of the 17th USENIX Security Symposium*, vol. 5, no. 2, 2008, pp. 139–154.

[19] J. Zhang, X. Luo, R. Perdisci, G. Gu, W. Lee, and N. Feamster, "Boosting the scalability of botnet detection using adaptive traffic sampling," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 124–134.

[20] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a scalable system for stealthy P2P-botnet detection," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 1, pp. 27–38, 2014.

[21] J. Kwon, J. Lee, H. Lee, and A. Perrig, "PsyBoG: A scalable botnet detection method for large-scale dns traffic," *Computer Networks*, vol. 97, pp. 48–73, 2016.

[22] X. Hu, J. Jang, M. P. Stoecklin, T. Wang, D. L. Schales, D. Kirat, and J. R. Rao, "BAYWATCH: Robust beaconing detection to identify infected hosts in large-scale enterprise networks," in *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 2016, pp. 479–490.

[23] Mandiant, "M-Trends© 2015: A View from the Front Lines, Mandiant," https://www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf, 2015.

[24] K. Labs, "A single DDoS attack can cost a company more than $400,000," http://www.kaspersky.com/about/news/business/2015/A-single-DDoS-attack-can-cost-a-company-more-than-400000-dollar, January 2015.

[25] Arbor Networks, "Worldwide infrastructure security report," https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf, November 2015.

[26] D. G. Andersen, "Mayday: Distributed filtering for internet services," in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, Seattle, Washington, USA, March 2003, pp. 31–42.

[27] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," in *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, Pennsylvania, USA, August 2002, pp. 61–72.

[28] H. Wang, Q. Jia, D. Fleck, W. Powell, F. Li, and A. Stavrou, "A moving target DDoS defense mechanism," *Computer Communications*, vol. 46, pp. 10–21, 2014.

[29] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: A cloud-enabled DDoS defense," in *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, Atlanta, Georgia, USA, June 2014, pp. 264–275.

[30] P. Wood, C. Gutierrez, and S. Bagchi, "Denial of service elusion (DoSE): Keeping clients connected for less," in *Reliable Distributed Systems (SRDS), 2015 IEEE 34th International Symposium on.* IEEE, 2015, pp. 1–10.

[31] N. Falliere, "Sality: Story of a peer-to-peer viral network," Tech. Rep., 2011.

[32] J. Wyke, "The zeroaccess botnet: Mining and fraud for massive financial gain," *Sophos Technical Paper*, 2012.

[33] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "SoK: P2PWNED-modeling and evaluating the resilience of peer-to-peer botnets," in *2013 IEEE Symposium on Security and Privacy (SP).* IEEE, 2013, pp. 97–111.

[34] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. C. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm." *LEET*, vol. 8, no. 1, pp. 1–9, 2008.

[35] Damballa, "A new iteration of the TDSS / TDL4 malware using DGA-based command-and-control." Damballa, Tech. Rep., 2012.

[36] L. Zhang, S. Yu, D. Wu, and P. Watters, "A survey on latest botnet attack and defense," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on.* IEEE, 2011, pp. 53–60.

[37] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation."

[38] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[39] H. Choi, H. Lee, and H. Kim, "Botgad: detecting botnets by capturing group activities in network traffic," in *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE.* ACM, 2009, p. 2.

[40] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation." in *Proceedings of the 16th USENIX Security Symposium*, vol. 7, 2007, pp. 1–16.

[41] S. Haas, S. Karuppayah, S. Manickam, M. Mühlhäuser, and M. Fischer, "On the resilience of p2p-based botnet graphs," in *Communications and Network Security (CNS), 2016 IEEE Conference on*. IEEE, 2016, pp. 225–233.

[42] J. Yan, L. Ying, Y. Yang, P. Su, and D. Feng, "Long term tracking and characterization of p2p botnet," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*. IEEE, 2014, pp. 244–251.

[43] B. B. Kang, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon *et al.*, "Towards complete node enumeration in a peer-to-peer botnet," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ACM, 2009, pp. 23–34.

[44] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding p2p bots with structured graph analysis." in *Proceedings of the 19th USENIX Security Symposium*, 2010, pp. 95–110.

[45] D. Dagon, G. Gu, C. P. Lee, and W. Lee, "A taxonomy of botnet structures," in *Proceedings of 23rd Annual Computer Security Applications Conference (ACSAC '07)*, 2007, pp. 325–339.

[46] A. Sanatinia and G. Noubir, "Onionbots: Subverting privacy infrastructure for cyber attacks," *CoRR*, vol. abs/1501.03378, 2015.

[47] A. Ramachandran, S. Seetharaman, N. Feamster, and A. Lakhina, "Monitoring stealthy network conversations with sampled traffic," *Georgia Tech Technical Report*, 2006.

[48] T. Alpcan and T. Basar, "An intrusion detection game with limited observations," in *Proceedings of the 12th Int. Symp. on Dynamic Games and Applications*, 2006.

[49] K. Khalil, Z. Qian, P. Yu, S. Krishnamurthy, and A. Swami, "Optimal monitor placement for detection of persistent threats," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[50] S. Schmidt, T. Alpcan, Ş. Albayrak, T. Başar, and A. Mueller, "A malware detector placement game for intrusion detection," in *International Workshop on Critical Information Infrastructures Security*. Springer, 2007, pp. 311–326.

[51] M. P. Wellman and A. Prakash, "Empirical game-theoretic analysis of an adaptive cyber-defense scenario (preliminary report)," in *International Conference on Decision and Game Theory for Security*. Springer, 2014, pp. 43–58.

[52] L. Garber, "Denial-of-service attacks rip the Internet," *Computer*, vol. 33, no. 4, pp. 12–17, April 2000.

[53] M. Geva, A. Herzberg, , and Y. Gev, "Bandwidth distributed denial of service: Attacks and defenses," *IEEE Security & Privacy*, vol. 12, no. 1, pp. 54–61, January/February 2014.

[54] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth Quarter 2013.

[55] A. Stavrou, *Encyclopedia of Cryptography and Security*, 2nd ed. Springer, 2011, ch. Overlay-Based DoS Defenses, pp. 891–897.

[56] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein, "MOVE: An end-to-end solution to network denial of service," in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2005)*, San Diego, California, USA, February 2005, pp. 81–96.

[57] Q. Duan, E. Al-Shaer, and H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *Proceedings of the IEEE Conference on Communications and Network Security (IEEE CNS 2013)*. Washington, DC, USA: IEEE, October 2013, pp. 260–268.

[58] P. Mittal, D. Kim, Y.-C. Hu, and M. Caesar, "Mirage: Towards deployable ddos defense for web applications," *arXiv preprint arXiv:1110.1060*, 2011.

[59] G. C. Moura, *Internet bad neighborhoods*. Giovane Cesar Moreira Moura, 2013, no. 12.

[60] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, "Using uncleanliness to predict future botnet addresses," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 93–104.

[61] Y. Zeng, X. Hu, and K. G. Shin, "Detection of botnets using combined host- and network-level information," in *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010, pp. 291–300.

[62] S. Venkatesan, M. Albanese, and S. Jajodia, "Disrupting stealthy botnets through strategic placement of detectors," in *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 2015, pp. 95–103.

[63] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[64] Y. Shinoda, K. Ikai, and M. Itoh, "Vulnerabilities of passive internet threat monitors." in *USENIX Security*, 2005.

[65] V. Shmatikov and M.-H. Wang, "Security against probe-response attacks in collaborative intrusion detection," in *Proceedings of the 2007 workshop on Large scale attack defense*. ACM, 2007, pp. 129–136.

[66] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2001, pp. 346–353.

[67] R. Chadha, T. Bowen, C.-Y. J. Chiang, Y. M. Gottlieb, A. Poylisher, A. Sapello, C. Serban, S. Sugrim, G. Walther, L. M. Marvel *et al.*, "CyberVAN: A cyber security virtual assured network testbed," in *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 2016, pp. 1125–1130.

[68] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia, "A moving target defense approach to disrupting stealthy botnets," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. ACM, 2016, pp. 37–46.

[69] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," in *ACM Sigmod Record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[70] Trustwave, "New POS malware emerges - punkey," https://www.trustwave.com/Resources/SpiderLabs-Blog/New-POS-Malware-Emerges—Punkey/, 2015, accessed: 2017-01-05.

[71] P. Stoica and R. L. Moses, *Introduction to spectral analysis*. Prentice hall Upper Saddle River, 1997, vol. 1.

[72] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[73] J. D. Scargle, "Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data," *The Astrophysical Journal*, vol. 263, pp. 835–853, 1982.

[74] F. Frescura, C. Engelbrecht, and B. Frank, "Significance tests for periodogram peaks," *arXiv preprint arXiv:0706.2225*, 2007.

[75] M. Vlachos and V. Castelli, "On periodicity detection and structural periodic similarity." vol. 5, 2005, pp. 449–460.

[76] A. J. Aviv and A. Haeberlen, "Challenges in experimenting with botnet detection systems." in *Proceedings of the 4th Workshop on Cyber Security Experimentation and Test (CSET '11)*, 2011.

[77] S. Garcia and V. Uhlir, "Malware capture facility project," http://mcfp.weebly.com/mcfp-dataset.html, 2013, accessed: 2017-01-30.

[78] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of dga-based malware." in *USENIX security symposium*, vol. 12, 2012.

[79] L. Spitzner, *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003, vol. 1.

[80] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. IEEE, 2009, pp. 299–304.

[81] Y. Wang, S. Wen, Y. Xiang, and W. Zhou, "Modeling the propagation of worms in networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 942–960, 2014.

[82] T. Micro, "How do threat actors move deeper into your network?" 2013.

[83] P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, and A. Sung, "Network based detection of virtual environments and low interaction honeypots," in *Proceedings of the 2006 IEEE SMC, Workshop on Information Assurance*, 2006, pp. 283–289.

[84] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–9.

[85] M. West and J. Vacca, "Preventing system intrusions," *Computer and information security handbook*, pp. 39–51, 2009.

[86] R. Ganesan, S. Jajodia, A. Shah, and H. Cam, "Dynamic scheduling of cybersecurity analysts for minimizing risk using reinforcement learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 4, 2016.

[87] J. M. Nascimento and W. B. Powell, "An optimal approximate dynamic programming algorithm for the lagged asset acquisition problem," *Mathematics of Operations Research*, vol. 34, no. 1, pp. 210–237, 2009.

[88] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[89] R. Bellman, "Dynamic programming," *Princeton, USA: Princeton University Press*, vol. 1, no. 2, p. 3, 1957.

[90] A. Gosavi, "Simulation-based optimization," *parametric optimization techniques and reinforcement learning*, 2003.

[91] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[92] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, Sep. 2009, pp. 99–100.

[93] A. Wang, A. Mohaisen, W. Chang, and S. Chen, "Delving into internet DDoS attacks by botnets: characterization and analysis," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2015)*. IEEE, 2015, pp. 379–390.

[94] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 2013.

[95] P. R. Freeman, "Sequential estimation of the size of a population," *Biometrika*, vol. 59, no. 1, pp. 9–17, 1972.

[96] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks." in *NDSS*, 2008.

[97] R. Ortalo, Y. Deswarte, and M. Kaâniche, "Experimenting with quantitative evaluation tools for monitoring operational security," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 633–650, 1999.

[98] S. M. Mc Carthy, A. Sinha, M. Tambe, and P. Manadhata, "Data exfiltration detection and prevention: Virtually distributed pomdps for practically safer networks," in *International Conference on Decision and Game Theory for Security.* Springer, 2016, pp. 39–61.

[99] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[100] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence.* ACM, 2011, pp. 43–58.

[101] W. Connell, M. Albanese, and S. Venkatesan, "A Framework for Moving Target Defense Quantification," in *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference, SEC 2017, Rome, Italy, May 29-31, 2017, Proceedings.* Springer, 2017, pp. 124–137.

[102] L. Wang, S. Noel, and S. Jajodia, "Minimum-cost network hardening using attack graphs," *Computer Communications*, vol. 29, no. 18, pp. 3812–3824, 2006.

[103] M. Albanese, S. Jajodia, and S. Noel, "Time-efficient and cost-effective network hardening using attack graphs," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on.* IEEE, 2012, pp. 1–12.

Biography

Sridhar Venkatesan graduated from Sri Sankara Senior Secondary School, Chennai, India in 2007. He received his Master of Science in Theoretical Computer Science from PSG College of Technology, Coimbatore, India in 2012.