Detecting Stealthy Malware Using Behavioral Features in Network Traffic

Ting-Fang Yen

August 2011

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

It is clearly in the interest of network administrators to detect hosts within their networks that are infiltrated by stealthy malware. Infected hosts (also called *bots*) can exfiltrate sensitive data to adversaries, or lie in wait for commands from a bot-master to forward spam, launch denial-of-service attacks, or host phishing sites, for example. Unfortunately, it is difficult to detect such hosts, since their activities are subtle and do not disrupt the network.

In this thesis, we hypothesize that malware-infected hosts share characteristics in their network behaviors, which are distinct from those of benign hosts. Our approach works by aggregating "similar" network traffic involving multiple hosts. We identify key characteristics that capture basic properties of botnet operation, and that can be observed even within coarse network traffic summaries, i.e., flow records. Using network traffic collected at the edge routers of the Carnegie Mellon University campus network, and network traffic generated from real bot instances in virtual machines and honeynets running in the wild, we demonstrate that this approach can reliably detect infected hosts with very few false positives.

In addition to identifying relevant behavioral features within hosts' network activities, another contribution of this thesis is in developing efficient algorithms for analyzing network traffic. Our algorithms utilize methods from diverse areas, including statistics, data mining, machine learning, and metric embeddings. We also introduce a technique to passively infer the application implementation on a host given only anonymized traffic summaries. This technique enables us to detect malware that is browser-dependent, and can also be applied to improve the accuracy of traffic deanonymization, i.e., identifying the web sites in anonymized flow records.

To complement empirical analyses, we apply analytical models from network theory to study peer-to-peer botnets. We focus on a structural property of networks, which characterizes the tendency for edges to exist between "similar" nodes, and examine its effect on network resiliency and the network's ability to recover after a fraction of the nodes are removed. We show that previous works may have over-estimated the power of certain botnet takedown strategies, and identify an alternative strategy that is more effective than those explored previously.

Acknowledgments

I would like to thank my advisor, Mike Reiter, for his continuous support and guidance. I owe much to his patience, his optimism, his attention to details, and his honest advices. It was his many encouragements that kept me going when I had doubts about myself. Of course, the North Carolina sunshine also helped.

I am grateful to my thesis committee, Lujo Bauer, Wenke Lee, and John McHugh, for their insightful discussions and comments. Thanks to Fabian Monrose for valuable experiences in data analysis and for his cheerful encouragements, to Chenxi Wang for advising me during my first two years in grad school as a Master's student, and to Yinglian Xie, Fang Yu, Martín Abadi for their mentoring during my internship at Microsoft Research and beyond.

My thanks go to those who helped provide the data that made this work possible. Moheeb Rajab and other members of the Johns Hopkins Honeynet Project, for malware binaries. Wenke Lee, Guofei Gu, David Dagon, and Yan Chen, for botnet traces. Chas DiFatta, Mark Poepping, Jim Gargani, and other members of the EDDY Initiative, for network traffic records from Carnegie Mellon University.

Thanks to my friends for making my grad-student days happy and memorable. To my parents, my family, and *benim dostum*, thank you so much for your love and support.

This research was funded by NSF awards 0326472, 0433540, 0756998 and 0831245.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Stealthy malware is largely responsible for attacks on the Internet infrastructure, for the proliferation of unsolicited spam e-mails, and for exfiltrating private information from end users and organizations. Infected hosts (also called *bots*) are also used to host, or serve as proxies for, phishing sites, spammer sites, and those with other malicious content. These activities are performed by infected hosts at the command of the attacker, i.e., the bot-master, where communications between bots and the bot-master often take place over existing network protocols, including Internet Relay Chat (IRC), HTTP, and peer-to-peer (P2P) networks.

Unfortunately, it is difficult to detect hosts infected with such malware, since by default they do little to arouse suspicion: e.g., generally their communications neither consume significant bandwidth nor involve a large number of targets. Analysis can be further complicated by infected hosts encrypting their network traffic, or communicating over peer-to-peer protocols to "blend-in" with peer-to-peer file-sharing traffic. While bots enlisted in aggressive scanning for other vulnerable hosts can be detected using known techniques, it would be better to detect the infected hosts prior to them engaging in malicious activities. For these reasons, bot detection has remained a challenging problem.

In this thesis, we hypothesize that hosts infected with stealthy malware share behavioral characteristics in their network activities that are distinct from those of benign hosts. We propose methods that detect infected hosts by simply identifying aggregates of "similar" communications, using only information contained in coarse network traffic summaries. In this chapter, we first review existing approaches for bot detection, and then present our approach and contributions.

1.1 Previous Bot Detection Techniques

Generally speaking, communications between bots and the bot-master can take place either through a centralized channel, e.g., Internet Relay Chat (IRC) or HTTP, or in a decentralized manner over peer-to-peer (P2P) protocols. In a centralized topology, every bot reports directly to the bot-master, whereas in a decentralized topology messages are relayed through other peers in the botnet. Examples of botnets that utilize the former include GTBot, AgoBot, SDBot, SpyBot [7], Zeus [15], Rustock [23], Clickbot [38], MegaD [24], Torpig [137], while Sinit, Phatbot [135], Storm [52, 66, 116, 138], Nugache [138], Waledac [17, 132], Conficker [117] fall into the latter category. Some botnets also have a hierarchical organization that includes both centralized and decentralized topologies at distinct layers (e.g., Storm, Waledac). The Stuxnet worm [45], which specifically targets machines in industrial systems that may not be connected to the Internet, communicates with centralized HTTP command-and-control servers (if the infected host is online), but also utilizes a P2P component to allow updates to be propagated to other bots in the local network.

Due to their unusual command-driven and coordinated nature, bot detection techniques largely work by examining network traffic for anomalies. Previous work in this area can be grouped into the following categories:

Signature-based Techniques. This group of proposals often rely on heuristics that assume certain models of botnet architecture or behavior. Such assumptions include IRC-based command-and-control [14, 29, 51, 93, 119], or the presence of scanning activities or denial-of-service attacks performed by the infected hosts [80]. These approaches, not unlike signature-based intrusion detection, can be evaded by malware that do not conform to their profile.

Other proposals generate traffic signatures that can be used in network intrusion detection systems (e.g., Bro [113]). For example, by examining bot binaries running in controlled environments, Perdisci et al. [115] clustered suspicious HTTP traffic that share similar statistical and structural properties. Wurzinger et al. [151] identified changes in the network behaviors of a bot, which are assumed to indicate that it received commands from the bot-master. Common substrings in the traffic immediately preceding behavior changes are used to construct signatures. These approaches are hindered by bots that encrypt their communications, and require access to the malware binary.

Behavior-based Techniques. In contrast to signature-based approaches, which are limited to specific activities performed by the infected hosts, several works proposed to detect bots by examining behavioral characteristics of their network traffic. These works focus on the spatial-temporal similarities between the behaviors of hosts participating in the same botnet, e.g., communication with the bot-master's command-and-control servers and performing the same malicious activities. For instance, BotHunter [53] detects compromised hosts by identifying events that take place when a vulnerable host is infected, and which show evidence of coordinated activities between the infected host and the bot-master. BotSniffer [55] identifies hosts with similar suspicious network activities, namely scanning and sending spam emails, and who also share common communication contents, defined by the number of shared bi-grams. BotMiner [54] groups together hosts based on destination or connection statistics (i.e., the byte count, the packet count, the number of flows, etc.), and on their suspected malicious activities (i.e., scanning, spamming, downloading binaries, or sending exploits). However, these approaches still rely on certain models of bot behavior, using signatures to detect the occurrence of infections and suspicious activities (and hence may require access to full packets that remain unencrypted).

Lu et al. [94, 95] assume bot activities to be more synchronized than human activities, and detect infected hosts by looking for hosts with similar byte frequency distributions in their payload within the same time window, e.g., one second. This approach can thus be evaded with encryption, and may not be applicable to delay-tolerant botnets [21]. Giroire et al. [50] proposed methods to detect centralized botnet command-and-control traffic by monitoring persistent and regular connections made to the same group of destination IP addresses, i.e., the botnet control servers. Since legitimate user traffic can also appear to be persistent and regular, this approach requires whitelisting common sites that benign users visit. It can also be evaded by malware that communicate in a decentralized manner over peer-to-peer protocols. BotProbe [56] exploits the determinism in the bots' pre-programmed behavior in response to commands from the bot-master. It actively replays, modifies, and intercepts packets of suspicious connections, and utilizes hypothesis testing to distinguish bot traffic from those of human users. However, this approach is hindered by encryption, and may require human participation.

Detecting Specific Botnet Operations. Another line of work includes techniques for identifying behaviors involving certain botnet operations. Ramachandran et al. [123] observed that bot-masters lookup DNS blacklists to tell whether their bots are blacklisted. They thus monitor lookups to a DNS-based blacklist to identify bots. Fast-flux is a technique used by botnets to hide the backend control server, or to host spam campaigns and phishing websites [67]. It operates by using dynamic DNS to establish a proxy network based on the infected hosts, such that a single domain is associated with many different IP addresses. Methods for identifying fast-flux domains include observing the number of DNS A records mapping to a hostname, and the geographic diversity in the IPs associated with a domain [64, 65, 109]. Hu et al. [68] also proposed detecting hosts participating in fast-flux networks by identifying HTTP redirection activity.

Infiltration and Disruption. Since some peer-to-peer botnets are built on existing network protocols, e.g., the Overnet protocol based on the Kademlia distributed hash table [97], researchers have infiltrated botnet networks in attempts to enumerate the infected hosts [77] or to disrupt botnet operation. As an example of the latter, researchers have injected a large number of fake nodes into the network to perform Sybil attacks [40, 60, 66], including content poisoning or eclipsing certain nodes from the rest of the P2P network. These studies showed that the effectiveness of the attack depends on the attack duration as well as the number of Sybils.

1.2 Thesis Approach and Contributions

Several drawbacks are present in previous works described in Section 1.1. First, there are common assumptions about specific models of behavior performed by the infected hosts (e.g., the protocol of communication, the presence of scanning or spamming activities), or reliance on signatures appearing in their communication content. These assumptions can be evaded by new malware that encrypt their messages or that move to alternative communication architectures (e.g., private peer-to-peer protocols designed by the bot-masters). Second, many proposals require information contained in network packets, which may not be feasible to collect in large networks due to bandwidth and storage requirements. Rather than packets, coarse summaries of each network connection are often stored instead, e.g., Cisco NetFlow [1] or Argus flows [1]. These drawbacks highlight the tension in the arms race between bot-masters and network defenders: while malware is becoming increasingly sophisticated, we have less information on which to base our investigations.

In this thesis, we take on the perspective of a network administrator that has access to traffic crossing the border of her network, and that aims to identify internal hosts that are infected. We hypothesize that infected hosts share certain characteristics in their network behaviors that are distinct from those of benign hosts. Since malware rarely infects a single victim in a large enterprise network, such characteristic behav-

[1] `http://www.qosient.com/argus`

iors should appear roughly coincidentally at multiple machines in the network. Based on this hypothesis, our approach simply identifies aggregates of "similar" network traffic that are associated with multiple hosts. In contrast to previous behavior-based malware detection proposals that focus (in part) on similar malicious activities performed by multiple hosts [54, 55], we aim to exploit characteristic behaviors that are fundamental to the method by which infected hosts are engaged with the bot-master.

We first demonstrate the effectiveness of this approach in detecting infected hosts participating in centralized botnets. In particular, to avoid high bandwidth and storage requirements in collecting full packet data, our technique only utilizes information contained in network flow records, i.e., Argus records, which include the source and destination addresses and ports, flow start and end times, packet and byte counts, and the first 64 bytes of the flow payload. We further address the detection of increasingly sophisticated malware variants, specifically those that may not be confined to infecting hosts of a particular operating system, and those that communicate via resilient peer-to-peer network topologies. These later detection techniques explicitly avoid payload inspection altogether (even the partial 64-byte payload) to accommodate malware that encrypt their communications.

To complement our empirical analyses, we apply models from network theory to peer-to-peer botnets, and investigate the effectiveness of various botnet takedown strategies. Network models allows the "usefulness" of the botnet, as well as the impact of takedown attempts, to be evaluated using graph properties, e.g., the size of the largest connected component, the average distance between pairs of nodes. We focus on a particular structural characteristic of networks — assortativity — that affects the resilience of a network to takedown attempts and also its ability to recover after a fraction of nodes are removed.

The rest of this section presents an overview of our approach and contributions.

1.2.1 Aggregating Network Traffic for Malware Detection

To detect infected hosts participating in centralized botnets, we propose a system, called TĀMD (an abbreviation for "Traffic Aggregation for Malware Detection") [155], that distills traffic aggregates from the traffic passing the edge of the network. Each traffic aggregate is defined by certain characteristics that the traffic grouped within it shares in common. By refining the aggregates to include only traffic that shares multiple relevant characteristics, TĀMD constructs a small set of new aggregates (i.e., without previous precedent) that it recommends for examination, for example, by more targeted (e.g., signature-based) intrusion detection tools.

Specifically, the characteristics on which TĀMD aggregates traffic include:

- **Common destinations**: T$\bar{\text{A}}$MD analyzes the external networks with which internal hosts communicate, and identifies traffic to busier-than-normal external destinations. Previous studies have shown that the destination addresses with which a group of hosts communicates exhibit stability over time [2, 84]. Since common sites are generally better administered and have a small chance of being compromised by the attacker, malware activities are hence likely to exhibit communication patterns outside the norm, i.e., contacting destinations that the internal hosts would not have contacted otherwise.

- **Similar payload**: T$\bar{\text{A}}$MD identifies traffic with similar payloads or, more specifically, payload prefixes (i.e., the first 64 bytes of each flow) for which a type of edit distance (string edit distance with moves [30]) is small. Intuitively, command-and-control traffic between a bot-master and the bots should share significant structure, i.e., pertaining to the protocol syntax, and would have a low edit distance between them.

- **Common software platforms**: T$\bar{\text{A}}$MD passively fingerprints the platforms of internal hosts, and identifies traffic associated with hosts with common operating systems. Most malware tend to be platform-dependent, e.g., infecting hosts running a particular operating system.

Alone, each of these characteristics for forming traffic aggregates would be too coarse for identifying malware, as legitimate traffic can exhibit these characteristics as well. In combination, however, they can be quite powerful at extracting malware communications (and relatively few others). In Chapter 2, we show that with traffic generated from real bot instances, we were able to reliably extract this traffic from all traffic passing the edge of the Carnegie Mellon University campus network, while the number of other aggregates reported is very low. This is achieved even when the network traffic is recorded in the form of flow records, and when the number of infected hosts comprise only 0.0097% of all internal hosts in the network.

In addition to identifying traffic aggregates and ways of combining them to find malware-infected hosts, our contributions also include efficient algorithms for analyzing network traffic, which are drawn from diverse areas including signal processing, data mining, and metric embeddings. We also detail each of these algorithms in Chapter 2.

There are several approaches by which malware writers can attempt to avoid detection by T$\bar{\text{A}}$MD. These approaches include 1) encrypting their traffic, so that our payload comparisons will be ineffective, 2) exploiting cross-platform applications (e.g., web browsers), so that the malware is no longer operating system-dependent, and 3) switching to alternative botnet architectures, such as those built over P2P networks, where infected hosts do not need to contact a single centralized server. To accommodate the use of encryption, our techniques can be generalized to define encrypted content as "similar", though a better method perhaps is to move away from payload inspection altogether (even the partial 64-byte payload), since this information is not typically included in flow records. We further address the other two methods of evading T$\bar{\text{A}}$MD by developing techniques to perform passive application fingerprinting (targeting browser-dependent malware in particular) and techniques to detect P2P bots, described in the following subsections.

1.2.2 Passive Browser Fingerprinting from Coarse Traffic Summaries

A limitation to the T$\bar{\text{A}}$MD system, described above, is that it only forms platform aggregates based on the hosts' operating system. As such, it would not be able to detect malware that is application-dependent, but otherwise operating system-independent. Such malware (e.g., the `Trojan.PWS.ChromeInject` trojan [2] that exploits Mozilla Firefox, the `MSIL.Yakizake` worm [3] that exploits Mozilla Thunderbird, the `Imspam` trojan [4] that sends spam through MSN or AOL Messenger) could span multiple traffic aggregates formed by operating system fingerprinting alone, or represent only a small subset of an operating system aggregate. In either case, the mismatch between the software fingerprinted (the O/S) and the software exploited (e.g., the web browser) can allow the infected hosts to go unnoticed.

In addressing application-dependent malware, we focus particularly on those that are browser-dependent, since web browsers are arguably one of the most important and widespread applications in use today. Rather than examining the traffic payload (e.g., client data in HTTP headers), we present a new technique to infer the web browser implementation on a host using only information contained in flow records [157]. Our observation is that differences in browser implementations will result in varying network behaviors, even when the same web page is retrieved. For example, browsers may retrieve objects on a given page in different orders, there can be different numbers of objects in one connection, and the number of simultaneously active connections also varies.

However, aside from the content and structure of the websites, users' browsing behavior, browser configuration, geographic location, and the client hardware configuration can also affect browser network behavior. To overcome these challenges, we collected network traffic to top web sites using different

[2] `http://www.bitdefender.com/VIRUS-1000451-en--Trojan.PWS.ChromeInject.B.html`
[3] `http://www.symantec.com/security_response/writeup.jsp?docid=2007-092623-0634-99`
[4] `http://www.symantec.com/security_response/writeup.jsp?docid=2007-041200-4330-99`

browser implementations from hosts in PlanetLab [25], which is a geographically distributed network of machines. This allows us to construct a browser classifier based on Support Vector Machines (SVM) [31]. Given the limited amount of information contained in flow records, we demonstrate that our browser classifier performs well on real user traffic recorded at the border of the Carnegie Mellon University campus network — even when the training and testing datasets are from different time frames, to different websites, and collected at different geographic locations.

In Section 3, we show that incorporating browser fingerprinting into the platform aggregation step in T$\bar{\text{A}}$MD induces small overhead, while allowing the detection of a wider range of malware (i.e., both those that are operating system-dependent and browser-dependent). Moreover, in a second application of browser fingerprinting, we demonstrate that knowledge of the client browser can be used to achieve a higher accuracy in the deanonymization of web *sites* than has previously been achievable from flow records. In our experiments on traffic from our campus network, the precision of website identification can be improved by 17% on average from the case without knowledge of the browser implementation.

1.2.3 Telling Peer-to-Peer File-sharing and Bots Apart

In contrast to centralized botnets, peer-to-peer (P2P) bots do not communicate with a single server, but rather relay messages between other peers in the network. The motivations for bots using P2P substrates are similar to those underlying the use of P2P protocols for file-sharing; the takedown of Napster, for example, highlighted the limitations of a centralized "command-and-control" infrastructure in that domain. It is thus not surprising that P2P substrates now support both file-sharing and botnet activities. Since no single control server exists for P2P botnets, it also becomes possible for infected hosts to evade the destination aggregation step in T$\bar{\text{A}}$MD.

A consequence of this common use of P2P technologies is that botnet command-and-control traffic will tend to "blend into" a background of P2P file-sharing, making it difficult to separate these two types of traffic. In both cases, status information about available peers needs to be maintained constantly to ensure the connectivity of the network; peers experience a high connection failure rate due to the dynamics of nodes joining and leaving (i.e., "churn"); and peers participate in both client and server activities. This commonality is punctuated by the fact that one highly publicized and well-studied P2P botnet, Storm, built its communication protocol based on the Overnet network, whose distributed hash table implementation [97] is incorporated in both eDonkey [5] and BitTorrent [6] file-sharing applications.

[5] `http://wiki.amule.org/index.php/FAQ_eD2k-Kademlia`
[6] `http://bittorrent.org/beps/bep_0005.html`

In light of this, the primary problem facing the detection of such bots is differentiating them from other P2P hosts. We focus specifically on the problem of P2P bot detection given this challenge [156]. By targeting basic properties of bots that operate over P2P networks, we construct a series of tests on network traffic to separate them from P2P file-sharing hosts. The characteristics on which our tests are based include:

- **Volume**: Since file-sharing hosts generally perform large multi-media file transfers (e.g., MP3, movies), but bots almost never do, traffic volume can be an rough indicator of suspicious activity.

- **Peer churn**: The peer membership of a file-sharing network is very dynamic, due to peers constantly joining and leaving the network. Studies [58, 126, 140] have shown that most file-sharing hosts appear only once a day, and remain connected for short durations. Bots, by contrast, are likely to experience less churn in their peer membership, since they are required to maintain connectivity to other peers to receive and execute commands from the bot-master. A bot also cannot control when network access will be available, and so it is often opportunistic in communicating with peers, i.e., whenever it has a chance. In addition, each bot maintains a list of known peers with which to communicate, such that it tends to contact the same hosts repeatedly.

- **Human-driven versus Machine-driven**: A more fundamental difference between bots and file-sharing hosts is that, while file-sharing activities are mainly human-driven, bots are almost entirely automated. This causes much of their traffic to exhibit temporal similarity that is rarely seen among those from human activities.

We use these characteristics in combination to build a technique for separating P2P bots from P2P file-sharing hosts. Evaluated on network traffic observed at the border of the Carnegie Mellon University campus network, our technique can detect Storm bots with up to 87.50% true positive rate and only 0.47% false positives, for example.

Another contribution of our work is in quantifying the degree to which malware would need to alter their behaviors to evade detection. We show that evading our techniques would require significant behavioral changes to existing botnets, and, due to the way in which our tests are constructed, it would typically not be evident to the bots how much change would be sufficient for evasion. Details of our technique and evaluations are presented in Chapter 4.

1.2.4 Revisiting Analytical Botnet Models

An analytical approach to studying botnets, adopted by several previous works [36, 41, 90, 91], is to apply graph models from network theory to P2P botnets. Each node in the network represents an infected host, and edges reflect communications between the hosts. Network models allow the "usefulness" of botnets to be quantified using properties of the graph, such as the diameter, the size of the largest connected component, and the average shortest distance between nodes. More importantly, the models can be used to assess the effectiveness of strategies aimed at taking down a botnet. For example, one strategy that was found to be effective for some network topologies is to target nodes with high degree, i.e., that communicate with many hosts [36, 41, 158].

We observe that previous works applying graph models to P2P botnets do not consider an important property of networks — assortative mixing [102]. Assortativity refers to the tendency for a node to attach to other "similar" nodes, and is commonly examined in terms of a node's degree, i.e., high-degree nodes are likely to be neighbors of other high-degree nodes. This property is also called the degree correlation. The existence of this correlation between neighboring nodes has been observed in many real-world networks [102, 104, 111]. It has also been found to be a property of growing networks [19, 86], where the network increases in size as nodes join over time, as is true in a botnet as more hosts become infected.

In Chapter 5, we show that assortativity plays an important role in network structure, such that neglecting it can lead to an over-estimation of the effectiveness of botnet takedown strategies. By generating networks with varying levels of degree correlation, we demonstrate that a higher level of assortativity allows the network to be more resilient to certain takedown strategies, including those found to be effective by previous works. Moreover, since bots are dynamic entities that can react and adapt to changes in the network, the botnet can potentially "heal" itself after a fraction of its nodes are removed. We specifically explore cases where nodes can compensate for lost neighbors by creating edges to other nearby nodes, e.g., that are within h hops. Our simulations show that the graph can recover significantly after takedown attempts, even when h is small, and that higher levels of assortativity can allow the network to recover more effectively.

We also identify alternative takedown strategies that are more effective than those explored in previous works. By targeting nodes with both high degree and low clustering coefficient, the connectivity and communication efficiency of the network will decrease significantly, such that it is considerably more difficult for the network to recover from the takedown attempt.

1.3 Outline

This thesis is organized as follows:

- Chapter 2 describes the T$\bar{\text{A}}$MD system, including algorithms for efficiently aggregating "similar" network traffic, and evaluations performed on network traffic collected at the border of our university campus network and those generated by real bot instances.

- Chapter 3 describes an approach to passively determine a remote host's web browser implementation using only information contained in network flow records, and its application to improving the platform aggregation in T$\bar{\text{A}}$MD.

- Chapter 4 presents techniques for distinguishing peer-to-peer bots from file-sharing hosts. Evaluations are performed on network traffic collected at the border of our university campus network, and also traffic from P2P bots in honeynets running in the wild. We also quantify operational costs required for bots to evade detection.

- Chapter 5 studies analytical network models and their application to peer-to-peer botnets. We explore in detail the effect that assortativity has on the resiliency of the network and on its healing ability.

- Chapter 6 summarizes key contributions of this thesis, its limitations, and potential future directions.

Chapter 2

Traffic Aggregation for Malware Detection

It is clearly in the interest of network administrators to detect computers within their networks that are infiltrated by malware. Infected hosts, i.e., bots, can exfiltrate sensitive data to adversaries, or lie in wait for commands from a bot-master to forward spam or launch denial-of-service attacks, for example. Unfortunately, it is hard to detect such hosts, since by default they do little to arouse suspicion: e.g., generally their communications neither consume significant bandwidth nor involve a large number of targets. While this changes if the bots are enlisted in aggressive scanning for other vulnerable hosts or in denial-of-service attacks — in which case they can easily be detected using known techniques (e.g., [101, 133]) — it would be better to detect the bots prior to such a disruptive event, in the hopes of averting it.

We hypothesize that even stealthy, previously unseen malware is likely to exhibit communication that is detectable, if viewed in the right light. First, since emerging malware rarely infects only a single victim, we expect its characteristic communications, however subtle, to appear roughly coincidentally at multiple hosts in a large network. Second, we expect these communications to share certain features that differentiate them from other communications typical of that network. Of course, these two observations may pertain equally well to a variety of communications that are not induced by malware, and consequently the challenge is to refine these observations so as to be useful for detecting malware in an operational system.

In this chapter, we describe such a system, called TĀMD, an abbreviation for "Traffic Aggregation for Malware Detection". As its name suggests, TĀMD distills *traffic aggregates* from the traffic passing the edge of a network, where each aggregate is defined by certain characteristics that the traffic grouped within it shares in common. By refining these aggregates to include only traffic that shares multiple relevant characteristics, and by using past traffic as precedent to justify discarding certain aggregates as normal, TĀMD constructs a small set of new aggregates (i.e., without previous precedent) that it recommends for examination, for example, by more targeted (e.g., signature-based) intrusion detection tools. The key to maximizing the data-reducing precision of TĀMD is the characteristics on which it aggregates traffic, which include:

- **Common destinations**: TĀMD analyzes the networks with which internal hosts communicate, in order to identify aggregates of communication to busier-than-normal external destinations. Spyware reporting to the attacker's site or bots communicating with a bot-master (e.g., via IRC, HTTP, or another protocol) might thus form an aggregate under this classification.

- **Similar payload**: TĀMD identifies traffic with similar payloads or, more specifically, payloads for which a type of edit distance (*string edit distance matching with moves* [30]) is small. Intuitively, command-and-control traffic between a bot-master and his bots should share significant structure and hence, we expect, would have a low edit distance between them.

- **Common internal-host platforms**: TĀMD passively fingerprints platforms of internal hosts, and forms aggregates of traffic involving internal hosts that share a common platform. Traffic caused by malware infections that are platform-dependent should form an aggregate by use of this characteristic.

Alone, each of these methods of forming traffic aggregates would be far too coarse to be an effective data-reduction technique for identifying malware, as legitimate traffic can form aggregates under these characterizations as well. In combination, however, they can be quite powerful at extracting aggregates of malware communications (and relatively few others). To demonstrate this, we detail a particular configuration of TĀMD that employs these aggregation techniques to identify internal hosts infected by malware that reports to a controller site external to the network. Indeed, botnets have been observed to switch controllers or download updates frequently, as often as every two or three days [51, 80]; each such event gives TĀMD an opportunity to identify these communications. We show that with traffic generated from real spyware and bot instances, TĀMD was able to reliably extract this traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low.

In addition to identifying aggregates and ways of combining them to find malware-infected hosts, the contributions of TĀMD include algorithms for computing these aggregates efficiently. Our algorithms draw from diverse areas including signal processing, data mining and metric embeddings. We also detail each of these algorithms in this chapter.

2.1 Related Work

Bot Detection. Previous works on bot detection are described in Chapter 1.1, where they are roughly categorized into signature-based techniques, behavior-based techniques, or those that focus on identifying specific operations of botnets. We believe our approach to be fundamentally different from previous works in the following respect. While previous approaches largely work from known models of malware operation, we focus on behavioral characteristics that are fundamental to the method by which infected hosts and the bot-master are engaged, and simply seek to identify new aggregates of communication that are not explained by past behavior on the network being monitored. Like all anomaly-detection approaches, our challenge is to demonstrate that the number of identified anomalous aggregates is manageable, but it has the potential to identify a wider range of as-yet-unseen malware. In particular, the assumptions underlying previous systems present opportunities for attackers to evade these systems by changing the activities performed by the bots.

More related to TĀMD are behavior-based techniques that incorporate aspects of using aggregation for detecting bots. For example, BotSniffer [55] looks for infected hosts displaying spatial-temporal similarity. It identifies hosts with similar suspicious network activities, namely scanning and sending spam emails, and who also share common communication contents, defined by the number of shared bi-grams. BotMiner [54] groups together hosts based on destination or connection statistics (i.e., the byte count, the packet count, the number of flows, etc.), and on their suspected malicious activities (i.e., scanning, spamming, downloading binaries, or sending exploits). BotMiner is more similar to TĀMD in the sense that they both identify hosts sharing multiple common characteristics, but the characteristics on which TĀMD and BotMiner cluster hosts are different. BotSniffer seeks to identify known bot activities, such as scanning or spamming, and limits its attention only to bots using IRC or HTTP to communicate with a centralized bot-master.

Techniques. The techniques we employ for aggregation, specifically on the basis of external subnets to which communication occurs, include some drawn from the statistics domain. While others have drawn from this domain in the detection of network traffic anomalies, our approach has different goals and hence applies these techniques differently. Coarsely speaking, past approaches extract packet header information, such as the number of bytes or packets transferred for each flow, counts of TCP flags, in search of volume anomalies like denial-of-service attacks, flash crowds, or network outages [8, 83, 143]. Lakhina et al. [89] studied the structure of network flows by decomposing OD flows (flows originating and exiting from the same ingress and egress points in the network) using Principal Component Analysis (PCA). They expressed each OD flow as a linear combination of smaller "eigenflows", which may belong to deterministic periodic trends, short-lived bursts, or noise, in the traffic. Terrell et al. [144] grouped network traces into time-series data and selecting features of the traffic from each time bin, including the number of bytes, packets, flows, and the entropy of the packet size and port numbers. They applied Singular Value Decomposition (SVD) to the time-series data, and were able to detect denial-of-service attacks by examining the low-order components.

In general, transient and light-weight events would go unnoticed by these approaches, such as spammers that send only a few emails over the course of a few minutes [122]. Our work, on the other hand, is targeted at such lighter-weight events and so employs these techniques differently, not to mention techniques from other domains (e.g., metric embeddings, passive fingerprinting). Ramachandran et al. [121], in assuming that spammers exhibit similar email-sending behaviors across domains, constructed patterns corresponding to the amount of emails sent to each domain by known spammers. The patterns are calculated from the mean of the clusters generated through spectral clustering [22]. This is similar to our method of finding flows destined to the same external subnets; however, they do not look at other aspects of spamming besides the destination.

Another technique we employ is payload inspection, specifically to aggregate flows based on similar content. Payload inspection has been applied within methods for detecting worm outbreaks and generating signatures. Many previous approaches assume that malicious traffic is significantly more frequent or widespread than other traffic, and so the same content will be repeated in a large number of different packets or flows (e.g., [79, 82, 108, 114, 133]); we do not make this assumption here. Previous approaches to comparing payloads includes matching substrings [79, 105] or n-grams [55, 108, 146], hashing blocks of the payload [83, 133], or searching for the longest common substring [87]. Compared to these methods, our edit distance metric is more sensitive and accurate in cases where parts of the message are simply shifted or replaced. ARAKIS from CERT Polska [1] is an early-warning system that generates signatures for new threats. Assuming new attacks will have payloads not seen previously, they examine traffic from honeypots and darknets to cluster flows with similar content (determined by comparing Rabin hashes) not seen before, and that are performing similar activities, i.e., port scanning. A signature is generated from the longest common substrings of the similar flows. However, ARAKIS currently only focuses on threats that propagate through port scanning.

[1] `http://www.arakis.pl`

Another tool for intrusion analysis is the commercial product StealthWatch from Lancope [2]. StealthWatch monitors all traffic at the network border, checking for policy violations or signs of anomalous behavior by looking for higher-than-usual traffic volumes. Although this is similar to our approach of using past traffic as a baseline for identifying busier-than-normal external destinations, they does not refine this information using, e.g., payload or platform aggregation as we do here. Thus, it is primarily useful for detecting large-volume anomalies like port scanning and denial-of-service attacks.

2.2 Defining Aggregates

Given a collection of bi-directional flow records observed at the edge of an enterprise network, our system aims to identify infected internal hosts by finding communication "aggregates", which consist of flows that share common network characteristics. Specifically, T$\bar{\text{A}}$MD deploys three aggregation functions to identify flows with the following characteristics: those that contribute to busier-than-usual destinations, that have payloads for which a type of edit distance is small, or that involve internal hosts of a common software platform.

The aggregation functions take as input collections of flow records, Λ, and output either groups (aggregates) of internal hosts that share particular properties or a value indicating the amount of similarity between the input flow record collections. We presume that each flow record $\lambda \in \Lambda$ includes the IP address of the internal host $\lambda.\mathsf{internal}$ involved in the communication and the external subnet $\lambda.\mathsf{external}$ with which it communicates. λ also includes some portion of the payload $\lambda.\mathsf{payload}$ of that communication, packet header fields, and the start and end time of the communication.

2.2.1 Destination Aggregates

Previous studies show that the destination IP addresses with which a group of hosts communicates exhibit stability over time, both in the amount of traffic sent and in the set-membership of the destinations [2, 84]. Malware activities are thus likely to exhibit communication patterns outside the norm, i.e., contacting destinations that the internal hosts would not have contacted otherwise.

The destination aggregation function $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$ takes as input two sets Λ, Λ_{past} of communication records. The variable τ is a parameter to the function, as described later in this section. By analyzing the external addresses with which internal hosts communicate in Λ and Λ_{past}, the function

[2] `http://www.lancope.com`

outputs a set $\mathsf{SuspiciousSubnets}$ of destination subnets for which there is a larger-than-usual number of interactions with the internal network, using Λ_{past} as a baseline. The function also outputs an integer $\mathsf{numAggs}$ and a set Agg_i ($1 \leq i \leq \mathsf{numAggs}$), where Agg_i are internal hosts (IP addresses) that originated traffic in Λ, and who contributed to larger-than-usual number of interactions with an external destination subnet in $\mathsf{SuspiciousSubnets}$.

At a high level, the set $\mathsf{SuspiciousSubnets}$ of selected "suspicious" external destinations is determined after filtering out periodic and regular activities in the communications of the network as represented in the past traffic Λ_{past}. External destinations observed in Λ that do not follow the norm, i.e., that according to Λ_{past} are busier than usual or have not been contacted before, are thus output in $\mathsf{SuspiciousSubnets}$.

Below we describe the three processing steps in $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$: (i) Trend filtering, which selects the set of suspicious external destinations; (ii) Dimension reduction, which first characterizes each host by a vector indicating which suspicious destinations it interacted with, and then reduces the dimensionality of these vectors while preserving most of the information; and (iii) Clustering, which forms clusters of the vectors (i.e., internal hosts) by the destinations they contacted.

Trend Filtering. Trend filtering aims to remove regular and periodic communications from Λ, so that external destinations showing behavior outside the norm are identified. In particular, the "norm" is defined, for each external destination subnet, by the average number of internal hosts that communicated with that subnet in various periodic intervals, as recorded in Λ_{past}. For example, periodic patterns, such as Windows machines connecting to the Windows update server on a weekly basis or banking websites experiencing traffic spikes on pay day each month, can be inferred from Λ_{past}. The change in activity of a destination in Λ can then be measured by how much more traffic it received in Λ compared to its average values for previous time intervals in Λ_{past}. In the current implementation, a destination is selected to be in $\mathsf{SuspiciousSubnets}$ if no internal host has been seen to communicate with it for all previous periodic time intervals in Λ_{past}.

Dimension Reduction. Given $\mathsf{SuspiciousSubnets}$, each internal host can be represented as a binary vector $v = (v[1], v[2], \cdots, v[k])$ for which the dimensionality k is equal to the number of destinations in $\mathsf{SuspiciousSubnets}$. A dimension $v[i]$ is set to 1 if the internal host communicated with destination i in $\mathsf{SuspiciousSubnets}$ (according to Λ), and 0 otherwise. However, the dimensions may be redundant or dependent on one another; e.g., retrieving a web page can cause other web servers to be contacted. To identify such relationships between the destinations and reduce the vectors' dimensionality, we apply

Principal Component Analysis (PCA).

PCA [74] is a method for analyzing multivariate data. It enables data reduction by transforming the original vectors onto a new set of orthogonal axes, i.e., principal components, while preserving most of the original information. This is done by having each principal component capture as much of the variance in the data as possible.

While a vector originally has length equal to the number of suspicious destinations in $\mathsf{SuspiciousSubnets}$, the transformed vector after PCA has a dimensionality that is the number of selected principal components, with each dimension now representing a linear combination of the external destinations. The number of selected principal components depends on the amount of variance we want to capture in the data, denoted as the parameter τ. The more variance to be captured, the more accurate the transformation represents the original data, but, at the same time, more principal components are needed, increasing the dimensionality.

Clustering. PCA reduces the vector dimensionality significantly, after which hosts connecting to the same combinations of destinations can be identified efficiently through clustering. We form clusters of the vectors (i.e., internal hosts) using a modified version of the K-means clustering algorithm [81], which does not require the number of clusters to be known in advance.

1. Randomly select a vector as the first cluster hub. Assign all vectors to this cluster.

2. Select the vector furthest away from its hub as a new cluster hub. Re-assign all vectors to the cluster whose hub it is closest to.

3. Repeat step 2 until no vector is further from its hub than half of the average hub-hub distance.

Cosine distance is used for comparing vector distances. For two vectors v_1 and v_2, their distance is defined as $\mathsf{CosineDist}(v_1, v_2) = \cos^{-1}((v_1 \bullet v_2)/(|v_1||v_2|))$, where the symbol $\bullet$ is the dot product between the two vectors, and $|v_1|$ is the length of vector v_1. Cosine distance is essentially a normalized dot product of the vectors, where a particular dimension would contribute to the final sum if and only if both vectors have a nonzero value in that dimension. In our case, each vector represents a particular internal source host, and each dimension represents a linear combination of destination subnets. Cosine distance thus captures well the relationship between internal hosts based on the common destinations they contacted.

Let $\mathsf{numAggs}$ denote the number of clusters from the above algorithm, and let Agg_i ($i = 1 \ldots \mathsf{numAggs}$) denote the hosts comprising the i-th cluster. As such, Agg_i is an aggregate of internal hosts interacting with the same busier-than-usual external subnets. All of $\mathsf{SuspiciousSubnets}$, $\mathsf{numAggs}$ and $\{\mathsf{Agg}_i\}_{1\leq i\leq \mathsf{numAggs}}$ are output from $\mathsf{ByDest}^{\tau}(\Lambda,\Lambda_{\mathsf{past}})$.

2.2.2 Payload Aggregates

Payload inspection algorithms for malware detection have previously focused on either modeling byte-frequency distributions (e.g., [79, 82, 108, 133]), which assumes that malicious traffic should exhibit an observably different byte-frequency distribution from that of normal traffic, or substring matching (e.g., [105, 146]). In contrast to these approaches, our measure of payload similarity is *edit distance with substring moves*, which we choose because it is capable of capturing syntactic similarities between strings, even if parts of one string are simply shifted or replaced. To our knowledge, ours is the first work that detects malicious traffic by computing (a type of) string edit distance between payloads, using techniques that scale these computations to high data rate environments.

For two character strings s_1 and s_2, $\mathsf{EditDist}(s_1, s_2)$ is defined as the number of character insertions, deletions, substitutions, or substring moves, required to turn s_1 into s_2. Given a string $s = s[1]\cdots s[\mathsf{len}(s)]$, a substring move with parameters i, j, and k transforms s into $s[1]\cdots s[i-1], s[j]\cdots s[k-1], s[i]\cdots s[j-1], s[k]\cdots s[\mathsf{len}(s)]$ for some $1 \leq i \leq j \leq k \leq \mathsf{len}(s)$. For example, swapping labeled parameters in a parameter list would be a substring move in a command string.

The payload comparison function $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda)$ that we introduce takes as input a set Λ of communication records, and outputs a value in the range $[0, 1]$. It is parameterized by an edit distance threshold δ_{Ed} that determines if communication records λ, λ' are "close enough", i.e., if $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$. Its output indicates from among all pairs $(\lambda, \lambda') \in \Lambda \times \Lambda$ such that $\lambda.\mathsf{external} = \lambda'.\mathsf{external}$ (i.e., that involve the same external subnet) and $\lambda.\mathsf{internal} \neq \lambda'.\mathsf{internal}$ (i.e., that are not from the same internal host), the (approximate, see below) fraction for which $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$.

Since Λ can be large, computing $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda)$ by computing $\mathsf{EditDist}(\lambda.\mathsf{payload},\ \lambda'.\mathsf{payload})$ for each relevant (λ, λ') pair individually can be prohibitively expensive, i.e., requiring time proportional to $|\Lambda| \cdot |\Lambda|$, where $|\Lambda|$ denotes the cardinality of Λ. A contribution of our work is an algorithm for approximating the fraction of relevant record pairs (λ, λ') that satisfy $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$ in time roughly proportional to $|\Lambda|$ if δ_{Ed} is small.

To perform this approximation, we first *embed* the EditDist metric within L1 distance. For two vectors $v_1 = v_1[1\ldots m]$ and $v_2 = v_2[1\ldots m]$, their L1 distance is defined as $\mathsf{L1Dist}(v_1, v_2) = \sum_{i=1}^{m} |v_1[i] - v_2[i]|$. That is, we transform each $\lambda.\mathsf{payload}$ into a vector v_λ so that if $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq$

δ_{Ed} then $\mathsf{L1Dist}(v_\lambda, v_{\lambda'}) \leq \delta_{\mathsf{L1}}$ for a known value δ_{L1}. We do so using an algorithm due to Cormode et al. [30] called *Edit Sensitive Parsing* (ESP). For this algorithm, the ratio of δ_{L1} over δ_{Ed} is bounded by $O(\log n \log^* n)$, where n is the length of $\lambda.\mathsf{payload}$. [3] In our evaluations, $n = 64$ and we set $\delta_{\mathsf{L1}} = \delta_{\mathsf{Ed}} \cdot \log_{10} 64$.

The embedding of EditDist into L1Dist is essential to our efficiency gains, since it enables us to utilize an approximate nearest-neighbor algorithm called *Locality Sensitive Hashing* (LSH) [39] to find vectors (and hence payload strings) near one another in terms of L1Dist (and hence in terms of EditDist), in time roughly proportional to $|\Lambda|$. Briefly, LSH hashes each vector using several randomly selected hash functions; each hash function maps the vector to a *bucket*. LSH ensures that if $\mathsf{L1Dist}(v_1, v_2) \leq \delta_{\mathsf{L1}}$, then the buckets to which v_1 and v_2 are hashed will overlap with high probability (and will overlap with much lower probability if not), where probabilities are taken with respect to the random selection of the hash functions. Consequently, we hash v_λ for each $\lambda \in \Lambda$, and explicitly confirm that $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$ only for pairs (λ, λ') for which v_λ and $v_{\lambda'}$ hash to at least one overlapping bucket.

2.2.3 Platform Aggregates

Forming traffic aggregates based on platform can be useful in identifying malware infections that are platform dependent. That is, suspicious traffic common to a collection of hosts becomes even more suspicious if the hosts share a common software platform.

Much host platform information can be inferred from traffic observed passively. Passive tools, unlike active fingerprinting tools like Nmap [4], do not probe hosts, but rather observe their communications silently. The most comprehensive passive operating system fingerprinting tool of which we are aware is p0f [5], which extracts various IP and TCP header fields from SYN packets and uses a rule-based comparison algorithm. However, p0f cannot be applied to traffic traces in flow records, since most individual packet information (including for SYN packets) is not retained. Other network intrusion detection systems also employ fingerprints of host software platforms when detecting intrusions, though most generate these fingerprints actively, e.g., building profiles of the network topology to remove ambiguities in how hosts interpret network traffic [131].

[3] $\log^* n$ denotes the *iterated logarithm* of n, i.e., the number of times the logarithm must be iteratively applied before the result is less than or equal to one.

[4] `http://nmap.org`

[5] `http://lcamtuf.coredump.cx/p0f.shtml`

T$\bar{\text{A}}$MD employs two heuristics for fingerprinting internal host operating systems passively. The first employs time-to-live (TTL) fields witnessed at the network border in packets from internal hosts. It is well-known that in many cases, different operating system types select different initial TTL values [6]. With a detailed map of the internal network, the observed TTL values can be used to infer the exact initial TTL value and so narrow the possibilities for operating system the host is running. However, a detailed map is typically unnecessary, as routes in most enterprise networks are sufficiently short that witnessing TTLs of packets as they leave the network enables the initial TTL values to be inferred well enough.

The second heuristic employed in T$\bar{\text{A}}$MD watches for host communications characteristic of a particular operating system platform. For example, Windows machines connect to the Microsoft time server by default during system boot for time synchronization, and the FreeBSD packages FTP server is more likely to be accessed by FreeBSD machines to install software updates. Once characteristic communications for platforms are identified, T$\bar{\text{A}}$MD can monitor for these to learn the platform of an internal host.

There are at least three limitations of such passive fingerprinting approaches for our purposes. First, DHCP-assigned IP addresses can be assigned to hosts with different operating systems over time, leading to inconsistent indications of the host operating system associated with an IP address. This suggests that T$\bar{\text{A}}$MD should weigh recent indications more heavily than older (and hence potentially stale) indications. Second, a machine with a compromised kernel could, in theory, alter its behavior to masquerade as a different operating system. In the absence of a possible IP address reassignment (e.g., for address ranges not assigned via DHCP), such a shift in behavior should itself be detectable evidence that a compromise may have occurred. In general, however, this limitation is intrinsic to *any* fingerprinting technique, passive or active, except those based on attestations from trusted hardware (e.g., TCG's Trusted Platform Module [7]). While we are unaware of malware that employs such a masquerading strategy, should platform-based aggregation for malware detection become commonplace, such systems would presumably need to migrate to attestation-based platform identification as it matures, in order to detect kernel-level compromises. The third limitation to forming aggregates based on platform is that it is likely for an enterprise to have the majority of its hosts running the same operating system. Thus ByPlatform would be more effective for networks with a diverse host population; for example, in a university setting.

T$\bar{\text{A}}$MD uses the aforementioned heuristics based on TTL values and communication with characteristic sites to identify platforms. We embody this in a function $\mathsf{ByPlatform}(\Lambda)$ that returns the largest fraction of internal hosts in Λ (i.e., among the hosts $\{\lambda.\mathsf{internal} : \lambda \in \Lambda\}$) that can be identified as having the same

[6] http://www.binbert.com/blog/2009/12/default-time-to-live-ttl-values/

[7] https://www.trustedcomputinggroup.org/developers/trusted_platform_module

operating system, based on these heuristics applied to the traffic records Λ.

2.3 Example Configuration

We detail a configuration of T$\bar{\text{A}}$MD that identifies internal hosts infected by malware by employing the aggregation functions $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda)$, and $\mathsf{ByPlatform}(\Lambda)$. This configuration identifies platform-dependent malware infections that report to common sites, e.g., IRC channels for receiving commands, public servers for downloading binaries, denial-of-service victims to attack, or database servers for uploading stolen information, and is based on several observations about such malware:

O1. For even moderately aggressive malware, it is rarely the case that only a single victim exists in a large enterprise network, and so we hypothesize that stealthy malware is likely to generate traffic that appears within the same, coarse window of time (e.g., within the same hour) from multiple infected hosts. Moreover, we would expect that the controller site is located in a subnet that would not be a common one with which benign hosts interact, as major services with substantial client populations are typically better managed. As such, infected hosts interacting with the controller site should generate a noticeable increase in the number of interactions with the controller's subnet in that window of time.

O2. We expect that the multiple instances of the malware communication to the controller site would be syntactically similar to each other, since the malware instances are communicating using the same protocol, and likely to be receiving or responding to similar commands.

O3. In the case of platform-dependent malware, the malware communications to the controller site will involve internal hosts all having the same host platform.

Using these observations, we have assembled the aggregation functions described in Section 2.2 into an algorithm $\mathsf{FindSuspiciousAggregates}$ to identify such malware infections, shown in Figure 2.1. The input to this function is a set Λ of traffic records observed in a fixed time interval (e.g., one hour) at the border of the network, and a set Λ_{past} of records previously observed at the border of the network. $\mathsf{FindSuspiciousAggregates}$ assembles and returns (in line 108) a set $\mathsf{SuspiciousAggregates}$ comprised of suspicious aggregates, where each aggregate is a set of internal hosts (IP addresses) that is suspected of being infected by malware.

```
$\mathsf{FindSuspiciousAggregates}(\Lambda, \Lambda_{\mathsf{past}})$
100: $\mathsf{SuspiciousAggregates} \leftarrow \emptyset$
101: $(\mathsf{SuspiciousSubnets}, \mathsf{numAggs}, \{\mathsf{Agg}_i\}_{1 \le i \le \mathsf{numAggs}}) \leftarrow \mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$
                                                  /* Form aggregates by external subnet */
102: for $i = 1 \ldots \mathsf{numAggs}$ do
103:   $\Lambda_i \leftarrow \{\lambda \in \Lambda : \lambda.\mathsf{internal} \in \mathsf{Agg}_i\}$        /* Traffic from hosts in $\mathsf{Agg}_i$ */
104:   $\Lambda_i^{\mathsf{susp}} \leftarrow \{\lambda \in \Lambda_i : \lambda.\mathsf{external} \in \mathsf{SuspiciousSubnets}\}$
                                  /* Traffic from hosts in $\mathsf{Agg}_i$ to suspicious subnets */
105:     if $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda_i^{\mathsf{susp}}) > 0.3$ then
                           /* Keep if traffic to same external subnet is self-similar */
106:       if $\mathsf{ByPlatform}(\Lambda_i^{\mathsf{susp}}) > 0.9$ then
                              /* Keep if most of aggregate consists of one platform */
107:         $\mathsf{SuspiciousAggregates} \leftarrow \mathsf{SuspiciousAggregates} \cup \{\mathsf{Agg}_i\}$
108: return $\mathsf{SuspiciousAggregates}$
```

Figure 2.1: The function used to find suspicious aggregates in the example configuration. ByDest^{τ} (line 101), $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ (line 105), and $\mathsf{ByPlatform}$ (line 106) are defined in Sections 2.2.1, 2.2.2 and 2.2.3, respectively.

$\mathsf{FindSuspiciousAggregates}$ first exploits observation O1, using ByDest^{τ} from Section 2.2.1 to find suspicious external subnets $\mathsf{SuspiciousSubnets}$ responsible for noticeably greater communication with the monitored network than in the past, and to find aggregates $\{\mathsf{Agg}_i\}_{1 \le i \le \mathsf{numAggs}}$, each of which includes internal hosts that interacted with one or more of these subnets. In line with observation O2, each aggregate is tested in line 105 to determine if distinct hosts in the aggregate communicate with suspicious subnets using similar payload. Finally, as motivated by observation O3, for each aggregate that has survived these tests, the platforms of the hosts in the aggregate are inferred using $\mathsf{ByPlatform}$ and, if the aggregate is adequately homogeneous (line 106), then it is added to $\mathsf{SuspiciousAggregates}$ (line 107).

There are numerous constants in Figure 2.1 that we have chosen on the basis of our evaluation that we will present in the next section. These constants include $\tau = 90\%$ or 95% for ByDest^{τ}, 0.3 in line 105 and 0.9 in line 106. In addition, as we will describe in Section 2.4, the data on which we perform our evaluation includes 64 bytes of payload per record λ, for which we found $\delta_{\mathsf{Ed}} = 15$ to be an effective value. We show that with traffic generated from real spyware and bot instances, and traces from real bots captured in a honeynet, this configuration of TĀMD was able to reliably extract malware traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low. This reliability is achieved even in tests where the number of simulated infected hosts comprise only about 0.0097% of the total number of internal hosts in the network, calculated as the maximum number of internal IP addresses observed communicating in any one hour period during our data collection (see

Section 2.4), which was over 33,000.

2.4 Evaluation

We present an evaluation of the particular configuration of TĀMD, described in Section 2.3, using traffic from real spyware and bot instances, which were overlaid onto flow records recorded at the edge of our university campus network.

2.4.1 Data Collection

Our network traffic traces were obtained from the edge routers on the Carnegie Mellon University campus network, which consists of two /16 subnets. The packets were organized into bi-directional flow records by Argus (Audit Record Generation and Utilization System) [8], which is a real time flow monitor based on the RTFM flow model [18, 61]. Argus inspects each packet and groups together those with the same attribute values into one bi-directional record. In particular, TCP and UDP flows are identified by the 5-tuple (source IP address, destination IP address, source port, destination port, protocol) [9], and packets in both directions are recorded as a summary of the communication, namely, an Argus flow record.

The fields extracted from Argus records are listed in Table 2.1. The rate of the traffic from the edge of our campus network is about 5,000 flow records per second. The traces were collected daily from 9 AM to 3 PM for three weeks in November and December 2007. In our evaluation, we focused on TCP and UDP traffic.

IP Header	Transport Header	Flow Attribute
Source IP	Source Port	Byte Count
Destination IP	Destination Port	Packet Count
Protocol	TCP Sequence Number	Payload (64 bytes)
TTL	TCP Window Size	

Table 2.1: Extracted Flow Fields

We also obtained network traffic traces for several malware. The malware traces used for testing are grouped into two sets, Class-I and Class-II, as described below.

Class-I Traces. We obtained four instances of malware: Bagle, IRCbot, Mybot and SDbot, and collected their traffic by infecting virtual machines hosts with each malware. The virtual hosts were all running

[8] `http://www.qosient.com/argus`

[9] Since Argus records are bi-directional, the source and destination IP addresses are swappable in the logic that matches packets to flows. However, the source IP address in the record is set to the IP address of the host that initiated the connection.

the Windows XP Professional operating system with the same VMWare image file. Each run of traffic collection was one hour long, and included the communications from eight instances of Bagle, three instances of IRCbot, five instances of Mybot, or five instances of SDbot. These numbers of instances were chosen to represent a very small fraction of the total campus hosts, specifically at most 0.0097% based upon the number of campus hosts observed sending traffic in the busiest hour, which has over 33,000 distinct IP addresses.

Briefly, the characteristics of these four malware instances are as follows:

1. **Bagle**[10] is spyware that, on execution, runs as a background process and attempts to download other malicious executables from various sites, while generating pop-up windows and hijacking the web browser to advertising websites. As with other types of spyware and adware, Bagle initiates connections to numerous destinations that are set up to exclusively host advertisements or other malicious content.

2. **IRCbot** [11] is a backdoor trojan that connects to an IRC server and waits for commands from the bot-master. In addition, after successfully connecting to the command-and-control server, the bot downloads an update executable from a designated web server, and goes on to scan the local /16 subnet attacking other machines with the LSASS vulnerability on port 445[12] and the NetBIOS vulnerability on port 139 [13].

3. **Mybot**[14] is spyware, a worm, and a bot that connects to an IRC server to wait for commands, and also records keystrokes and steals other personal information on the victim host. This malware is especially subtle in its communications. When it is only waiting for commands on the IRC server, the bot initiates one connection every 90 seconds, in the form of IRC PING/PONG messages. In the hour of our traffic collection, Mybot simply waited for commands on the IRC channel, and its only outbound connections were these PING/PONG messages.

4. **SDbot** [15] is a trojan and a bot that opens a back door to connect to an IRC server. Similar to Mybot, when it is waiting for commands from the attacker, SDbot only makes outbound connections once every 90 seconds, in the form of IRC PING/PONG messages.

[10] `http://www.trendmicro.com/vinfo/virusencyclo`.
[11] `http://www.symantec.com/security_response/writeup.jsp?docid=2002-070818-0630-99`.
[12] `http://www.microsoft.com/technet/security/Bulletin/MS04-044.mspx`.
[13] `http://support.microsoft.com/kb/269239`.
[14] `http://www.sophos.com/security/analyses/w32rbotxf.html`.
[15] `http://www.symantec.com/security_response/writeup.jsp?docid=2003-050714-1919-99`.

Class-II Traces. We also obtained network traces of botnets gathered from honeynets, including the IRC-based Spybot, a HTTP-based botnet [55], and a large IRC botnet from a honeynet running in the wild [54]. The Spybot trace contains communications from four bots for the duration of 32 minutes; the HTTP-bot trace contains communications from four bots over the course of three hours; and the large botnet trace contains traffic from more than three hundred bots over seven minutes.

For testing, we overlaid flows from these malware instances onto one hour of our recorded campus network traffic, and assigned the malware traffic to originate from randomly selected internal hosts observed to be active during that hour. More specifically, we assigned malware traffic to random internal hosts running the operating system that the malware exploits, as determined by the time-to-live (TTL) field in packets. This makes our testing scenario much more realistic, since the internal hosts to be identified still exhibit their normal connection patterns, in addition to subtle malware activities. While certain aspects of the hosts' network behaviors (e.g., the volume of traffic generated by the host) would be affected by this method of overlaying malware traffic, it does not impact the characteristics on which TĀMD operates. We hence believe that this experiment setup is adequate for the present purpose.

For the trace that spans multiple contiguous hours, i.e., the HTTP botnet trace, we overlaid it onto the same number of contiguous hours in the campus network traffic, performed analysis on each of the hours "covered" by the malware trace, and reported the hour that TĀMD detected the malware aggregate. This time window was then shifted by one hour, and the experiment repeated until we reached the end date of our campus traffic collection.

2.4.2 Outlier Hosts

In the early stages of our analysis, we found that often TĀMD failed to detect the malware-laden hosts, but rather identified other internal hosts as more symptomatic of malware. Upon further inspection, we identified the internal hosts that resulted in these false alarms: PlanetLab nodes [25] and a Tor node [142].

In the case of PlanetLab nodes, we noticed that during the destination aggregation function, the vectors after PCA analysis often had very low dimensionality, e.g., two, where a small number of principal components were able to capture over 90% of the data variance. Clustering these vectors resulted in a few outliers forming their own individual clusters, unlike any of the vectors from Λ_{past} (the "old vectors"), or even those in Λ (i.e., the "new vectors"). This is shown in Figure 2.2. The two axes correspond to the top two principal components onto which the original data is projected. Cluster hubs are shown in rectangles. The outliers forming their own individual clusters were found to be PlanetLab nodes, which, being

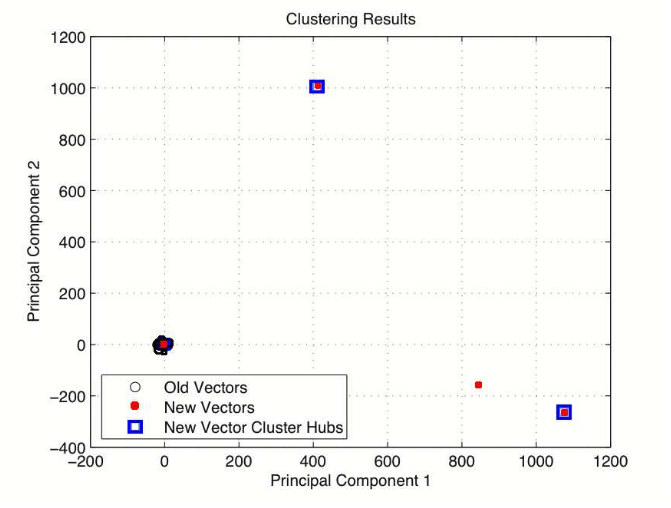


Figure 2.2: Clustering results after dimension reduction by PCA. The three outliers were found to be PlanetLab nodes.

a development and testing platform, exhibit behavior deviating from that of other hosts. Their existence was also the reason why PCA analysis was able to reduce the vector dimensionality down to only two, since PlanetLab nodes' behavior is so different from other hosts that only two principal components were needed to capture most of the data variance.

In another example from experiments involving the Bagle trojan spyware, we noticed that even though T$\bar{\text{A}}$MD was able to form a final aggregate containing all spyware traffic and spyware traffic only, at times it also combined another unknown host into the spyware-hosts aggregate, both in the ByDest$^{\tau}$ and the ByPayload$^{\delta_{\text{Ed}}}$ functions. Similar investigations revealed that this additional node is a Tor router inside the campus network. Tor offers online anonymity by routing packets over random routes between Tor servers so that the source and destination of the packet is obfuscated. Because the traffic originates from multiple hosts, it is possible that, even though the Tor router itself is not infected, another host routing traffic through the Tor node may be a spyware victim.

For this work, we removed PlanetLab and Tor nodes from our analysis.

2.4.3 Detecting Malware

As described in Section 2.4.1, T$\bar{\text{A}}$MD was given all TCP and UDP traffic collected at the edge of our university network in hourly batches, overlaid with malware traffic assigned to randomly selected internal hosts. The same analysis steps were repeated for each hour over three weeks in November and December 2007.

The granularity of external destinations was set to be /24 subnets. While the communication records from the current hour were given to $\mathsf{FindSuspiciousAggregates}$ as Λ, the set Λ_{past} was selected from communication records in the past (specifically, from the beginning of our traffic collection dating to the first week of September 2007) that represented the general trend and the periodicity in the traffic. Specifically, Λ_{past} consisted of traffic from, in reference to the time frame for Λ, (i) the same hour from the same days of the week, (ii) the same hour from the same days of the month, (iii) the same hour from the previous two days, and (iv) the previous two hours. For example, if Λ consists of traffic from 2 to 3 PM on Wednesday, November 28th, then Λ_{past} will include traffic from 2 to 3 PM every Wednesday before that, from 2 to 3 PM in the previous two days (November 27th and 26th), and from 12 to 2 PM on November 28th.

In all experiments, T$\bar{\text{A}}$MD was able to identify all the infected hosts (with the exception of the Class-II large IRC trace, as described later) while the number of additional aggregates reported was only 1.23 per hour on average. For the Class-II HTTP-botnet trace that spans multiple hours, T$\bar{\text{A}}$MD always detected the infected hosts in the very first hour. For the case of the Class-II large IRC botnet trace, which contains 340 infected bots, T$\bar{\text{A}}$MD was able to identify 87.5% of the bots on average, and these bots were all grouped in a single aggregate. We suspect that the reason not every bot in the botnet was detected is due to the randomness in our choice of selected internal hosts to which the malware traffic was assigned, such that a selected internal host that was also contacting other suspicious subnets (not relevant to the botnet) is likely to bias the dimension reduction and clustering algorithms.

Figure 2.3 shows, for each malware experiment (the rows), the number of aggregates remaining after applying each aggregation function (the columns), averaged over all test hours. The number of aggregates is reduced after each aggregation function, as they become more refined to satisfy multiple characteristics. The single aggregate consisting solely of infected hosts was always identified, in every malware experiment. As shown in the figure, even for homogeneous networks where the majority of internal hosts are of the same platform, applying ByDest^{τ} and $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ would still yield good results.

2.4.4 Unknown Aggregates

As indicated in Figure 2.3, our methodology detected a small number of unknown aggregates (about 1.23 per hour, on average) in addition to the one aggregate of infected hosts that we overlaid on the trace. We found that some of these same unknown aggregates regularly appeared for that hour of input data, across different malware experiments. Further investigation based on the 64 bytes of flow payload available to

Malware traces	ByDest$^{\tau}$ (line 101)	ByPayload$^{\delta_{Ed}}$ (line 105)	ByPlatform (line 106)
Class-I			
Bagle	47.46 (± 23.13)	4.19 (± 2.34)	2.55 (± 1.33)
IRCbot	35.10 (± 20.51)	2.74 (± 1.41)	1.98 (± 0.98)
Mybot	45.60 (± 25.10)	3.19 (± 1.76)	2.13 (± 1.09)
SDbot	52.15 (± 43.87)	3.55 (± 1.88)	2.34 (± 1.16)
Class-II			
Spybot	39.18 (± 22.31)	2.95 (± 1.44)	2.04 (± 0.92)
HTTP bot	53.97 (± 26.54)	3.31 (± 1.91)	2.22 (± 1.21)
Large IRC bot	44.54 (± 16.16)	4.39 (± 2.75)	2.39 (± 1.32)

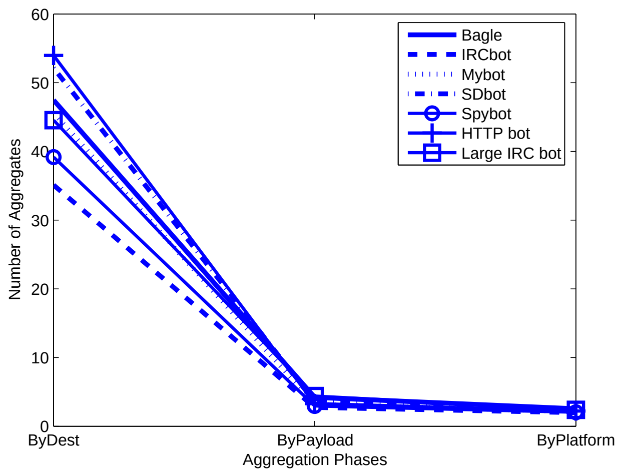


Figure 2.3: Mean number of aggregates (± standard deviation) remaining after each function in Figure 2.1

us, port numbers, and protocol field (for privacy reasons, the IP addresses were anonymized), showed that these aggregates included NetBIOS messages on port 137, DNS name server queries, SMTP connection timeout messages, and advertising-related HTTP requests; several of these suggest that additional investigation may be warranted. Others included connections to online game servers and large flows over non-standard ports, which we suspect to be peer-to-peer (P2P) transfers. All of these aggregates consisted of internal hosts contacting rare sites, and often include less than five hosts sharing one or two common destination subnets.

In theory, a group of internal hosts visiting a new popular website (i.e., the "slashdot" effect) could also form an aggregate. However, it is unlikely that all of the hosts would come from the same platform, and in our experiments, we believe we saw very few such aggregates. We thus believe that T$\bar{\text{A}}$MD is a useful data reduction tool for malware identification.

2.4.5 Performance

The left half of Table 2.2 shows the run times in seconds for each aggregation function and for each malware instance, averaged over the traffic (in one-hour intervals) we used to perform our experiments. In our implementation of $\text{T}\bar{\text{A}}\text{MD}$, ByDest^{τ} is implemented in Matlab, and $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ and $\mathsf{ByPlatform}$ are implemented in C. For the numbers reported in Table 2.2, ByDest^{τ} was executed on a PC with a Pentium IV 3.2 GHz processor and 3 GB of RAM, and $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ and $\mathsf{ByPlatform}$ were executed on a Dell PowerEdge server with dual core 3 GHz processors and 4 GB of RAM.

Malware traces	ByDest^{τ} (line 101)	$\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ and $\mathsf{ByPlatform}$ (lines 105, 106)	Total time	Size of $\mathsf{SuspiciousSubnets}$	Internal hosts contacting $\mathsf{SuspiciousSubnets}$
Class-I					
Bagle	79.48 (± 264.54)	14.08 (± 18.07)	93.48 (± 271.51)	701.87 (± 596.78)	754.73 (± 812.75)
IRCBot	94.67 (± 350.13)	20.19 (± 16.78)	114.86 (± 356.72)	927.23 (± 561.33)	742.13 (± 836.55)
Mybot	63.82 (± 177.34)	10.96 (± 15.28)	70.93 (± 183.43)	686.03 (± 565.11)	728.45 (± 708.65)
SDbot	102.34 (± 355.25)	10.21 (± 19.51)	112.55 (± 359.23)	749.01 (± 577.49)	952.96 (± 1191.66)
Class-II					
Spybot	86.30 (± 276.81)	63.42 (± 38.56)	151.15 (± 286.99)	850.14 (± 609.19)	777.71 (± 848.43)
HTTP bot	83.12 (± 278.76)	15.75 (± 20.62)	99.31 (± 287.11)	697.36 (± 609.15)	776.76 (± 848.43)
Large IRC Bot	110.64 (± 253.78)	46.00 (± 34.78)	156.64 (± 260.42)	760.83 (± 548.48)	1104.42 (± 799.58)

Table 2.2: Mean run times of each phase in seconds of algorithm in Figure 2.1 and means of measures impacting performance (± std. dev.)

The running times of the aggregation functions depend on several factors, including the number of external destinations identified as suspicious (i.e., $\mathsf{SuspiciousSubnets}$ as computed by ByDest^{τ}) and the number of flows to those suspicious destinations; averages for these numbers are also listed in the right half of Table 2.2. The amount of traffic in Λ_{past} is especially critical to the performance of $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$, since it accesses significant amounts of historical data (i.e., Λ_{past}) to define the “normal” behavior for this network. While the implementation of $\text{T}\bar{\text{A}}\text{MD}$ is not optimized, retrieving historical data from the database contributed to the majority of the slowdown. This problem can be alleviated in the future by performing these calculations in advance and update incrementally as more data is collected.

2.5 Discussion

In this section, we discuss potential limitations to $\text{T}\bar{\text{A}}\text{MD}$, and explore alternative ways of assembling the three aggregation functions ByDest^{τ}, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$, and $\mathsf{ByPlatform}$.

2.5.1 Potential Limitations

An approach by which malware writers might attempt to avoid detection by our technique is to encrypt their malware traffic, so that our payload comparisons will be ineffective. To accommodate encryption, our technique can be generalized to define encrypted content (which itself is generally easy to detect) as "similar". Malware writers could go further and have their malware communicate steganographically, though at the cost of greater sophistication and lower bandwidth. Detecting steganographic communication is itself an active area of research (e.g., [118]) from which TĀMD could benefit. However, a better method perhaps is to move away from payload inspection altogether, since such information is not typically included in flow records, e.g., NetFlow records [1].

Another way that malware writers could try to avoid detection is by exploiting cross-platform applications (e.g., web browsers), so that the malware is no longer operating system-dependent. We address application-dependent malware in Chapter 3, and particularly focus on web browser-dependent malware, since browsers are one of the most common attack vectors today.

Bot-masters can also evade TĀMD by utilizing alternative botnet architectures, such as those built over P2P networks. However, bots participating in other architectures may still exhibit characteristics that should be detectable by TĀMD. For example, the Storm botnet [52, 66, 116, 138], while using a P2P network to transfer addresses of compromised web servers among the infected hosts, still require bots to connect to those web servers to download malicious executables for sending spam or performing denial-of-service attacks. This activity of collectively contacting web servers matches the behavior that our technique successfully detected in our evaluations. Vogt et al. [145] suggested a "super-botnet", where the botnet is composed of individual smaller centralized botnets, and the controllers from each smaller botnet peer together in a P2P network. Since the individual smaller botnets still use a centralized architecture, this should be still be detectable via our technique. Wang et al. [147] proposed a hybrid P2P botnet where each bot maintains its own peer list and polls other bots periodically for new commands. However, in order to monitor the IP addresses and resources associated with the bots, the botnet supports a command by which the bot-master can solicit all bots to report to a specific server. Again, this behavior should be detectable by TĀMD.

That said, P2P bots can still avoid contacting a common server for the transfer of executables or other tasks. For example, Phatbots find peers by registering themselves as Gnutella clients, and the Sinit trojan sends out random probes for peer discovery [135]. In these cases, forming aggregates based on payload similarity should remain effective, provided that similarity is generalized as described above to accommodate encrypted traffic (which Phatbot utilizes). However, a more challenging aspect of their detection is in distinguishing P2P bot traffic from those of P2P file-sharing hosts. We address this challenge in Chapter 4.

2.5.2 Alternative Configurations

The default T$\overline{\text{A}}$MD configuration, described in Section 2.3, assembles the aggregation functions ByDest^{τ}, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$, $\mathsf{ByPlatform}$, in that order. The priority given to the common destination characteristic (in ByDest^{τ}) is motivated by the centralized nature in how such malware communicates with the bot-master. Furthermore, the relatively expensive computations required by $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ and the coarse granularity of $\mathsf{ByPlatform}$ also allows the default configuration to be more efficient and accurate compared to others that apply $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ or $\mathsf{ByPlatform}$ initially.

To illustrate how the ordering of the aggregation functions affects T$\overline{\text{A}}$MD's ability at detecting malware, we repeated our experiments for each of the malware traces described in Section 2.4 overlaid on one day of traffic collected from the border of the Carnegie Mellon University campus. Figure 2.4 shows the number of aggregates identified by each T$\overline{\text{A}}$MD configuration, averaged over all malware experiments. The first aggregation function applied largely determines the final number of aggregates output by T$\overline{\text{A}}$MD, since functions applied in later stages simply refine identified aggregates. We omit plotting cases when platform aggregates are formed first (using $\mathsf{ByPlatform}$) as this characteristic is too coarse to allow aggregates to be refined successfully.

Among the configurations we explore, the default (i.e., ByDest^{τ}, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$, $\mathsf{ByPlatform}$) results in the least number of unknown aggregates (outside of the single aggregate containing the overlaid malware traffic) that require additional inspection by network administrators.

2.6 Chapter Summary

In this chapter, we presented T$\overline{\text{A}}$MD, a system that identifies hosts internal to a network that are infected by stealthy malware. Our approach is to find those hosts that share unusual network communications. In particular, T$\overline{\text{A}}$MD employs three aggregation functions to group hosts based on the following characteristics. First, the destination aggregation function, ByDest^{τ}, forms aggregates of internal hosts that contact the same combination of busier-than-usual external destinations. A binary vector is formed for each internal host, with each dimension representing one of the selected external destinations. The vectors are processed by PCA for dimension reduction, and clustered by a modified version of K-means clustering.

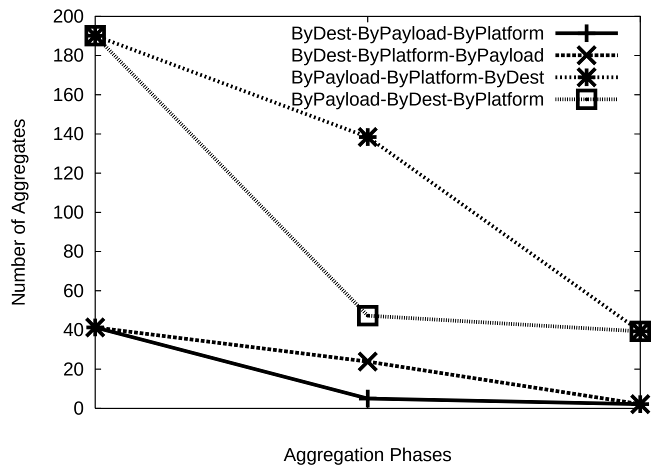


Figure 2.4: The number of aggregates identified by alternative T$\bar{\text{A}}$MD configurations, averaged over all malware experiments over one day of traffic.

New clusters are selected as those that do not conform to preceding communication patterns. Second, the payload aggregation function, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$, identifies communications with similar payloads in terms of a type of edit distance. This is done by first embedding the payload strings into vectors in L1 space, and then finding close vectors by an approximate nearest-neighbor algorithm. Third, the platform aggregation function, ByPlatform, forms aggregates that involve hosts running on common operating systems, as inferred using TTL (time-to-live) values or platform-specific sites to which they connect.

We detailed a configuration of T$\bar{\text{A}}$MD that employs these functions in combination to identify platform-dependent malware infections that report to common sites. A common site might be an IRC channel for receiving commands, a public web server for downloading binaries, a denial-of-service victim they are instructed to attack, or a database server for uploading stolen information, as is typical of most bots and spyware. Our experiments show that, with traffic generated from real spyware and bot instances, this configuration of T$\bar{\text{A}}$MD reliably extracted malware traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low. This is achieved even in tests where the number of simulated infected hosts comprised only about 0.0097% of over 33,000 internal hosts in the network.

Chapter 3

Browser Fingerprinting from Coarse Traffic Summaries

One of the characteristics on which the TĀMD system aggregates traffic is the *platform* of the internal hosts involved in sending or receiving that traffic, which is useful for identifying platform-dependent malware infections. That is, suspicious traffic common to a collection of hosts becomes even more suspicious if the hosts share a common software platform. Previously, forming platform aggregates in TĀMD was based solely on the hosts' operating systems. As such, malware that is application-dependent, might span multiple aggregates formed by O/S fingerprinting alone (if the exploit works on multiple operating systems) or might represent a small subset of an O/S aggregate (e.g., all Windows machines). In either case, the mismatch between the software fingerprinted (the O/S) and the software exploited (the application) can cause platform aggregation to fail to detect an exploit.

We address such malware by focusing on a specific application, namely web browsers. Our focus on web browsers is partly due to their relative importance among applications today, but is also due to the proliferation of attacks that exploit their vulnerabilities. By extending the platform aggregation in TĀMD to incorporate browser fingerprinting, we aim to detect both operating system-dependent and browser-dependent malware.

While the browser implementation on a client host is transmitted to the web server as part of the HTTP request header, we do not make use of the payload information to perform browser fingerprinting. Our reason for evading payload inspection is two-fold: 1) It is becoming increasingly common for malware to encrypt their communications; 2) Payload information is not typically recorded in flow records (e.g., NetFlow [1]). Rather, we construct a browser fingerprinting technique that utilizes only the start and end times of each flow, as well as the byte and packet counts. We show that this information, available in coarse flow records, reflect behavioral features of the traffic in which browsers are involved when interacting with regular sites, hence allowing browser implementations to be distinguished. It is arguably surprising that browsers could be discerned in this way, since a browser's network behavior is primarily determined by the content and structure of the pages it accesses. Moreover, classification could be complicated by various factors that are inherent in traffic, including variations in users' browsing behavior or browser configuration, differences in the web page content being retrieved (both across different websites and in the same website over time), the client hardware configuration, and the different geographic locations from which the content is retrieved. A contribution of this work is in evaluating the impact of the above factors on the browser classification accuracy.

In this chapter, we first describe the construction of our browser fingerprinting technique, and present evaluations performed on real user traffic collected from the edge routers of our university campus network. We then incorporate browser fingerprinting into the platform aggregation function in TĀMD and quantify the resulting overhead.

3.1 Related Work

Many fingerprinting tools are *active* in nature, probing services with carefully crafted queries (e.g., those produced by `Nmap` [1]) to detect implementation-specific characteristics [28, 107]. More relevant to our work are *passive* fingerprinting techniques that infer the implementations of network applications or operating systems based solely on observing the traffic they send. Passive fingerprinting tools and techniques are numerous, though most focus on identifying TCP/IP implementations and utilize specific information [13, 92, 112] that is unavailable in coarse flow records. While passive techniques have more recently been proposed to identify the *application* (e.g., peer-to-peer file transfers versus web retrievals) or the class of application (e.g., interactive sessions versus bulk-data transfers) reflected in packet traces [12, 33, 62, 78, 124], few proposals (e.g., [26, 44, 99, 159]) have done so from coarse flow records. Moreover, to the best of our knowledge, none of these proposed techniques attempt to identify particular *implementations* of an application (e.g., the browser) from passive observations of flow records alone.

[1]`http://nmap.org`

3.2 Data Sets

Our analysis takes advantage of several sources of data recorded in the Argus (Audit Record Generation and Utilization System [2]) flow format, similar to those used in the evaluation of T$\bar{\text{A}}$MD in Section 2.4. The browser fingerprinting technique we describe in this chapter requires only that each flow record include the source and destination IP addresses and ports, the protocol, and the total bytes and packets sent in each direction. While the Argus records that are available to us also include the first 64 bytes of payload from each flow record, we use this information solely for the purpose of determining ground truth of certain attributes to use in our evaluation. To be clear, this additional information is not used by our classifiers, and is only taken into consideration when determining the accuracy of our techniques and for extracting testing instances from live network data.

We use the following data sources in our evaluations:

The CMU dataset. Similar to that used in the evaluation of T$\bar{\text{A}}$MD in Section 2.4, this dataset consists of anonymized traffic from the edge routers of the wired CMU campus network, which includes one /16 subnet. We do not consider hosts (that is, IP addresses) from the wireless network, since those hosts typically have short-lived DHCP-assigned IP addresses, such that hosts using different browsers may be associated with the same address, leading to inconsistencies in the data. The rate of the traffic in the CMU dataset is about 5,000 flow records per second, and was collected daily from 9 AM to 3 PM over six weeks from October to December 2007. We are interested in reducing this dataset only to web retrievals for the purposes of evaluating our browser classifier, but one of the challenges in processing live network data is in accurately identifying the boundaries that separate website retrievals (c.f., [32, 134]). Here we leverage the first 64 bytes of each flow to identify the start boundary of a website retrieval from a host internal to the CMU network. More specifically, we define a web retrieval to begin with a port-80 connection comprised of an HTTP request of the form “GET / ”, as such a connection would be highly unlikely to be part of another retrieval. The web retrieval is then comprised of this flow and all subsequent flows originating from the same host in the next 10 seconds. Our choice of 10 seconds is based on empirical evaluations. The use of the flow payload for parsing web retrievals can be replaced, for example, by checking for a certain amount of idle time before a burst of web traffic [85], though we do not explore this alternative here. Incomplete retrievals, or those with less than three flows, do not carry enough information about the browser implementation in order for the classifier to make a well-grounded decision, and so we only

[2] `http://www.qosient.com/argus`

consider retrievals with more than three flows in our analysis.

As mentioned earlier, we examine the 64 bytes of available payload in each flow to infer the browser involved in the retrieval. Specifically, for the purposes of ground-truth, a host is identified to be using the Opera browser if the user-agent string in its HTTP request starts with the string “Opera”. Firefox hosts are identified by the special “safe-browsing” requests issued by the browser to check the validity of the website being contacted [3]. Due to the 64-byte restriction in the available payload length, we were not able to reliably identify hosts using IE and Safari in the CMU data set.

The PlanetLab-Native dataset. In order to perform our evaluations in which the CMU dataset serves as the testing data, we would like a training dataset from hosts that are diverse in terms of geography and hardware platform. PlanetLab [25] offers a platform that is generally available and that enables the retrieval of web pages from a wide range of hosts with different hardware configurations and geographic locations. To collect this dataset, we deployed a program to fourteen hosts across five PlanetLab networks; this program sequentially retrieved the front page (i.e., generating “GET / ” HTTP requests) of the top 150 most popular websites in the U.S. [4]. repeatedly over the course of one month. Each web retrieval was comprised of the flows observed in the thirty seconds since the start of the retrieval. Machines on PlanetLab are required to run a Linux operating system, so we performed retrievals from Linux-compatible browsers, namely Firefox and Opera[5]. Recall that these two browsers are also the only ones reliably identifiable in the CMU dataset, and so the PlanetLab-Native dataset can serve well as training data for testing with the CMU dataset.

The PlanetLab-QEMU dataset. In an effort to develop a dataset that includes traffic for all of the major browsers (IE, Firefox, Opera and Safari), we utilized a processor emulator, QEMU [11], to run an emulated Windows operating system on PlanetLab hosts. As in the PlanetLab-Native dataset, we ran an automated program to sequentially retrieve the front page of the top 150 most popular websites in the U.S. repeatedly over the course of one month. Each web retrieval was comprised of the flows observed in the thirty seconds since the start of the retrieval. We deployed this emulated version of Windows on seven hosts across three PlanetLab networks[6].

Arguably, the PlanetLab datasets may not accurately represent website retrievals generated by actual

[3] `http://www.mozilla.com/en-US/firefox/phishing-protection/`

[4] According to `alexa.com`

[5] To generate our PlanetLab-Native dataset, we used Firefox 2.0.0.16 and Opera 9.51.

[6] To generate our PlanetLab-QEMU dataset, we used IE 7.0, Firefox 2.0.0.13, Opera 9.51 and Safari 3.1.

user activities, where frequent visits to a particular website may result in much of the content being cached. To compensate for this effect, we set the browser cache sizes to be sufficiently large (400 MB) so that objects are less likely to be evicted from the cache.

Flow Statistics	Byte count (in each direction)
	Packet count (in each direction)
	Flow duration
	Number of flows active simultaneously to this one
	Start time minus most closely preceding flow start time
Retrieval Statistics	Total number of flows
	Cumulative byte count from destination
	Cumulative flow duration
	Retrieval duration

Table 3.1: Main features extracted for each web retrieval.

Feature Selection

To capture browser-specific characteristics in network traffic, we extracted nine main features from each website retrieval, listed in Table 3.1. The mean, standard deviation, maximum, minimum, median, first and third quartile, inter-quartile range, and the cumulative sum, were also calculated for each flow statistic. Our feature selection strategy is based on examining the information gain associated with each of the statistics for the aforementioned nine main features. More specifically, using the PlanetLab-Native dataset, we selected the top statistics whose cumulative information gain accounted for at least 90% of the overall information gain. These selected statistics were combined into a feature vector F_r for website retrieval r.

Among the most important features are those associated with the byte and packet counts in each direction, the cumulative flow duration, and the retrieval duration. While we have not fully explored the root cause for all of these differences, they are related to the different orders in which the browsers retrieve objects on a given page, different numbers of objects retrieved in one connection, and the numbers of connections that can be active simultaneously.

Of course, while these features play an important role in distinguishing different browser implementations in our tests, we acknowledge that they may not be optimal for distinguishing browsers not included in the training data, or future browser versions that behave fundamentally differently from the ones covered in this study. That said, the methodology outlined in this chapter can be easily applied to incorporate new browser types into the classifier.

3.3 Browser Identification from Flow Records

As discussed earlier in this chapter, our goal is to develop techniques for inferring the browser implementation from web traffic recorded in the form of flow records. At first, it might seem that distinguishing the browser should be difficult, since a browser primarily serves to interpret and render the HTML and other types of content it receives. As such, its behavior should be primarily dictated by the content it is accessing.

An example of why this intuition might not be true is shown in Figures 3.1 and 3.2. Each figure shows one feature (see Table 3.1) for the four most popular browsers (IE, Firefox, Opera, and Safari) when each retrieved `http://www.cnn.com/` at nearly the same time and from a host in the University of North Carolina campus network. The feature pictured in Figure 3.1 is the number of packets sent from the browser, accumulated over all flows that comprise the retrieval. It is evident that in these retrievals, Firefox initiates more flows than the other browsers, Opera sends more packets in earlier flows, and Safari sends fewer packets overall. Figure 3.2 shows the start time of each flow minus the most closely preceding flow start time, accumulated over all flows in the retrieval. This feature clearly shows that certain browsers (e.g., Firefox) try to improve response time by multiplexing the retrieval of content across substantially more flows than other browsers.

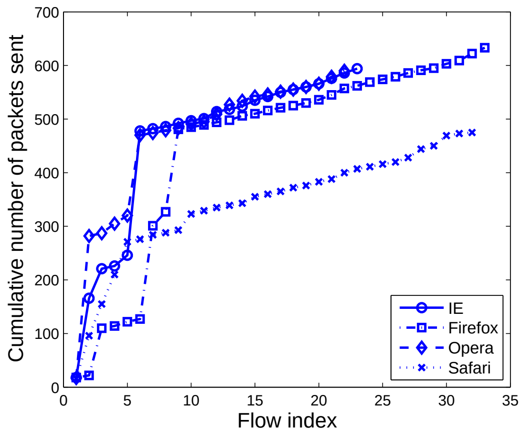


Figure 3.1: Number of packets sent from the browser, accumulated over all flows that comprise the retrieval. Each retrieval is to `www.cnn.com`.

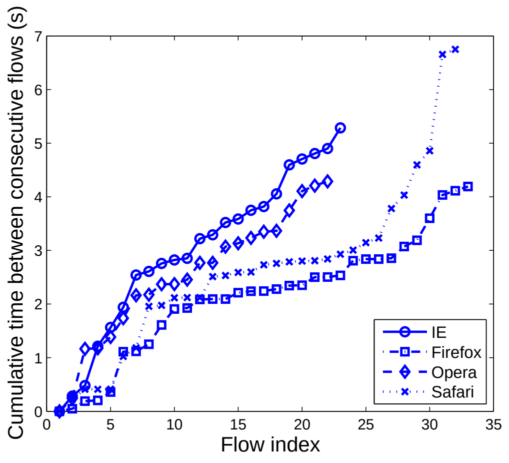


Figure 3.2: Cumulative time between the start of consecutive flows that comprise the retrieval. Each retrieval is to `www.cnn.com`.

However, using these differences to reliably determine the browser from flow records is not as straightforward as it may seem, and in particular is not as easy to automate as Figures 3.1–3.2 might suggest. Aside from the content and structure of the websites, users' browsing behavior, browser configuration, geographic location, and the client hardware configuration can also affect browser network behavior. As such, in the remainder of this section we test with what precision and recall an automatic classifier can distinguish among browsers in different scenarios.

More specifically, the classifier type that we utilize is Support Vector Machine (SVM) [31][7], which has been widely applied to many supervised learning problems (e.g., text classification [73], face recognition [106]). For classification problems with a small number of resulting classes, SVM usually performs better than other types of classifiers [160]. Given two sets of labeled data, SVM finds a hyperplane that separates the data and maximizes the distance to each data set. When multiple classes are involved, the SVM generates a group of pair-wise binary classifiers. Each binary classifier gives a vote to a class, and the final classification is the class with the highest vote. Loosely speaking, since an instance is classified depending on which side of the separating hyperplane it lies on, and not necessarily on how far from the hyperplane it is, there can be cases where an instance is misclassified if it is located "close" to the separating hyperplane.

To aid in our classification, we modify the aforementioned application of SVM to incorporate a notion of "confidence". The confidence threshold is the minimum distance of the hyperplane from the testing instance, where only instances with distance to the hyperplane greater than the confidence threshold are classified. This allows the classifier to avoid making decisions in ambiguous situations that would likely result in incorrect classifications.

The general structure of each test described below is that we first train a browser classifier on one dataset and then classify each retrieval in another dataset to obtain a guess of the browser used in that retrieval. Each website retrieval is classified only if its distance to the separating hyperplanes is greater than the confidence threshold. The browser used by host s is determined to be the browser classified most often in s's retrievals. To avoid errors due to a host having a small number of retrievals, we only consider hosts with more than thirty classified retrievals in our analysis. Our choice of thirty retrievals was determined empirically, and provides a good balance between precision and the number of hosts classified from the dataset.

We denote the classification for host s to be $\mathsf{browserguess}(s)$, and the actual browser used by host s to be $\mathsf{browser}(s)$. Note that $\mathsf{browser}(s) = \perp$ if the actual browser for s could not be determined, which occurred in the CMU dataset in some cases; see Section 3.2. Also, $\mathsf{browserguess}(s) = \perp$ can result if the classifier makes no classification for s, since no overwhelming choice arises for s's retrievals (e.g., all of

[7]We utilize the SVM implementation included in the Weka machine learning package [150].

the testing instances being close to the SVM hyperplane). The precision and recall across all hosts in the test dataset is defined as follows:

$$\begin{aligned}
\mathsf{Precision} &= \Pr[\mathsf{browser}(s) = b \mid \mathsf{browserguess}(s) = b \neq \bot] \\
&= \frac{|\{s : \mathsf{browserguess}(s) = \mathsf{browser}(s)\}|}{|\{s : \mathsf{browserguess}(s) \neq \bot\}|} \\
\mathsf{Recall} &= \Pr[\mathsf{browserguess}(s) = b \mid \mathsf{browser}(s) = b \neq \bot] \\
&= \frac{|\{s : \mathsf{browserguess}(s) = \mathsf{browser}(s)\}|}{|\{s : \mathsf{browser}(s) \neq \bot\}|}
\end{aligned}$$

A classifier that makes random guesses, i.e., classifying each host as a particular browser with $\frac{1}{n}$ probability, where n is the number of browsers, and a network where the browsers are distributed evenly among the hosts, the precision can only be expected to be $\frac{1}{n^2}$.

3.3.1 Tests on PlanetLab-QEMU Dataset

In an ideal web browsing scenario, only one website retrieval is taking place at any time, such that boundaries between consecutive retrievals are clearly delineated, and each web page is allowed to fully download before the next one. While this idealistic scenario will be compounded by many other issues in practice, we argue that tests in a controlled environment are valuable in that they enable us to better understand what factors influence classification the most.

We evaluated our browser classifier under this setting using the PlanetLab-QEMU dataset. To simulate multiple hosts, each running a specific browser implementation, data from each PlanetLab host was separated by the browser that generated the traffic. This traffic pertaining to a specific browser from one host served as testing data, while the classifier was trained on traffic from all other hosts, for each experiment. Since in some applications it will not be possible to obtain retrievals from every website that may be present in the testing data, we set the training data to be traffic from the top 100 websites, and used traffic from the remaining 50 websites (from top 100 to 150) for testing.

The precision and recall are shown in Figure 3.3, for confidence thresholds set to one of $\{$0.35, 0.65, 0.95, 1.15, 1.30, 1.50$\}$. The rise in precision with the increase in confidence threshold is likely due to incorrect classifications being filtered out, to the point that most of a host's classified retrievals are then correct. On the other hand, recall decreases with the confidence since more hosts are unclassified (i.e., $\{s : \mathsf{browserguess}(s) = \bot\}$). In all cases the correct browser can be identified with at least 71% precision and recall, and the precision grows to 100% with recall at 43% as the confidence threshold is increased.

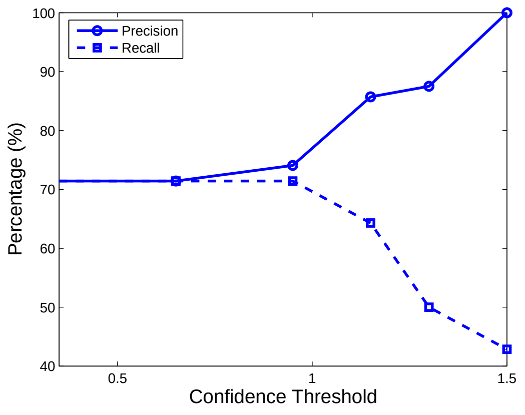


Figure 3.3: Precision and recall for browser classification on the PlanetLab-QEMU dataset.

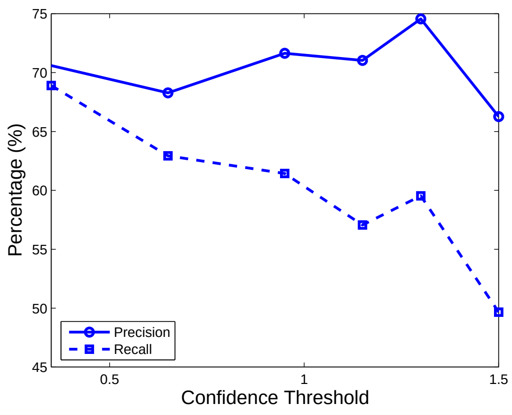


Figure 3.4: Precision and recall for browser classification on the CMU dataset (Train: PlanetLab-Native, Test: CMU).

These results show that browser implementations exhibit different traffic behaviors that can be identified even in coarse flow records.

3.3.2 Tests on CMU Dataset

Unlike the controlled setting of the PlanetLab experiments, the CMU dataset provides a setting for evaluating our techniques on traffic recorded in the real world. That said, we remind the reader that for purposes of ground truth, we could only reliably identify hosts using Firefox and Opera in the CMU dataset, and consequently, our analysis here is restricted to these cases. Out of those hosts, the vast majority of them used Firefox, and to not bias our results to that of a single-browser evaluation, we randomly select Firefox hosts in the CMU dataset but ensure that we have an equal number of Firefox and Opera hosts. The browser classifier is trained on the PlanetLab-Native dataset.

Figure 3.4 shows the precision and recall for the CMU dataset, for confidence thresholds set to one of $\{0.35, 0.65, 0.95, 1.15, 1.30, 1.50\}$. The precision generally increases slightly with the confidence threshold, as incorrectly classified instances were filtered out (because they were too close to the separating hyperplane), while recall decreases as a higher threshold leads to more unclassified instances (i.e., $\{s : \mathsf{browserguess}(s) = \perp\}$). As the confidence threshold increases, some hosts whose majority of retrievals were correctly classified now have those correct classifications filtered out, so that these hosts are left with more misclassified retrievals that cause the browser to be identified incorrectly; this results in a decrease in precision at the end of the curve. The peak in precision is 74.56%, when the confidence threshold is

1.30. We note that in this test (where the number of Firefox and Opera hosts are balanced) our precision is substantially greater than that of random guessing (i.e., 25%).

3.4 Incorporating Browser Fingerprinting into TĀMD

In this section, we consider the impact of reliable browser fingerprinting on TĀMD. Specifically, we modified its platform aggregation function ByPlatform so that a platform aggregate is identified when the largest fraction of hosts sharing the same O/S *or* the same web browser is above a given threshold. In doing so, we aim to detect both platform-dependent and browser-dependent malware, while incurring only slight overhead.

To quantify this overhead, we followed the same experiments that were performed for TĀMD in Section 2.4, which involved seven types of O/S-specific (but not browser-specific) malware. Recorded network traffic from the malware were overlaid onto the CMU dataset by assigning malware traffic to originate from randomly selected internal hosts. This combined data, consisting of the CMU dataset overlaid with malware traffic, was then given to TĀMD— configured to identify common host platforms based on their O/S or browsers, but otherwise configured identically as in Section 2.3 — in hourly batches, where the goal is to identify the single aggregate consisting of the malware traffic. The same experiment was repeated for each hour of traffic collected over three weeks in November and December 2007, for each of the seven different malware.

Figure 3.5 shows the number of browser aggregates identified by this new version of TĀMD in each malware experiment, in addition to the malware aggregate and other O/S aggregates, for different thresholds on the homogeneity of the platform aggregate. When the threshold is set to 90%, meaning that at least 90% of the hosts in the aggregate are required to share a common browser (which cannot be $\perp$), the number of additional aggregates reported due to browser similarity on average per hour is 0.0229. This shows that incorporating browser fingerprinting into TĀMD induces a limited amount of additional cost, while giving TĀMD the ability to detect a wider range of malware, i.e., both O/S-dependent and browser-dependent malware.

For network administrators, it is possible that a mapping of internal hosts to their browser implementations is available, or can be built by examining the payload of sampled packets. In these cases, we expect that TĀMD can be improved further, since errors from browser classification are eliminated. We thus believe that the results reported here for TĀMD augmented by our browser classifier should serve as

Malware Traces	Homogeneity threshold		
	70%	80%	90%
Bagle	0.25 (±4.95)	0.09 (±3.05)	0.09 (±3.05)
IRCbot	0.05 (±2.18)	0.01 (±0.99)	0.01 (±0.99)
Mybot	0.03 (±1.39)	0.00 (±0.00)	0.00 (±0.00)
SDbot	0.06 (±1.94)	0.00 (±0.00)	0.00 (±0.00)
Spybot	0.02 (±1.40)	0.00 (±0.00)	0.00 (±0.00)
HTTP bot	0.03 (±1.39)	0.00 (±0.00)	0.00 (±0.00)
Large IRC bot	0.19 (±3.05)	0.06 (±2.19)	0.06 (±2.19)

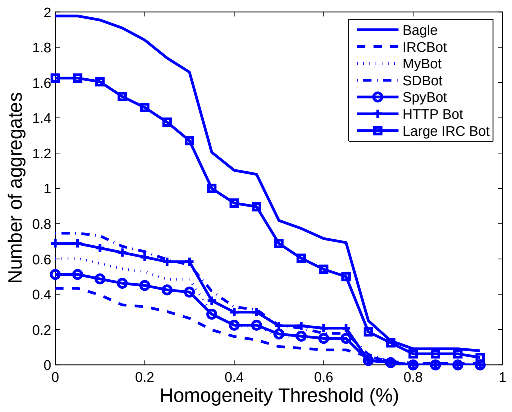


Figure 3.5: Mean number of aggregates per hour (± standard deviation) due to browser similarity, in addition to the identified malware cluster and to O/S aggregates.

a rough lower bound for what can be achieved in host infection classification.

In addition to enhancing the TĀMD system, applications that can potentially benefit from our passive browser fingerprinting technique include other intrusion detection systems that operate on flow records (e.g., [27, 48, 54, 80]), and approaches for profiling network traffic (e.g., [2, 78, 153]).

3.5 Chapter Summary

In this chapter, we develop a browser classification that passively identifies browser implementations from coarse flow records. We show that browser implementations can be identified with substantial precision and recall, even within flow records from real user traffic that is recorded at a different time and on a different network from the traffic used to train the classifier.

We also demonstrate how browser identification can be used to improve TĀMD, our network intrusion detection system described in Chapter 2, by permitting the system to identify aggregates of hosts on the network that share the same browser. Suspicious traffic is even more suspect when coming from such an aggregate, since this may indicate that these hosts have succumbed to a browser-specific exploit. Our browser fingerprinting techniques would enable TĀMD to detect more types of malware, i.e., those that are browser-dependent, while incurring slight overhead.

In addition to enhancing network intrusion detection systems, we also show that knowledge of the browser implementation can be used to improve the accuracy of traffic deanonymization. The details of this study can be found in Appendix A.

Chapter 4

Are Your Hosts Trading or Plotting?

Peer-to-peer (P2P) networks were used as botnet communication channels as early as 2003 [52]. The decentralized nature of these networks overcame the single-point-of-attack limitation in centralized control, making the botnet resilient to individual peer failures and also harder to detect and take down. These motivations for using P2P substrates are similar to those underlying the use of P2P protocols for file-sharing; the takedown of Napster, for example, highlighted the limitations of a centralized "command-and-control" infrastructure in that domain. It is thus not surprising that P2P substrates now commonly support both activities.

A consequence of this common use of P2P technologies is that infected hosts may not communicate with the same destinations, such that the ByDest^{τ} function T$\bar{\text{A}}$MD can fail. Moreover, botnet command-and-control traffic will tend to "blend into" a background of P2P file-sharing, making it difficult to separate these two types of traffic. In both types of P2P networks, status information about available peers needs to be maintained constantly to ensure the connectivity of the network; peers experience a high connection failure rate due to the dynamics of nodes joining and leaving (i.e., "churn"); and peers participate in client and server activities simultaneously. This commonality is punctuated by the fact that one highly publicized and well-studied P2P botnet, Storm, built its communication protocol based on the Overnet network, whose distributed hash table implementation [97] is incorporated in both eDonkey [1] and BitTorrent [2] file-sharing applications.

In light of this, the primary problem facing the detection of such bots is differentiating them from other P2P hosts. We focus specifically on the problem of P2P botnet detection given this challenge, and

[1]`http://wiki.amule.org/index.php/FAQ_eD2k-Kademlia`

[2]`http://bittorrent.org/beps/bep_0005.html`

construct a series of tests on network traffic to separate P2P bots from P2P file-sharing hosts, to which we will refer as Plotters and Traders, respectively. Our tests work exclusively on traffic summaries (e.g., flow records) with no access to individual packets (much less payloads), and so can scale to very busy networks where per-packet logging may not be cost-effective. Our technique is thus also unaffected by encryption of bot payload contents.

Given the varied nature of malware behaviors, we focus on characteristics of the traffic that do not depend on particular attack activities performed by the infected hosts (e.g., spam forwarding, denial-of-service attacks), but rather that are basic properties of Plotters that operate over P2P networks. At a high level, these characteristics include:

- **Volume**: Since Traders generally perform large multi-media file transfers (e.g., MP3, movies), but Plotters almost never do, traffic volume should be a good indicator of suspicious activity. However, as we will show, examining volume alone yields many false positives.

- **Peer churn**: The peer membership of a file-sharing network is very dynamic, due to peers constantly joining and leaving the network, the availability of the desired file, and connections between hosts being terminated soon after the completion of the file transfer. Previous studies [58, 126, 140] also showed that most Traders appear only once a day, and remain connected for short durations (minutes). Plotters, by contrast, are likely to experience less churn in peer membership due to several reasons. First, without a centralized command-and-control server, Plotters are required to maintain connectivity to their peers in order to receive and execute commands from the bot-master. Second, the Plotter cannot control when network access will be available, and so it is often opportunistic in communicating with peers, i.e., whenever it has a chance. Lastly, each Plotter maintains a list of known peers with which to communicate, such that they tend to contact the same hosts repeatedly.

- **Human-driven versus Machine-driven**: Perhaps a more basic difference between Plotters and Traders is that, while file-sharing activities are mainly human-driven, Plotters are almost entirely automated. This causes much of their traffic to exhibit temporal similarity that is rarely seen among those from human activities. Previous studies on distinguishing humans and bots in Internet chat rooms also observed that human behaviors are more complex than bots [49].

We construct measures of each of these characteristics, framing them into tests that distinguish Plotters from Traders. To our knowledge, our work is the first to target Plotters from the perspective of their commonality (or the lack thereof) with other P2P protocols.

We evaluate the ability of our technique to identify Plotters within traffic observed at the Carnegie Mellon University campus network. Our results show that Storm bots can be identified with up to 87.50% true positive rate and only 0.47% false positives, despite the fact that Traders using the *same* P2P substrate were present in our tests. We also perform tests with Nugache bots, where we show that for a false positive rate of 0.57%, we can detect 34.80% of the bots. We will explore the reasons behind our lower — though still substantial — detection rate in this case.

A final contribution of this chapter is to examine how much malware behavior would need to change to evade our technique. We quantify for each of our component tests the degree to which Plotters would need to alter their behaviors to evade them. The results suggest that evading our technique would require significant behavioral changes of existing botnets. Moreover, due to the way in which our tests are constructed, it would typically not be evident to the Plotters how much change would be sufficient to evade them.

4.1 Related Work

Much work on Plotters focused on understanding how such botnets operate, including Storm [52, 66, 116, 138], Nugache [138], Waledac [17, 132, 136], and Conficker [117]. Early work on disrupting Plotters (targeting Storm, in particular) injected a large number of fake nodes into the network to perform various Sybil attacks [41, 60, 66], such as content poisoning or eclipsing certain nodes from the rest of the P2P network. These studies show that the effectiveness of the attack depends on the attack duration as well as the number of Sybils. Kang et al. [77] developed a P2P monitor that infiltrated the Storm botnet to identify the IP addresses of infected hosts. Their monitor was able to detect bots behind firewalls or NAT devices, achieving a broader coverage than others that actively crawl the network.

Behavior-based detection techniques, described in Chapter 1.1, also target Plotters by examining correlated characteristics of network traffic. This includes identifying hosts performing suspicious activities (e.g., scanning, spamming) and sharing common communication contents [55], or exhibiting similar traffic statistics and suspicious activities [54]. However, these approaches may be evaded by changes in malware behavior, many of which have already taken place, such as turning to social engineering as an infection vector instead of scanning, or using encryption to make payload analysis difficult. Still others (e.g. [20]) use behavioral analysis to identify P2P-bot behaviors exhibited over non-P2P protocols.

In contrast to previous work, we focus specifically on distinguishing Plotters, whose command-and-control channel is implemented in a P2P fashion, from Traders. We do so by observing network-level characteristics inherent to P2P applications, but that are able to distinguish Plotters from Traders due to the different goals and circumstances behind how they utilize the P2P protocol. For example, Plotters communicate over P2P networks mainly for subtlety and resilience, instead of large file exchanges. They are also incentivized to maintain persistent connections to other peers in the network, in contrast to Traders, who have been observed to go offline after the completion of file transfers [58].

Jelasity et al. [72] studied techniques that can be deployed by Plotters to evade P2P traffic detection. However, they only consider the case where traffic dispersion graphs (TDGs) [70] are used to identify P2P traffic. The TDG-approach assumes a global view of the network, constructing a communication graph between all nodes to check if the average degree and the fraction of nodes with both incoming and outgoing connections are above a threshold. To evade such detection, the authors specifically focused on reducing the number of peers each Plotter contacts, such that most of the botnet's traffic are routed through a few fixed nodes. While this approach may limit the number of detectable Plotters using TDGs, its impact on other methods for identifying P2P traffic (that do not require the communication graph) is not evaluated.

One of the characteristics explored in this work is the difference between human-driven and machine-driven traffic. This observation has also been applied in other contexts, including cheat detection in online games [129], distributed denial-of-service attack defenses [59, 75], and chat bot detection in Internet chat rooms [49]. While most approaches to identifying automated traffic were host-based (e.g., deploying trusted software components on the client host), Gianvecchio et al. [49] found that the network traffic from human activities shows a higher entropy than those from bots for the case of Internet chat room traffic. Giroire et al. [50] proposed a method to detect centralized botnet command-and-control traffic by monitoring persistent and regular connections made to the same group of destination IP addresses, i.e., the command-and-control server. Since legitimate user traffic can also appear to be persistent and regular, this approach requires whitelisting common sites users visit, and is not suitable for detecting Plotters that communicate over P2P networks. BotProbe [56] exploits the determinism in the bots' pre-programmed behavior in response to commands from the bot-master. It actively replays, modifies, and intercepts packets of suspicious connections, and utilizes hypothesis testing to distinguish bot traffic from those of human users. However, this approach is hindered by encryption, and may require human participation.

4.2 Data Collection

As in the TĀMD system (see Chapter 2), we assume the role of a network administrator that aims to identify Plotters internal to her network by observing only traffic crossing the border of the network. The network traffic utilized in our analysis was organized into bi-directional Argus [3] flow records. In addition to the source and destination IP addresses and ports, the protocol, the start and end times of the flow, and the packet and byte counts, each Argus record also includes the first 64 bytes of the payload on the connection. In contrast to TĀMD, we do not use the payload in our technique. Rather, the payload information is used solely for determining ground truth, that is, determining whether the host is a Plotter or a Trader.

We use the following datasets in our analysis:

CMU dataset. Similar to that used in the evaluation of TĀMD in Section 2.4, this dataset consists of anonymized traffic from the edge routers of the Carnegie Mellon University campus network, which has two /16 subnets. The rate of this traffic is about 5,000 flows per second, and was collected from 9 AM to 3 PM over eight days in November 2007. We focus on only TCP and UDP traffic in this dataset.

Trader dataset. We identified those hosts in the CMU dataset that participated in known P2P file-sharing networks (i.e., the Traders), using the 64 bytes of payload in each flow record available to us. Specifically, we focus on three popular file-sharing applications: Gnutella, eMule, and BitTorrent. Hosts running Gnutella were identified by the protocol keywords “GNUTELLA”, “CONNECT BACK”, and “LIME” in their payload. [4] eMule hosts were identified by the initial byte ‘0xe3’ or ‘0xc5’, followed by various byte sequences as specified in the protocol specification [88]. BitTorrent hosts were identified by the protocol keyword “BitTorrent protocol”, web requests to trackers beginning with “GET /scrape” or “GET /announce”, and distributed hash table control messages with the substrings “d1:ad2:id20” or “d1:rd2:id20”. [5]

Plotter dataset. We also obtained Plotter traffic traces gathered from honeynets running in the wild in late 2007 [54]. These include a 24-hour trace of Storm, which contains traffic from 13 bots, and a 24-hour trace of Nugache, which contains traffic from 82 bots. Spamming and scanning activities were blocked

[3] `http://www.qosient.com/argus`
[4] `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`
[5] `http://wiki.theory.org/BitTorrentSpecification`

during the collection of these traces, and so the remaining traffic consists mostly of botnet control traffic, e.g., for peer discovery. As we show in this chapter, these traces were used in our evaluation, where they were overlaid onto the CMU traffic by assigning them to randomly selected internal hosts that were active in the CMU dataset. This method of overlaying malware traffic onto the CMU dataset is similar to that performed in Chapter 2 for the TĀMD system, which was evaluated using traces from centralized bots.

4.3 Methodology

Given network traffic observed at the border of an enterprise network, our goal is to identify internal hosts that are Plotters, where the main challenge in doing so is to distinguish them from Traders. We construct a set of tests that quantify the characteristics volume, peer churn, and human-driven versus machine-driven, which aim to take advantage of the different goals and circumstances behind how Plotters and Traders utilize P2P networks. Each test takes as input a collection of traffic, Λ, which involves a group S of internal hosts over one day, and outputs a subset of hosts in S that exhibit characteristics for which the test evaluates. In the following, we detail the rationale behind each of the characteristics, how they can be useful indicators for distinguishing Plotters from Traders in particular, and the construction of the corresponding test functions. We then describe how multiple tests can be combined to refine the results to narrow in on Plotters within the local network.

4.3.1 Volume

The first distinguishing characteristic we consider between Plotters and Traders is the amount of traffic each host contributes to the network. A common purpose of Traders is to exchange data, and much of the data found on popular P2P file-sharing applications are large multi-media files (e.g., several MBytes in size [126]). By contrast, the use of P2P architectures by Plotters is not so much for the sharing of information as for resilience and subtlety. Their traffic hence tends to be much lower in volume. In fact, the Storm botnet was observed to use the P2P protocol only for exchanging control messages, while file transfers were performed over HTTP [52, 138].

We examine traffic volume for a host in terms of the average number of bytes per flow that it contributes to the network (i.e., uploaded by the host). Compared to the cumulative byte count, this metric is less likely to be biased by the number of flows generated by a host, since a Plotter that is chatty can accumulate a large byte count over a short time window, while each individual flow is quite light-weight.

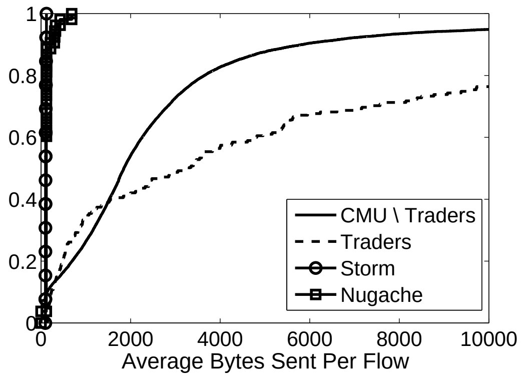


Figure 4.1: Cumulative distribution of the average flow size per host in each dataset over one day.

Figure 4.1 shows the cumulative distribution of this value per host, plotted from a single day of traffic from the CMU dataset, the Trader dataset, and the Plotter traces. This figure shows that the amount of data contributed by the Plotters (i.e., the Storm and Nugache bots) is significantly smaller than Traders.

Tests on Volume By quantifying a host's traffic volume using the average number of bytes sent per flow, we can define a test function θ_{vol} that uses this characteristic to distinguish between Traders and Plotters. The function takes as input a collection of traffic, Λ, which involves a group S of internal hosts over one day, and a threshold τ_{vol}. Hosts whose average flow size is less than τ_{vol} are returned in the set $\mathsf{S}_{\mathsf{vol}}$.

In practice, τ_{vol} can be set dynamically depending on the current traffic makeup, for example, as the median value observed across all hosts in S. This can make it more difficult for a Plotter to masquerade itself as a Trader; e.g., the amount of data it sends per flow needs to be larger than the majority of the hosts in the local network, though the Plotter would presumably be unaware of that amount that it must exceed.

4.3.2 Peer Churn

Peer churn refers to the dynamics of peers joining and leaving the network, and is a common phenomenon among both Traders and Plotters. This characteristic is often reflected in the high ratio of failed connections observed in P2P networks [9, 26]. Previous studies on P2P file-sharing networks have shown that peers are often connected for only short durations (a few minutes on average) [58, 126, 140], and many of them leave the network permanently after requesting a single file [58].

We hypothesize that even though the dynamism in peer membership is present in both systems, peer churn is less significant among Plotters than among Traders. This is because Plotters have motivation to keep up persistent communications with each other and maintain the connectivity of the botnet, since the bot-master needs to be able to control her bots. The Plotter also cannot control when network access will be available on the infected machine, and so it is often opportunistic in initiating communications, i.e., whenever it has a chance, making a Plotter's network activities more persistent in doing so. In addition, most Plotters store a list of known peers with which it maintains communications, both for bootstrapping itself into the network [17, 52, 66, 116, 138] and to limit the number of active connections. Such behaviors make it more likely for Plotters to contact the same hosts than Traders, whose sets of peers are mainly determined by file availability.

This observation allows us to characterize peer churn using the set membership of the destination IPs that a host contacts. We quantify this by the fraction of new IP addresses that a host contacts in one day, or more specifically, the ratio of (i) the number of IP addresses that a host first contacts after its first hour of activity on that day, and (ii) the total number of IP addresses it contacts in that day. A higher percentage of new contacts indicates a higher amount of churn. Figure 4.2 shows the percentage of new addresses contacted by Plotters and Traders (in one-day's worth of traffic from the Plotter and Trader datasets). Most Nugache Plotters do not contact any more new IPs after their first hour of activity, while around 60% of the IPs contacted by Storm Plotters were new. By contrast, the majority of Traders contact more than 85% new destinations.

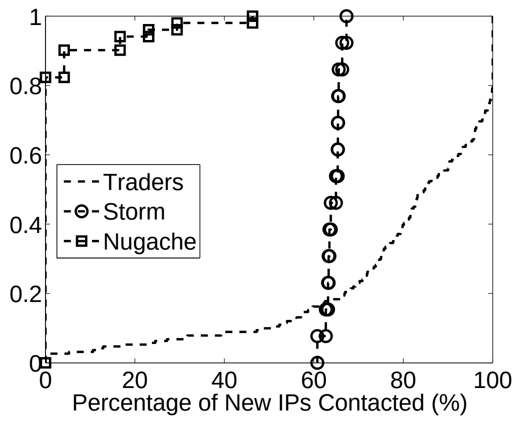


Figure 4.2: Cumulative distribution of the percentage of new IPs contacted by Traders and Plotters over one day.

Tests on Peer Churn Similar to the test for volume, we also distinguish Plotters from Traders using churn by performing a coarse separation between the two sets of hosts. The test function for peer churn, θ_{churn}, identifies hosts that have a relatively "low" churn (which are likely Plotters) using a threshold

τ_{churn}. By taking as input a collection of traffic Λ involving hosts S and a threshold τ_{churn}, the peer churn test $\theta_{\mathsf{churn}}(\Lambda, \mathsf{S}, \tau_{\mathsf{churn}})$ outputs a set $\mathsf{S}_{\mathsf{churn}}$ of hosts that contact a percentage of new IP addresses less than τ_{churn}.

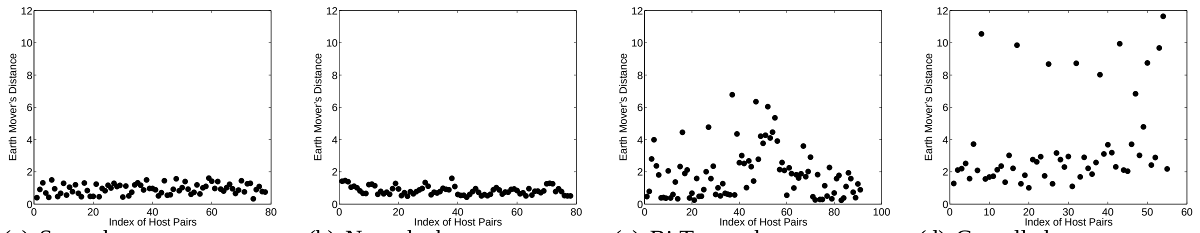
In practice, a Plotter could attempt to evade detection by increasing the fraction of new hosts it contacts, for example, by performing random scanning or initiating connections to different peers on its peer list at every communication attempt. This approach is risky, since it could make the Plotter detectable via other means (e.g., by identifying scanning activities) and reduces the stealthiness of the Plotter. We discuss evasion techniques that can be carried out by Plotters and quantify their induced costs in Section 4.5.

4.3.3 Human-driven vs. Machine-driven

Several works on botnet detection have studied the difference between human and machine-driven activities [54, 55, 56, 94, 95, 119], particularly focusing on the automated and synchronized nature of machine-driven behaviors. Only a few of these previous works have applied their technique to detecting P2P Plotters [54, 55]. However, these approaches rely on the presence of specific attack activities performed by the infected hosts.

We approach this problem by directly using timing-related information to characterize the similarity of machine-driven activities, such as periodic keep-alive/status messages exchanged between peers or scheduled checks performed by the Plotters to download new commands. Specifically, for each host, we examine the interstitial time distribution of its "activities" to the same destination IP, where an "activity" is a group of flows that overlap in time, such as multiple connections that are initiated in parallel. This distribution is observed across all destinations contacted by the host, since we do not know which ones are P2P peers. Since Plotters in the same botnet are likely to run similar versions of the bot binary, the timers used in triggering their activities should also follow the same algorithm. Hence the per-destination interstitial time distributions for Plotters should not only stand out from those of Traders, whose activities lack the regularity seen in automated traffic, but also appear "similar" to each other.

Tests on Human-driven vs. Machine-driven To compare the per-destination interstitial time distribution between hosts, we define a function, θ_{hm}, that uses a non-parametric approach to construct a histogram that approximates the underlying distribution for each host [47]. The Earth Mover's Distance [125] is then applied as the distance metric for comparing distributions. This allows us to identify clusters of hosts who exhibit similar timing patterns in their network traffic, where hosts whose traffic are mainly machine-



(a) Storm bots (b) Nugache bots (c) BitTorrent hosts (d) Gnutella hosts

Figure 4.3: Earth Mover's Distance between pairs of hosts within one day's worth of traffic. σ_{hm} is set to 2%.

driven, e.g., Plotters, should have different interstitial time distributions from hosts that are human-driven, e.g., Traders, and thus fall within separate clusters.

- **Constructing Histograms**. Given a collection of the observed interstitial time samples $\mathsf{v}(s)$ for a host s, we approximate its underlying distribution by constructing a histogram. The choice of histogram bin width is critical in this approximation, since a large value leads to over-smoothing, and a small value increases the sampling error. Moreover, applying a fixed bin width makes it straightforward for a Plotter to manipulate its traffic to evade detection.

  In this work, we follow a method proposed by Freedman et al. [47] to identify the optimal bin width, where the goal is to minimize the mean-squared error between the true distribution and the histogram. They show that the bin width can be computed as a function of the sample size $|\mathsf{v}(s)|$ (i.e., the number of observed interstitial time values for host s) and the "spread" of the samples, as represented by the inter-quartile range of the sample values, $IQR(\mathsf{v}(s))$. Specifically, the bin width for host s, b_s, is calculated by $b_s = 2 \times IQR(\mathsf{v}(s)) \times |\mathsf{v}(s)|^{-1/3}$.

- **Clustering Histograms**. One of the metrics for comparing distributions is the Earth Mover's Distance (EMD) [125]. Briefly, EMD is defined as the amount of work that is required to change one distribution into the other by moving "distribution mass" around. It is based on the transportation problem [37], where the challenge is to find routes that will minimize the cost of shipping goods from a group of suppliers I to a group of consumers J. That is, find a set of routes f_{ij} to minimize $\sum_{i \in I} \sum_{j \in J} c_{ij} f_{ij}$, where c_{ij} is the cost of shipping from supplier i to consumer j. By defining c_{ij} as the distance between the i^{th} and j^{th} bins in the histograms, the "distribution masses" are preferably moved between nearby bins. In this way, EMD is especially useful when the distributions are shifts of each other, but otherwise identical.

To find hosts whose network traffic exhibit similar timing patterns, we perform clustering on the histograms using an agglomerative hierarchical algorithm. In each step, we merge the two existing clusters for which the distance between host histograms, averaged over all ways of drawing one host from the first cluster and one from the second, is minimized (*average linkage clustering* [81]). This iterative process constructs a hierarchical clustering tree with the weight of each link being the distance (as described above) between the two existing clusters it connects. The final set of clusters is formed by cutting the top σ_{hm} percentage of links with the largest weights.

Figures 4.3(a) and 4.3(b) show the Earth Mover's Distance among pairs of Storm and Nugache bots [6] from our Plotter traces, when σ_{hm} is set to 2%. Compared to pairs of Traders, as shown in Figures 4.3(c) and 4.3(d), the Plotters have much "closer" distributions.

In addition to σ_{hm}, θ_{hm} also takes as input a threshold parameter, τ_{hm}; θ_{hm} filters out clusters whose diameters exceed τ_{hm}. Similar to the two previous tests, τ_{hm} can be set dynamically as a function of the diameters across all clusters. The output from the human-driven versus machine-driven test, $\theta_{\mathsf{hm}}(\Lambda, \mathsf{S}, \tau_{\mathsf{hm}}, \sigma_{\mathsf{hm}})$, is the union of the host clusters not filtered out in this way.

4.3.4 Combining the Tests

Each of the above tests, θ_{vol}, θ_{churn}, and θ_{hm}, aims to find Plotters using behavioral characteristics of a host's network traffic. Alone, each test may be too coarse to be effective at identifying Plotters. In Section 4.4, though, we show that when used in combination, they can narrow in on the Plotters, while largely eliminating other hosts.

Specifically, we combine the tests into an algorithm, $\mathsf{FindPlotters}$, shown in Figure 4.4. The algorithm takes as input a collection of traffic Λ involving a set of hosts S observed over one day, and outputs hosts who pass our various tests, i.e., that are likely to be Plotters.

4.4 Evaluation

We present an evaluation of the tests described in Section 4.3, using traffic from Plotters overlaid onto flow records recorded at the edge of the CMU network (the CMU dataset). For each day of traffic in the CMU dataset, we overlay the bot traces by assigning them to randomly selected internal hosts that are

[6]Specifically, here we used the top 25% Nugache bots in terms of the number of flows they generate. We will return to this in Section 4.4.2

```
FindPlotters($\Lambda$, S)
200: $\mathsf{S}_{\mathsf{vol}} \leftarrow \theta_{\mathsf{vol}}(\Lambda, \mathsf{S}, \tau_{\mathsf{vol}})$        /* Returns hosts with low traffic volume */
201: $\mathsf{S}_{\mathsf{churn}} \leftarrow \theta_{\mathsf{churn}}(\Lambda, \mathsf{S}, \tau_{\mathsf{churn}})$        /* Returns hosts with low peer churn */
202: $\mathsf{S}_{\mathsf{hm}} \leftarrow \theta_{\mathsf{hm}}(\Lambda, \mathsf{S}_{\mathsf{vol}} \cup \mathsf{S}_{\mathsf{churn}}, \tau_{\mathsf{hm}}, \sigma_{\mathsf{hm}})$
                                   /* Returns hosts with similar timing patterns in their traffic */
203: return $\mathsf{S}_{\mathsf{hm}}$
```

Figure 4.4: The algorithm used to find suspected Plotters by combining the tests on volume (line 200), peer churn (line 201), and human-driven versus machine-driven traffic (line 202), described in Section 4.3.1, 4.3.2, and 4.3.3.

active during that day (including possibly Traders). This makes our testing scenario more realistic, since those hosts still exhibit their normal behaviors, in addition to Plotter activities.

4.4.1 Initial Data Reduction

To serve as an initial data reduction step in our analysis, we first deploy a simple method to filter out hosts that are unlikely to be running P2P applications at all, by considering only hosts that have relatively high failed connection rates. The failed connection rate has been utilized in previous works that identify P2P traffic (e.g., [9, 26]), and here we use it simply as a coarse data-reduction step for eliminating hosts that are likely not running P2P applications at all, i.e., that are neither a Trader nor a Plotter.

Figure 4.5 shows the cumulative distribution of the percentage of failed connections per host, plotted from a single day of traffic from the CMU dataset, the Trader dataset, and the Plotter traces. Only hosts that initiated successful connections within that day were included. There is a clear distinction between the curves for the CMU$\setminus$Trader and Trader datasets, pointing out that P2P hosts do exhibit significantly higher failed connection rates compared to non-P2P hosts. A closer examination of the Traders with a small percentage of failed connections (e.g., less than 10%) revealed that they are BitTorrent hosts downloading Torrent files from trackers over HTTP, but that are not otherwise involved in P2P file-sharing activities.

Surprisingly, the Plotter traces also exhibit very different failed connection rates. In particular, many of the peer discovery messages sent by Nugache Plotters in our trace were unsuccessful, because the remote peer was either not active or not responding. This causes all Nugache Plotters to have more than 65% failed connections. Note that the curves for Storm and Nugache in Figure 4.5 are generated from the Plotter traces *only*. When they are overlaid onto the CMU dataset (Section 4.4.2), the percentage of failed flows can be biased by the traffic from the CMU host to which we assigned the Plotter traces.

As a data-reduction step to filter out those hosts who are likely *not* involved in P2P activities—while retaining hosts that are in fact running P2P applications—we use the median value among hosts in the

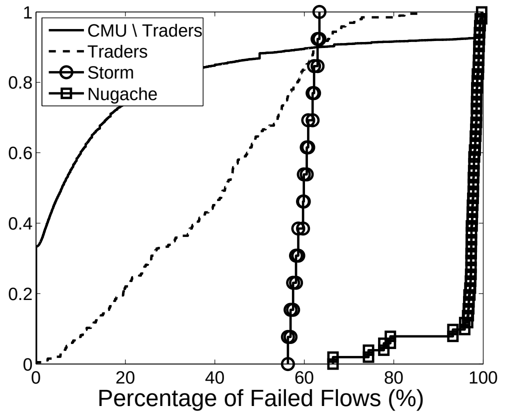


Figure 4.5: Cumulative distribution of the percentage of failed connections per host in each dataset over one day.

CMU dataset with Plotters overlaid (and that initiated successful flows) as the threshold for deciding which hosts to remove from consideration. This value is determined anew for each day of traffic. For example, for the case of Figure 4.5, the threshold for failed connection rate would be roughly 5% (i.e., 5.75%, the median value for the CMU dataset, then adjusted due to the overlaid Plotter data). Hosts with failed connection rates higher than the threshold are selected as "possibly P2P". This approach not only allows us to eliminate half of the hosts that are not likely to be Plotters, but is also more difficult for a Plotter to evade compared to setting a fixed threshold.

4.4.2 Identifying Plotters

We overlaid the Storm and Nugache Plotter traces onto each day of traffic in the CMU dataset by assigning them to originate from randomly selected internal hosts in the CMU campus network active on that day. This combined traffic is first passed through the initial data reduction step, and then given as input to the tests, where each returns a set of hosts that survived the test.

Figures 4.6, 4.7, 4.8 show ROC (Receiver Operating Characteristic) curves for the volume, churn, and human-driven vs. machine-driven tests. The input to the volume and churn tests is the set S of hosts that passed the initial data reduction step described in Section 4.4.1. The ROC curves are generated by setting the threshold τ_{vol} to be the 10, 30, 50, 70, or 90th percentile of the average bytes sent per flow per host, and τ_{churn} to be the 10, 30, 50, 70, or 90th percentile of the fraction of new IP addresses contacted per host. The input to the human-driven vs. machine-driven test, θ_{hm}, are those hosts that were retained by one of the volume or churn tests (i.e., $S_{\text{vol}} \cup S_{\text{churn}}$) with their respective thresholds set at the 50th percentiles (and by the initial data reduction step). To generate the ROC curve in Figure 4.8, the threshold τ_{hm} for θ_{hm} is set to be the 10, 30, 50, 70, or 90th percentile of the cluster diameters, and σ_{hm} is set to be 2%, 5%, or 10%. We emphasize that each ROC curve plots the true and false positive rates *relative to its input set* (i.e., S for θ_{vol} and θ_{churn}, and $\mathsf{S}_{\mathsf{vol}} \cup \mathsf{S}_{\mathsf{churn}}$ for θ_{hm}), as opposed to the overall CMU dataset with Plotters overlaid, in order to highlight the independent discriminating power of each test.

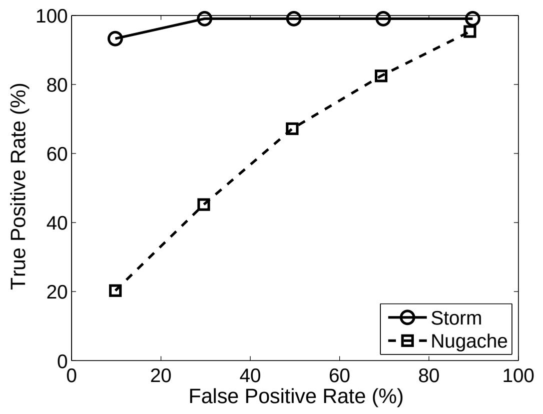


Figure 4.6: ROC curves for the volume test θ_{vol} when the Storm and Nugache traces are overlaid onto the CMU dataset, after filtering as in Section 4.4.1. Results are averaged over the eight days in the CMU dataset.

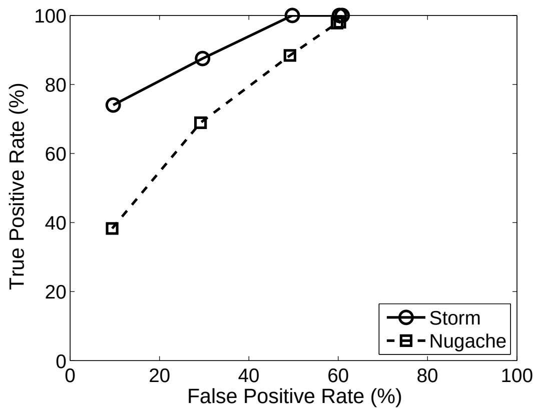
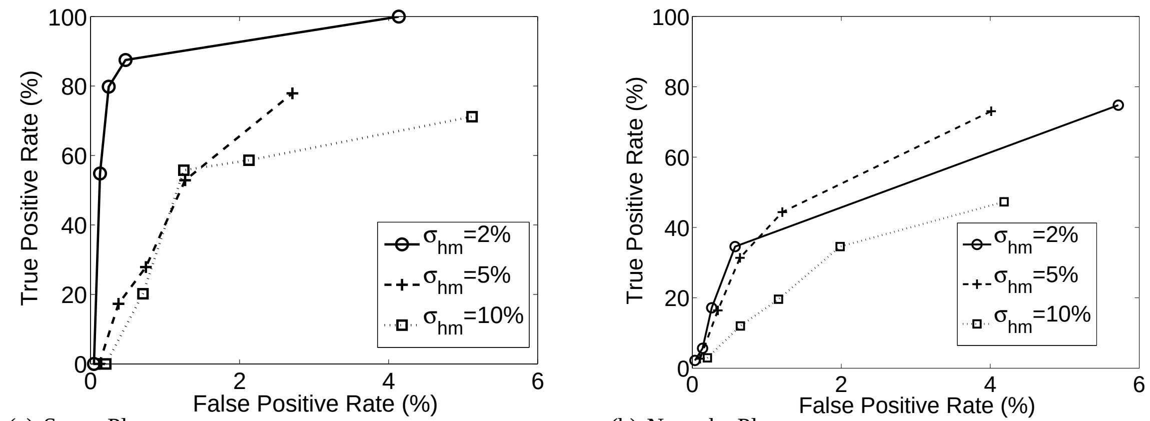


Figure 4.7: ROC curves for the churn test θ_{churn} when the Storm and Nugache traces are overlaid onto the CMU dataset, after filtering as in Section 4.4.1. Results are averaged over the eight days in the CMU dataset.



(a) Storm Plotters.

(b) Nugache Plotters.

Figure 4.8: ROC curves for the human-driven vs. machine-driven test θ_{hm} when the Storm and Nugache traces are overlaid onto hosts in the CMU dataset, after filtering as in Section 4.4.1 and by θ_{vol} and θ_{churn}. Results are averaged over the eight days in the CMU dataset.

Two observations from Figures 4.6–4.8 are evident. First, the true positive rates for Storm are higher than Nugache across all three tests, often reaching 100%. We will explore the reasons for this difference at the end of this section. The second observation is that alone, each of the tests would be too coarse to be effective at identifying Plotters, producing high false positive rates that can reach to 90% (e.g., the volume test).

In combination, however, they can be powerful at extracting Plotters from Trader-like hosts. To show this, we utilized the tests together as in the algorithm FindPlotters (Figure 4.4). To strike a balance between the true positive and false positive rates, we use the 50th percentile among the hosts as the threshold for both τ_{vol} and τ_{churn}, the 70th percentile of the cluster diameters for τ_{hm}, and 2% for σ_{hm}. Figure 4.9 shows how the results are refined at each step, where the maximum false positive rate (across tests for Storm and Nugache) is reduced to 0.57% (i.e., 0.47% and 0.57% for Storm and Nugache, respectively), while maintaining a true positive rate of 87.50% for Storm and 34.80% for Nugache. The percentage of Traders (from the Trader dataset) that remain after each test is also shown for comparison. The maximum percentage (across tests for Storm and Nugache) of remaining Traders is 5.47%, which comprises 13.14% of all the hosts returned by FindPlotters.

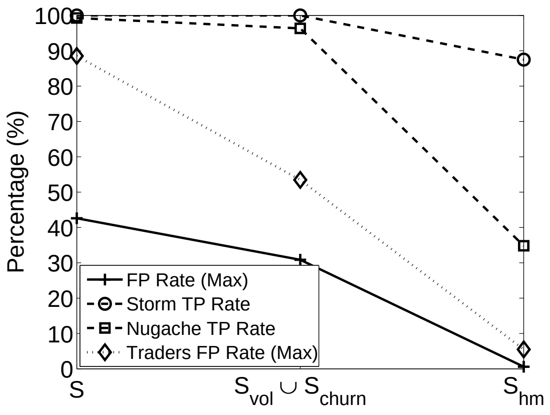


Figure 4.9: Results after applying the tests in sequence, averaged over eight days in the CMU dataset with overlaid Plotter traffic.

We took a closer look at those hosts that were identified as Plotters, but to which we did not assign malware traffic. Since our datasets were anonymized, we made use of the port numbers, protocol, and the 64 bytes of flow payload available to us. Among those identified hosts, many of them appeared to be running P2P-like applications. The same non-standard source port was used across communications with multiple IP addresses, and the host also received incoming connections on that port. These could be Traders that were online but not actively transferring files. Other hosts were found to be running instant messaging applications, such as Yahoo! Messenger or MSN, or mail clients that periodically contacted mail servers to check for incoming messages (e.g., via the POP3 or IMAP protocol). In practice, a network administrator can easily whitelist these applications to reduce the false positive rate even further.

We now return to the differences in detection rates between Nugache and Storm. As shown in Figure 4.9, most false negatives for Nugache resulted from θ_{hm}. Further investigation into these results showed that each test, but particularly θ_{hm}, tended to filter out less communicative Plotters, as shown in Figure 4.10. We have been unable to confirm a reason behind the large variance in the activity levels of the Nugache bots in our trace, though those who originally recorded the trace suggested that this may be due to the limited viability of the Nugache botnet at the time this trace was recorded.[7] A Plotter that is unable to connect to a given peer may attempt to contact several other Plotters before approaching the failed peer a second time, if it does so at all. Such uncertainties in the Plotter's state before successfully engaging in the botnet results in irregular behaviors that render our tests less effective, as shown in Figures 4.6–4.8.

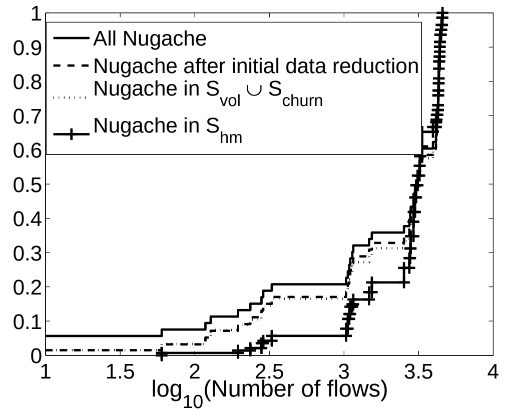
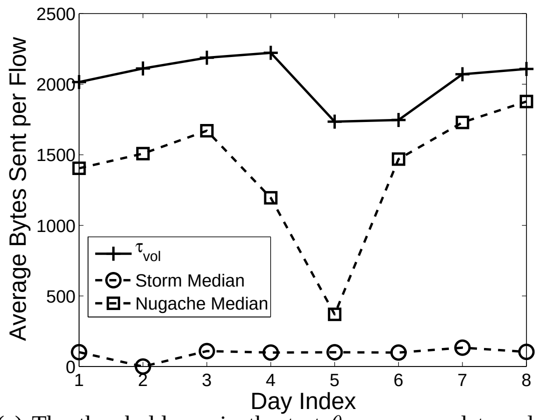


Figure 4.10: Cumulative fractions of the number of flows generated by the Nugache Plotters that remain after each test, in base-10 log scale. Results are accumulated over the eight days in the CMU dataset.

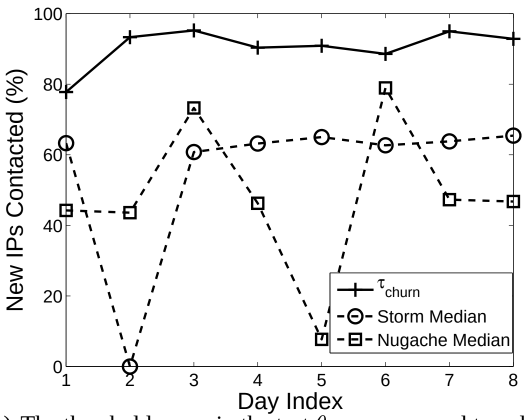
4.5 Evasion

A Plotter could attempt to change its network behaviors to evade our tests, e.g., by increasing its traffic volume so that it will escape the volume test. However, since the thresholds used in our tests are not fixed at set values, but instead are dependent on traffic statistics from *all* active hosts in the local network, a Plotter would have difficulty in determining the precise thresholds that will allow it to masquerade as a Trader.

[7]Guofei Gu, personal communication, October 2009.



(a) The threshold τ_{vol} in the test θ_{vol} compared to values observed from hosts with overlaid Plotter traffic.



(b) The threshold τ_{churn} in the test θ_{churn} compared to values observed from hosts with overlaid Plotter traffic.

Figure 4.11: Challenges for detected Plotters to evade θ_{vol} or θ_{churn}. Each of the eight days in the CMU dataset is shown.

Figures 4.11(a) and 4.11(b) show, for the volume test θ_{vol} and churn test θ_{churn} conducted on each day of traffic in the CMU dataset, the detection threshold used (i.e., the median among the hosts) versus the median value among the Plotters that were detected, once assigned to hosts. To evade the volume test, θ_{vol}, the median Storm Plotter would need to generate more than *20* times its original traffic volume per flow. The corresponding multiplicative factor for the median Nugache Plotter is roughly 1.3. To evade the churn test, θ_{churn}, a Plotter can either refrain from contacting hosts it had previously communicated with, or generate connections to a large number of new hosts it talks to only once. As an example of the latter case, a Plotter who wants to raise its percentage of new IPs from 60% to 90% (a typical value of τ_{churn}), while still maintaining the same number of hosts with which it communicates, would need to increase the fraction of new hosts it contacts by a factor of 1.5. Such evasion attempts from Plotters that increase their traffic volume or the number of new hosts (such as through random scanning) can compromise their stealthiness, making their presence in the network observable through other means (e.g., scan detection) or even by the owner of the infected machine.

The human-driven vs. machine-driven test, θ_{hm}, clusters hosts based on the distribution of their per-destination interstitial activity times, and identifies hosts that have similar timing patterns in their communications. Plotters belonging to the same botnet can avoid falling into the same cluster or increase the cluster diameter, for example, by having each Plotter select a different frequency at which to contact peers. This could affect our choice of bin width in histogram construction — which is dependent on both the number of interstitial time samples observed and the inter-quartile range of the samples (see

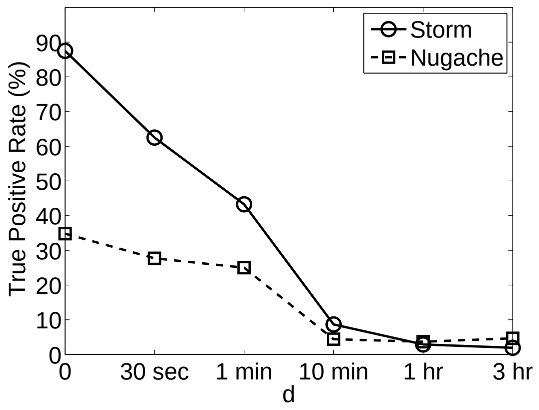


Figure 4.12: Challenges for Plotters to evade θ_{hm}. The y-axis is the true positive rate averaged over eight days of the CMU dataset with overlaid Plotter traffic.

Section 4.3.3) — and therefore alter the Earth Mover's Distance (EMD) between Plotters.

To quantify the operational cost for Plotters that want to evade θ_{hm}, we simulated Plotters who, instead of initiating communications at regular intervals, always add (or subtract) a random delay before each activity. By manipulating the distribution from which the interstitial times are drawn, the Plotters may disrupt our algorithm so that they no longer fall within the same cluster, or that the cluster diameter exceeds the threshold τ_{hm}.

We use the same Plotter traces that were used in the evaluation for this simulation, but add (or subtract) a random delay before every activity a Plotter initiates to a peer with which it had previously communicated. The delay is drawn from a uniform distribution over the interval $\pm d$, for each activity. Figure 4.12 shows the decay in the true positive rate as a function of d, ranging from 30 seconds to three hours. This suggests that Plotters must randomize their activities by minutes in order to evade detection via this test, potentially slowing the responsiveness of the botnet. Moreover, the per-destination interstitial activity time distribution of other machines in the local network also affects the needed value of d, which may be difficult for Plotters to measure.

4.6 Discussion

The experiments we reported above used network traffic collected in November 2007. The dominant protocols in this data (based on the port numbers and the 64 byte of flow payload available to us) include HTTP, DNS, HTTPS, SMTP, BitTorrent, and instant messaging (e.g., AOL, MSN). Today, there are a number of other popular peer-to-peer protocols for conducting activities other than file sharing, e.g., content distribution systems (e.g., SopCast [8], PPLive [9], Zattoo [10]) and Internet telephony (e.g., Skype [11]). Traffic to the default Skype port [12] accounted for less than 50 flows per day in our dataset, and so we believe that our false positives include a very small number of Skype hosts, if any. The prevalence of these other peer-to-peer systems in our data are even smaller.

That said, these newer protocols that gained popularity after our data collection raise the question of whether our techniques would be effective on today's networks. While we have not conducted a thorough analysis of this issue, it does appear that Skype, for example, would be at risk of being classified as a Plotter network due to its machine-like characteristics, particularly the exchange of periodic and low-volumed keep-alive messages between clients for maintaining connectivity [10]. The use of non-standard ports and encryption further makes it difficult to whitelist this application.

Certain features of Skype may enable a classifier to effectively distinguish it from Plotter networks, however. First, each Skype client is required to connect to a super node, which is a host running Skype that has high network bandwidth, high computing power, and a public IP address. The online duration of Skype super nodes has been observed to be quite high, e.g., over five hours on average [57], such that clients are likely to experience less peer churn than Plotters. Second, even though Skype clients do exchange keep-alive messages that are only a few bytes long [10], voice and video calls (the latter consisting of more than a quarter of all Skype-to-Skype calls [13]) will cause Skype clients to generate a higher volume of traffic than Plotters. If these or other features do not provide a sufficiently sound basis for separating Skype traffic from Plotter networks, then it may also be possible for administrators to identify Skype nodes by using known Skype instances to interact with them, in a way analogous to how administrators might try to infiltrate a botnet. Several works also proposed detecting Skype traffic by observing connection statistics (e.g., the packet arrival rate, packet size [16]) or by identifying relayed traffic [141].

4.7 Chapter Summary

In networks where P2P file-sharing is commonplace, a challenge in identifying bots managed via P2P infrastructures is the similarities that their network behaviors share with P2P file-sharing applications. In this chapter, we develop a series of tests for separating the two classes of P2P applications, and in

[8]`http://www.sopcast.org/`
[9]`http://pplive.en.softonic.com/`
[10]`http://zattoo.com/`
[11]`http://www.skype.com`
[12]According to `http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers`
[13]`http://blogs.skype.com/en/2008/04/interview_with_skype_ceo_josh.html`

particular for identifying bots within a network prior to their engaging in overt attacks. Our tests work on flow records, without access to individual packets. As such, our technique is scalable to busy networks where packet capture (or even packet header capture) is not cost-effective, and is also immune to bot payload encryption.

Using bot traces and traces of traffic collected at the edge of a university network, we show that our technique enabled the identification of Storm and Nugache bots with false positive rates of only 0.47% and 0.57% on average, respectively. At these false positive rates, we identified 87.50% of the implanted Storm bots, and 34.80% of the Nugache bots. Our lower detection rate for Nugache derives from the low and variable activity of the bots in our data (see Section 4.4.2), and so we believe this number to be conservative. We further evaluate the changes in bot behavior needed to evade our technique, and show that bots would need to increase their average flow size by roughly a factor of 1.3; increase the fraction of new IP addresses they contact by a factor of 1.5; or randomize their interstitial connection times significantly (e.g., ranging from minutes to hours). Moreover, the bots would need to accomplish this despite other traffic from the host it occupies, and since we defined our tests' thresholds relative to the background traffic, the behavior necessary to evade detection in any given network would typically be unknown to the attacker.

Since our tests focus on characteristics that describe differences in Plotter and Trader behavior, a limitation of this approach is in identifying Plotters that only affect Traders, e.g., a Plotter binary that spreads through P2P file-sharing networks. In this case, the Plotter traffic could be obscured by activities from the Trader, if the Trader is a heavy file-sharing user generating high volumes of traffic, for example. One method of distinguishing between Plotter and Trader traffic on a host might be to separate traffic by application, such as determined using port numbers. Traffic from each port, or a group of associated ports, can then be applied individually to the tests in Section 4.3. However, in our evaluations, the hosts to which we assigned bot traces were sometimes Traders, and were still effectively identified by the FindPlotters algorithm.

Chapter 5

Revisiting Botnet Models and Their Implications for Takedown Strategies

In this chapter, we turn to an analytical approach to study peer-to-peer botnets by applying graph models from network theory. Such models have been used in the study of many real-world networks, including social, biological, and computer networks [4, 46, 139]. For example, Erdös-Rényi random graphs [43] model networks where the edges are created with uniform probability between every pair of nodes. Watts-Strogatz small-world graphs [149] model networks where the diameter of the network is small, i.e., increasing logarithmically with the size of the network. Barabási-Albert scale-free graphs [5] model networks with a few highly connected "hub" nodes and many leaf nodes. These models allow the spread of information (or infection) [110, 149] to be analyzed in various network topologies, as well as their resilience to node and edge failures [3, 35, 63].

Recently, several works have also applied graph models from network theory to study decentralized peer-to-peer (P2P) botnets [36, 41, 90, 158]. Each node in the network represents an infected host, and edges reflect communications between the hosts. Properties of the graph can quantify the botnet's "usefulness". For instance, the diameter of the network measures the efficiency of bot communications, and the size of the largest connected component is the number of bots that are reachable by the attacker and can carry out her instructions. Assuming that P2P botnets are structured according to known models, these works aim to assess the effectiveness of strategies to take down a botnet. For example, one strategy that was found to be effective for some network topologies is to target nodes with high degree, i.e., that communicate with many hosts [36, 41, 158].

We observe that previous works applying graph models to P2P botnets do not consider an important property of networks — assortative mixing [102]. Assortativity refers to the tendency for a node to attach to other "similar" nodes, and is commonly examined in terms of a node's degree, i.e., high-degree nodes are likely to be neighbors of other high-degree nodes. This property is also referred to as *degree correlation*. The existence of this correlation between neighboring nodes has been observed in many real-world networks [102, 104, 111]. More importantly, it has been found to be a property of *growing* networks [19, 86], where the network increases in size as nodes join over time, as is true in a botnet as more hosts become infected.

We show that assortativity plays an important role in network structure, such that neglecting it can lead to an over-estimation of the effectiveness of botnet takedown strategies. By generating networks with varying levels of degree correlation, we demonstrate that a higher level of assortativity allows the network to be more resilient to certain takedown strategies, including those found to be effective by previous works. Moreover, we note that bots are dynamic entities that can react and adapt to changes in the network, and so the botnet can potentially "heal" itself after a fraction of its nodes are removed. We specifically explore cases where nodes can compensate for lost neighbors by creating edges to other nearby nodes, e.g., that are within h hops. Our simulations show that the graph can recover significantly after takedown attempts, even when h is small, and that higher levels of assortativity can allow the network to recover more effectively.

Another contribution in this chapter is in identifying alternative takedown strategies that are more effective than those explored by previous works. Specifically, we show that targeting nodes with both high degree and low clustering coefficient will decrease the connectivity and communication efficiency of the network significantly, and also makes it considerably more difficult for the network to recover from the takedown attempt.

5.1 Related Work

Botnet models Several previous works have studied botnets using network models. Cooke et al. [29] described three potential botnet topologies: centralized, P2P, and random, and discussed their design complexity, detectability, message latency, and survivability. Other works [36, 90] apply theoretical network models to botnets, including Erdös-Rényi random graphs [43], Watts-Strogatz small world graphs [149], and Barabási-Albert scale-free graphs [5]. This allows the effectiveness of takedown strategies to be quantitatively evaluated using graph properties, such as the network diameter, the average shortest distance between pairs of nodes, and the size of the largest connected component. Davis et al. [41] compared Overnet, which is utilized by the Storm botnet [66, 116], with random and scale-free networks to justify the choice of structured P2P networks made by bot-masters. They simulated takedown efforts on the networks by removing nodes at random, in descending order of node degree, or in a "tree-like" fashion by identifying nodes reachable from an initial node, and found Overnet to be more resilient than other graph models.

To our knowledge, no previous work on botnet modeling has considered the effect of *degree assortativity* in networks [102]. This property, defined as the correlation coefficient between the degrees of neighboring nodes, has been found to be high in many real-world social, biological, and computer networks [103, 111]. It has been studied analytically in the statistical physics literature, and found to be an inherent property of *growing* networks where nodes join and edges are created over time [19, 86], since older nodes are likely to have higher degree and connect to each other. Studies in this domain focus on understanding the underlying interactions between nodes that would result in a network that matches one empirically measured in the real world. By contrast, a network of bots is elusive and difficult to quantify in practice [120]. Making assumptions about the graph structure or node correlation (e.g., that there is none) is thus unfounded.

Network takedown strategies. The resilience of networks to attacks or failures have been explored in the physics branch of complex networks [3, 35, 63]. A scale-free network, which consists of a few highly-connected "hub" nodes and many "leaf" nodes, has been found to be particularly vulnerable to attacks where high-degree nodes are removed first. A takedown strategy that targets high-degree nodes is also recommended by previous works that studied botnet models [36, 41, 158], particularly for unstructured P2P networks where there are "super-peers" present.

Other types of takedown efforts on networks have also been explored in the complex networks literature, such as cascaded node removals [34, 148], removing nodes according to their betweenness centrality, or removing edges instead of nodes [63]. These works focus on the resilience of different network topologies, and do not take assortativity into account. Newman et al. [103] studied the prevalence of assortativity in real-world networks. Even though their focus is on measuring and generating assortative networks, they also showed, through simulation, that higher assortativity allows a network to have a larger connected component after a small fraction of high-degree nodes are removed. However, they did not explore other takedown strategies, the effect on other graph properties, or the network's ability to "heal" itself. In our work, we explicitly study the effect of assortativity on network resilience and the ability of dynamic networks (such as P2P botnets) to recover from takedown attempts.

5.2 Constructing and Measuring Assortative Networks

We first define degree assortativity, following the definition by Newman et al. [102], and perform an empirical analysis of the assortativity of a portion of the Storm botnet [66, 116, 138]. We then describe our algorithm for adjusting the level of assortativity in simulated networks, and the metrics we use to quantify the "usefulness" of a network. The metrics are aimed at capturing notions of communication efficiency between nodes and the number of reachable bots, which are likely to be of importance to the attacker.

5.2.1 Degree Assortativity

Degree assortativity, defined as the correlation coefficient between the degrees of neighboring nodes, measures the tendency for nodes to be connected to others who are "similar" in terms of their degree. For example, this property is especially significant in social networks, where gregarious people are likely to be friends with each other [71, 104]. It is also found to be a property of growing networks, where the network size increases as new nodes join and edges are created [19, 86].

We define assortativity following the definition of Newman et al. [102]. Let the fraction of nodes in a network graph with degree k be denoted p_k. If we choose an edge from the graph at random, and follow it to one of its ends, the probability that the node at which we arrive has a degree of k is proportional to k. This is because we are more likely to end up at a node with high degree, which has more edges connected to it. To account for the edge from which we arrived, the *remaining degree* of the node is its degree minus one. The probability q_k that we arrive at a node with remaining degree k is then

$$q_k = \frac{(k+1)p_{k+1}}{\sum_{j=0}^{\infty} j p_j} \tag{5.1}$$

Let $e_{j,k}$ be the probability that a randomly selected edge connects nodes of remaining degree j and k, where $\sum_{j,k} e_{j,k} = 1$. The assortativity γ of the network, being the correlation coefficient between the

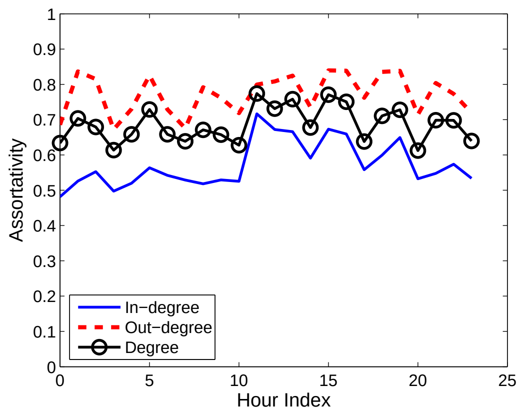


Figure 5.1: The assortativity for 13 Storm bots in a honeynet running in the wild.

degrees of neighboring nodes, is

$$\gamma = \frac{1}{\sigma_q^2} \sum_{j,k} jk(e_{j,k} - q_j q_k) \tag{5.2}$$

where σ_q^2 is the variance of the distribution of q_k, i.e., $\sigma_q^2 = \sum_k k^2 q_k - [\sum_k k q_k]^2$. A higher value of γ indicates that there is higher correlation between the degrees of two neighboring nodes. In a random graph, where every pair of nodes is connected with uniform probability, no correlation exists and $\gamma = 0$.

Even though high assortativity is found in many real-world networks, measuring it in practice can be challenging, for example, due to difficulties in observing all interactions between nodes in a large network. This is especially true for botnets. As an estimate of what the assortativity would be for a real botnet, we performed an empirical analysis by obtaining network traffic from a honeynet running in the wild in late 2007 [54]. This consists of a consecutive 24-hour trace from 13 hosts participating in the Storm botnet, whose peer-to-peer communications have been studied extensively in previous works [66, 116, 138].

Figure 5.1 shows the assortativity measured among the 13 Storm bots, where snapshots of their communications are taken on an hourly basis. The "degree" of a bot is represented by 1) the number of distinct source IP addresses from which it receives packets (the in-degree), 2) the number of distinct destination IPs to which it sends packets (the out-degree), or 3) the total number of distinct IPs with which it interacts. Since the rest of the Storm botnet is not directly observable, we calculated the assortativity from traffic between only the 13 Storm bots. However, this value is still quite high, ranging from 0.48 to 0.84, while that of social networks is only around 0.3 [103]. This suggests that a botnet may be significantly assortative, and highlights the importance of this property in considering botnet network models.

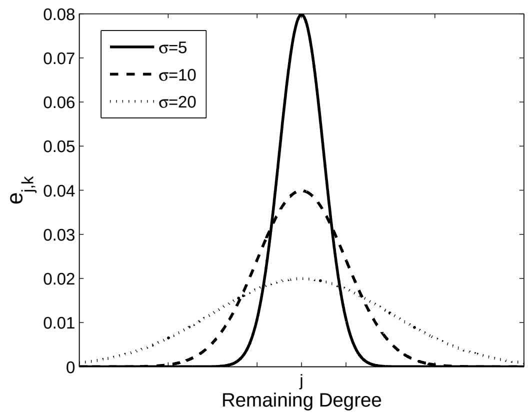


Figure 5.2: Edge probabilities as a normal distribution centered at j with different values for the standard deviation σ.

5.2.2 Generating Assortative Networks

To study the effect of assortativity on networks, we need to be able to generate networks with varying levels of assortativity. One method for this is to rewire edges in a given network [154]: At each step, select two edges at random, and shuffle them so that the two nodes with larger remaining degrees are connected, and the two nodes with smaller remaining degrees are connected. Repeating this step will result in the network becoming increasingly assortative. However, rewiring causes the shortest path length between nodes to increase rapidly [154], which may bias the comparison between networks with different levels of assortativity.

We apply another method for constructing assortative networks, similar to that described by Newman et al. [103]. This method takes as input the number of nodes in the network, the desired degree distribution p_k, and the edge probabilities $e_{j,k}$. Each node in the network is assigned a degree drawn from p_k. The remaining degree distribution q_k can then be calculated from p_k, and edges are added by connecting each pair of nodes of remaining degree j and k with probability $e_{j,k}$.

To control the level of assortativity in the resulting network, we specify $e_{j,k}$ as follows. For a fixed value j, assume that $e_{j,k}$ follows a normal distribution centered at j, where the standard deviation σ is the adjustable knob for tuning the level of assortativity. Figure 5.2 illustrates $e_{j,k}$ centered at j. A smaller σ causes the normal distribution to become more peaked, where nodes with remaining degree j have a higher probability of sharing edges with other nodes of remaining degree close to j, resulting in a more assortative network.

In our simulations, p_k is chosen so that the resulting network is scale-free, specifically, $p_k \sim k^{-3}$.

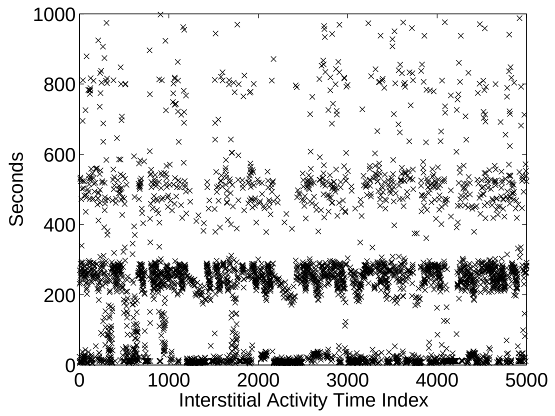


Figure 5.3: The interstitial activity time values for a Storm bot in a honeynet running in the wild.

We focus on scale-free networks because it is representative of many real-world networks, including unstructured P2P networks [96]. Empirical analysis by Dagon et al. [36] also suggest that the Nugache P2P botnet [138] has a scale-free structure. We set the number of nodes to 5,000 to represent a small botnet, following the simulation settings in previous work [36]. All of the edges are assumed to be undirected.

5.2.3 Metrics

We utilize the following two properties of graphs to quantify the "usefulness" of a botnet: 1) the size of the largest connected component, and 2) the inverse geodesic length. These metrics have been used by Dagon et al. [36] to compare the utility of different botnet topologies, and were also used in analyzing the resilience of various networks in the physics literature [63].

The fraction S of nodes in the largest connected component is an upper bound on the number of bots that are directly under the control of the attacker (assuming that she is part of one of the connected components). The more hosts that can carry out the attacker's commands, the larger the scale of the attack that can be launched, e.g., in performing denial-of-service attacks or sending spam.

In addition to controlling many infected hosts, another property that is likely to be of importance to the attacker is the efficiency of communication, i.e., how long it takes for messages to be relayed through the botnet. In practice, this is dependent on both the number of hops between nodes and their communication frequency. For example, if nodes communicate with their neighbors every T seconds, then the time it takes for a node to reach a peer h hops away would be roughly $h\frac{T}{2}$ seconds. As an estimate of the value of T for real bots, Figure 5.3 shows the interstitial activity time values (see Section 4.3.3) observed for a Storm bot in a honeynet running in the wild. The horizontal bars in the figure reflect regularities in the bot's activities, with the dominant frequency of communication being every 250 seconds or so. Setting $T = 250$, reaching a node 10 hops away would take roughly 1,250 seconds (around 20 minutes). This suggests that a larger hop count between nodes can increase the communication delay by minutes or even hours.

We focus on the average number of hops between pairs of nodes as a measure of the botnet communication efficiency. Specifically, let N be the total number of nodes, V be the set of nodes, $|V| = N$, and $d(u, v)$ be the length of the shortest path between node u and node v. The inverse geodesic length [63] is defined as

$$L^{-1} = \frac{1}{N(N-1)} \sum_{u \in V} \sum_{v \neq u, v \in V} \frac{1}{d(u, v)} \tag{5.3}$$

Measuring the inverse geodesic length is particularly useful in cases where the graph may be disconnected, since the distance $d(u, v)$ between two nodes u and v that belong to separate connected components would be infinite (and so its contribution to L^{-1} is zero). The larger L^{-1} is, the shorter the distances between nodes, and hence more efficient their communication. In evaluating the effect of assortativity on network takedown attempts, we are more interested in measuring the *normalized* inverse geodesic length, which is defined as

$$\hat{L}^{-1} = \frac{\sum_{u \in V} \sum_{v \neq u, v \in V} \frac{1}{d'(u,v)}}{\sum_{u \in V} \sum_{v \neq u, v \in V} \frac{1}{d(u,v)}} \tag{5.4}$$

where $d'(u, v)$ is the *modified* length of the shortest path between nodes u and v, that is, after takedown efforts or after the network tries to heal itself. Note that both the numerator and denominator in $\hat{L}^{-1}$ are summed over the original set of nodes, V. However, nodes that are removed have infinite distance to the rest of the network, the inverse of which is zero, and so do not contribute to the sum in Eqn. 5.4. The value that $\hat{L}^{-1}$ takes ranges from 0 to 1, where a smaller value indicates more disruption to network communication, and lower communication efficiency.

We measure $\hat{L}^{-1}$ and S of a network before and after takedown to evaluate the effectiveness of the takedown strategy (Section 5.3), and also measure them after the network attempts to "heal" itself to assess the effectiveness of recovery mechanisms (Section 5.4).

5.3 Network Resilience

In attempts to take down a P2P botnet, network administrators may wish to prioritize their efforts to focus on the more "important" nodes first, i.e., nodes whose removal will cause the most disruption to botnet operation. Using the two metrics described in Section 5.2.3, we investigate the effectiveness of botnet takedown strategies, and how they may be sensitive to the assortativity of the network.

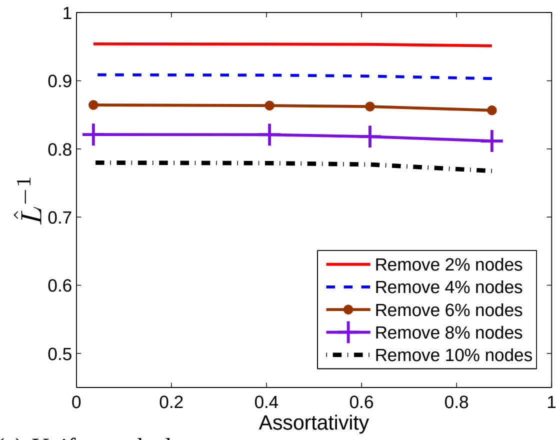
5.3.1 Uniform and Degree-Based Takedown Strategies

We first focus on strategies explored in previous works that study botnet models [36, 41, 90, 158]:

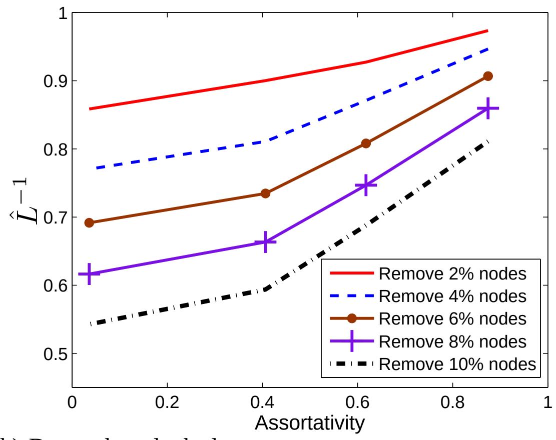
- **Uniform takedown**: removing nodes from the network by selecting them uniformly at random.
- **Degree-based takedown**: removing nodes from the network in descending order of node degree, that is, targeting high-degree nodes first.

Uniform takedown is similar to that which occurs when users and network administrators patch infected hosts as they are discovered, without coordinating bot discoveries or patching activities. It has also been used to study random failures in the context of communication networks or biological networks [3]. While most networks are found to be resilient to uniform takedown, they are much more vulnerable to a degree-based strategy. This targeted takedown strategy is especially effective against scale-free networks, since the few highly-connected "hub" nodes responsible for maintaining the connectivity of the network are removed first, e.g., the "super-peers" that are found in unstructured P2P networks. Using edges to indicate communication between nodes, the degree can be interpreted as the number of hosts with which a node communicates, and has also been used in network intrusion detection systems (e.g., [98, 128, 130]). In practice, these strategies do not necessarily require access to the entire network graph, but can be applied to takedown efforts within a subgraph as well, e.g., within a local network. We further discuss implementation challenges in Section 5.5.

As described in Section 5.2.2, we adjust the standard deviation σ of the edge probability distribution $e_{j,k}$ to generate networks of varying assortativity. For a scale-free network with 5,000 nodes, we set σ to 1, 5, 10, and 15 to obtain networks covering a range of assortativity from 0.04 to 0.87. Figures 5.4 and 5.5 show how networks with varying levels of assortativity respond to uniform and degree-based takedown, when 2%, 4%, 6%, 8%, or 10% of nodes are removed according to each strategy. The numbers are averaged over 50 networks generated for each value of σ. We omit the standard deviations from the plots since they are generally small, that is, within 0.007 for both $\hat{L}^{-1}$ and S.

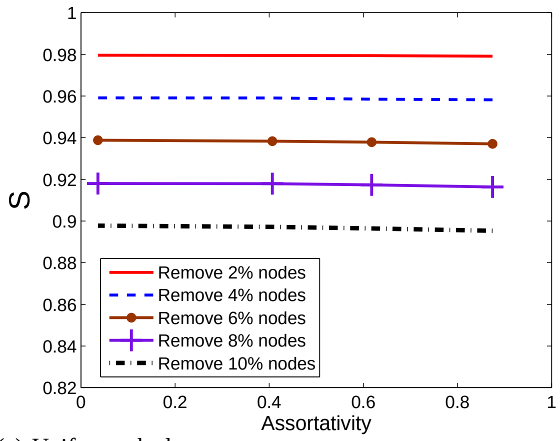


(a) Uniform takedown.

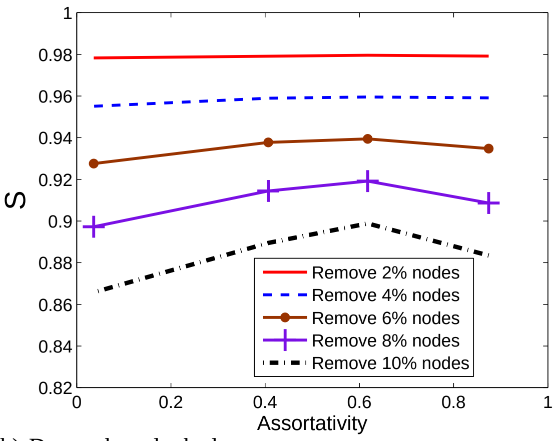


(b) Degree-based takedown.

Figure 5.4: The average normalized inverse geodesic length after uniform or degree-based takedown strategies.



(a) Uniform takedown.



(b) Degree-based takedown.

Figure 5.5: The average fraction of nodes in the largest connected component after uniform or degree-based takedown strategies.

We find the degree-based strategy to be much more effective at taking down a network compared to uniform takedown, in agreement with previous works. However, as shown in Figure 5.4(b), the effectiveness of the degree-based strategy is highly dependent on the level of assortativity of the network. A lower assortativity, e.g., toward the left of Figure 5.4(b), results in the network experiencing a larger decrease in $\hat{L}^{-1}$ after takedown attempts. The difference between the decrease in $\hat{L}^{-1}$ for assortative and non-assortative networks grows as more nodes are removed. When 10% of the nodes are removed via the degree-based strategy, this difference can be as much as 0.26. A similar phenomenon can be observed in Figure 5.5(b) for the fraction S of nodes in the largest connected component. With the exception of highly assortative networks (e.g., greater than 0.6), the fraction of nodes retained in the largest connected component increases with the level of assortativity. That is, more bots remain reachable to the attacker in moderately assortative networks.

The higher resilience in assortative networks can be attributed to nodes of similar degree "clustering" together. When the high-degree nodes are removed due to the degree-based strategy, only a connected subset of neighboring nodes are lost in effect. Moreover, since high-degree nodes tend to connect to each other, fewer of their edges are attached to nodes of low degree — who would be prone to isolation if their neighbors were removed. However, this also means that there are fewer high-degree nodes that can act as "bridges" between clusters of nodes with varying degrees. As more high-degree nodes are removed, the loss of those "bridging" nodes eventually cancels out other factors contributing to resilience as assortativity increases, and the network can disintegrate, as shown on the far right of Figure 5.5(b). These discrepancies in how networks are affected by the same takedown strategy underline the importance of taking assortativity into account, both in evaluating takedown strategies and in considering botnet network models.

5.3.2 Other Takedown Strategies

While the degree-based strategy is much more effective than the uniform strategy, the former is sensitive to the level of assortativity in the network, as shown in Figures 5.4 and 5.5. In the search for a takedown strategy that would be effective even for assortative networks, we explore alternative approaches based on other graph properties, described below.

- **Neighborhood connected components**: We define the local neighborhood of a node u to be those reachable within h hops from it. Figure 5.6(a) shows an example of the neighborhood of node u within three hops, where the edge labels indicate distances to u. If we were to remove u from the network, its local neighborhood would be split into separate "connected components", as shown in Figure 5.6(b). Without a view of the entire network, the number of "connected components" that remains in the neighborhood of a node can be an approximation of its local importance, since communication between components may have to be routed through u. Hence, as an alternative takedown strategy, we remove nodes in descending order of the number of connected components in their local neighborhood. A similar metric has also been used to detect hit-list worms [27].

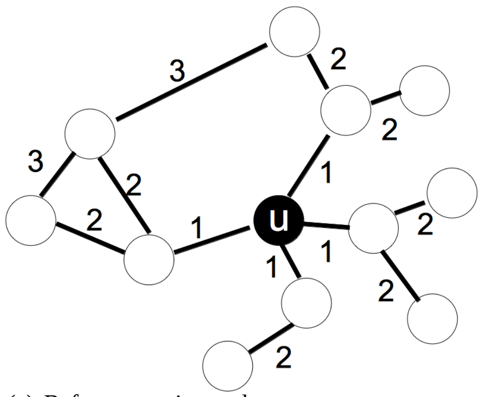
- **Closeness centrality**: Closeness centrality for a node u is defined as the sum of the inverse geodesic distance from u to all other nodes in the network. A larger value indicates that the node is at a more "centered" location, and has more influence over the spread of information within the network. In this takedown strategy, we remove nodes in descending order of their closeness centrality.

- **Clustering coefficient with degree**: The clustering coefficient measures how dense the connections are between the neighbors of a node. For a node u, it is defined as the number of edges that exist between u's neighbors, divided by the number of possible edges between u's neighbors. For example, in Figure 5.7, this value for u is 4/10, while that for all other nodes is 1. A smaller value means that the neighbors of u may be disconnected if it were not for u. Ignoring nodes with the smallest degrees — in our tests, nodes with degree less than one-fifth of the maximum degree — we remove nodes in increasing order of their clustering coefficient, and among those with the same clustering coefficient, in decreasing order of degree.

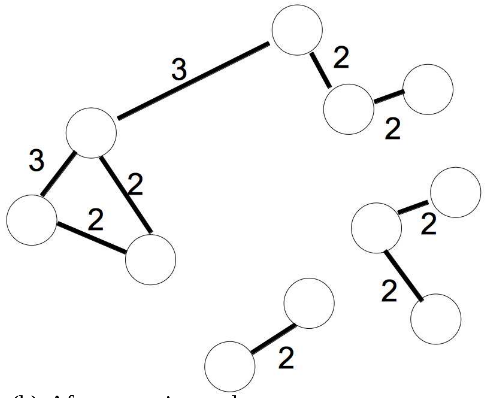
- **Breadth-first traversal (BFS)**: Given an initial node u, this strategy remove nodes from the network according to a breadth-first traversal with u as the root. That is, we remove node u, the one-hop neighbors of u, the two-hop neighbors of u, and so on, in that order. The choice of the initial node is biased in favor of higher degree nodes, i.e., the probability of a degree k node chosen as the initial root is proportional to k. This strategy simulates that of network administrators discovering other infected hosts by examining the peer list of a bot that is already captured. It is also similar to the "tree-like" bot disinfection strategy studied by Davis et al. [41], and to cascaded node failures in complex networks [34, 148].

- **Uniform set expansion**: Another strategy to identifying nodes to take down is to examine the set B of bots that are already discovered, e.g., through the uniform strategy, and then expand B by uniformly select among all one-hop neighbors of nodes in B. In practice, this strategy can be performed by examining the peer lists of captured bots, similar to the breadth-first traversal.

Figures 5.8 and 5.9 show the fraction S of nodes in the largest connected component, and the normalized inverse geodesic length $\hat{L}^{-1}$ after each of the above takedown strategies, for networks of different levels of assortativity. The results are plotted after removing 2% or 10% of the nodes, and averaged over 50 networks generated for each level of assortativity. The standard deviations are all within 0.03 for both $\hat{L}^{-1}$ and S. Compared with the uniform and degree-based strategies discussed earlier, the connected components strategy seems more effective at lowering the connectivity of the network, as shown in Figure 5.8,



(a) Before removing node u.



(b) After removing node u.

Figure 5.6: An example of the connected components within the neighborhood of node u. The edge labels indicate number of hops to u.

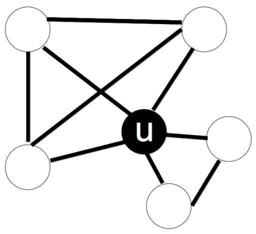
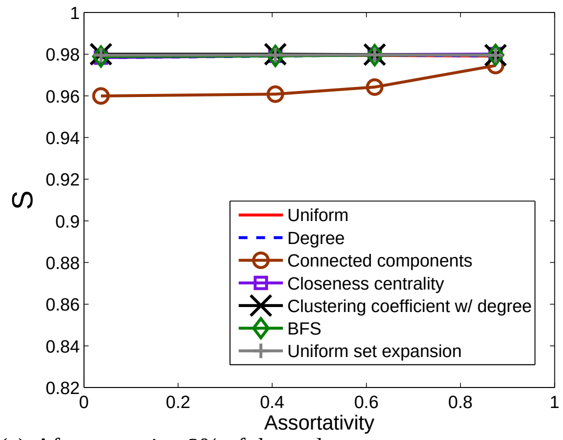


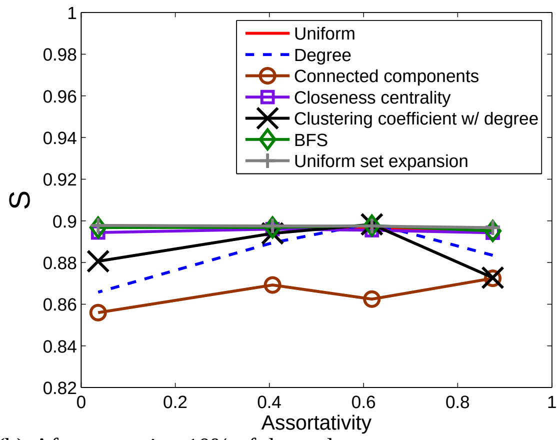
Figure 5.7: An example of the edges between the neighbors of node u.

while the clustering coefficient strategy is more effective at decreasing the network communication efficiency, as shown in Figure 5.9. In both of these cases, the alternative takedown strategy out-performs the degree-based strategy that previous works found to be effective [36, 41, 158].

One of the reasons that the clustering coefficient strategy works well is because nodes that "cluster" together in assortative networks are likely to have higher clustering coefficient as well, since their neighbors also have similar degree. However, while the nodes at the center of a "cluster" may have a clustering coefficient close to 1, this value is likely to be much smaller for those connecting the cluster to the rest of the network. For example, all nodes in Figure 5.7 have a clustering coefficient of 1 except for node u, who turns out to be the "bridge" between the two clusters of degree two and three nodes. The removal of nodes with small clustering coefficient in this strategy is hence likely to affect communications within the network.

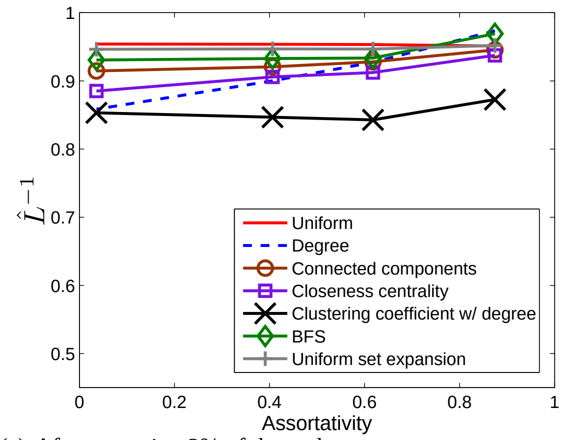


(a) After removing 2% of the nodes.

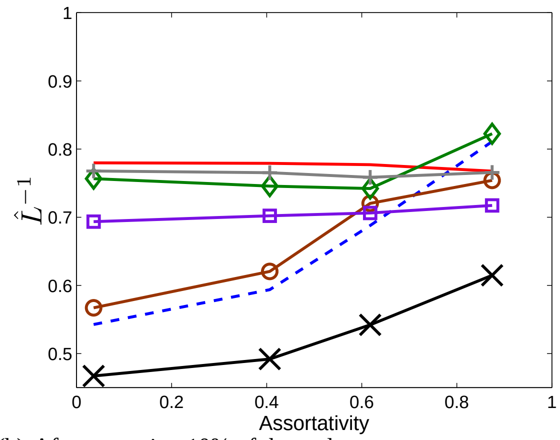


(b) After removing 10% of the nodes.

Figure 5.8: The average fraction of nodes in the largest connected component after removing 2% or 10% of the nodes according to each takedown strategy.



(a) After removing 2% of the nodes.



(b) After removing 10% of the nodes.

Figure 5.9: The average normalized inverse geodesic length after removing 2% or 10% of the nodes according to each takedown strategy.

5.4 Network Recovery

The dynamism inherent in peer-to-peer networks means that each individual bot is required to adapt to changes in its surroundings, for example, due to newly infected hosts joining the network or current peers going offline, even without takedown attempts taking place. These mechanisms for nodes to discover previously unknown peers and create new edges hence provide opportunities for the network to *recover* itself, i.e., restoring connectivity or reconstructing shortest paths between nodes, in the face of takedown attempts.

While previous works tend to regard a botnet as a static entity, and evaluate changes to the network immediately after takedown efforts as a measure of their effectiveness, we explicitly consider the ability of dynamic networks to heal themselves. Specifically, we model a recovery process where nodes can "look out" to a distance h and find peers that are within h hops. When a node loses a neighbor, e.g., due to takedown, it compensates for that lost neighbor by creating a new edge to a randomly selected node within distance h from it. As has been documented in P2P bot studies [17, 66, 116, 132, 138], a bot has only a local view of the network according to its peer-list, which it updates by constant exchanges with its neighbors. The h-neighborhood of a node u hence represents those hosts that are on u's peer-list, to which u looks for maintaining connectivity with the rest of the botnet.

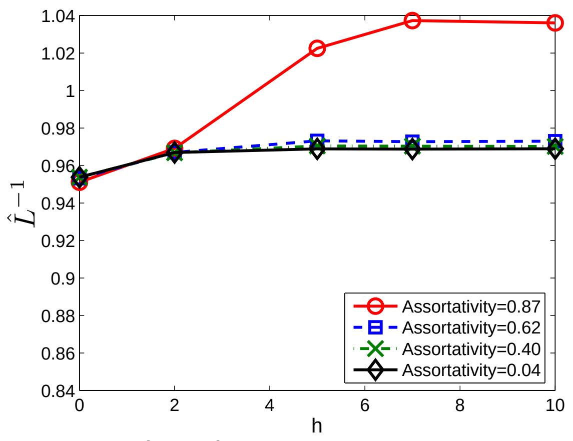
5.4.1 Recovering from Uniform and Degree-Based Takedown Attempts

We first consider the ability of botnets to recover after takedown attempts employing the uniform or degree-based strategies described in Section 5.3.1. We focus on the inverse geodesic length metric, since it better illustrates the difference between networks of varying levels of assortativity. Figure 5.10 shows the normalized inverse geodesic distance $\hat{L}^{-1}$ for networks after they attempt to recover from uniform or degree-based takedown strategies, when 2% or 10% of the nodes are removed. The look-out distance h is set to 2, 5, 7, and 10. As h increases, $\hat{L}^{-1}$ increases as well, even reaching above 1 in Figure 5.10(a), i.e., the shortest distance between nodes becomes even shorter than before the takedown! However, while the increase in $\hat{L}^{-1}$ for networks with lower assortativity falls flat after a small h (even decreasing slightly, as in Figure 5.10(d)), the increase for networks with higher assortativity continues.

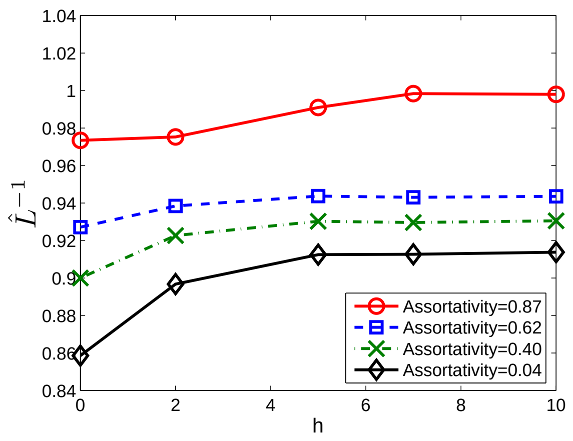
One reason for the continued recovery benefit enjoyed by assortative networks is high-degree nodes "clustering" together, since nodes tend to connect to others of similar degree. A node that is able to reach a high-degree node upon "looking out" is likely to be able to reach other high-degree nodes as well at a similar distance. This increases the probability that a compensation edge attaches to a high-degree node, hence shortening path lengths within the network and resulting in a higher $\hat{L}^{-1}$. This phenomenon is more pronounced in networks recovering from uniform takedown (see Figures 5.10(a) and 5.10(c)), since fewer high-degree nodes remain after the degree-based strategy.

5.4.2 Recovering from Takedown Attempts Using Other Strategies

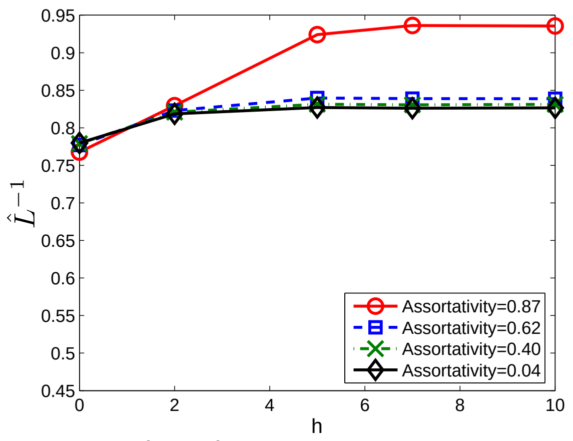
Figures 5.11 and 5.12 show how networks of high and low assortativity recover from those alternative takedown strategies described in Section 5.3.2, when 2% or 10% of the nodes are removed. We observe a



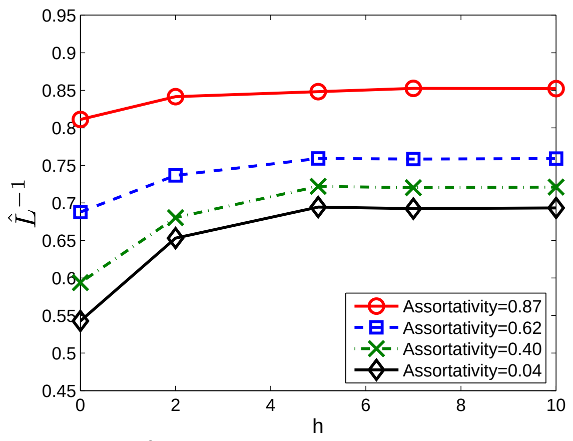
(a) Recovery after uniform takedown by removing 2% nodes.



(b) Recovery by degree-based takedown by removing 2% nodes.



(c) Recovery after uniform takedown by removing 10% nodes.

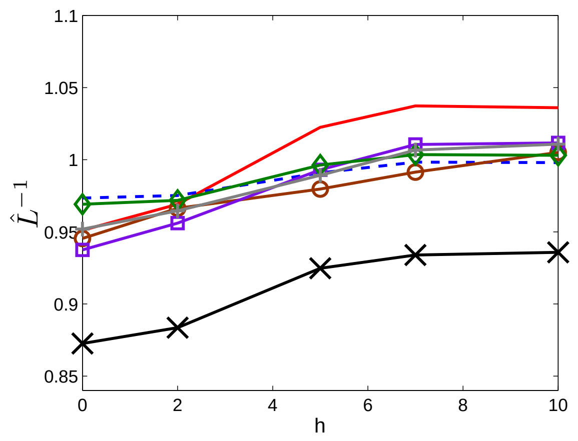


(d) Recovery after degree-based takedown by removing 10% nodes.

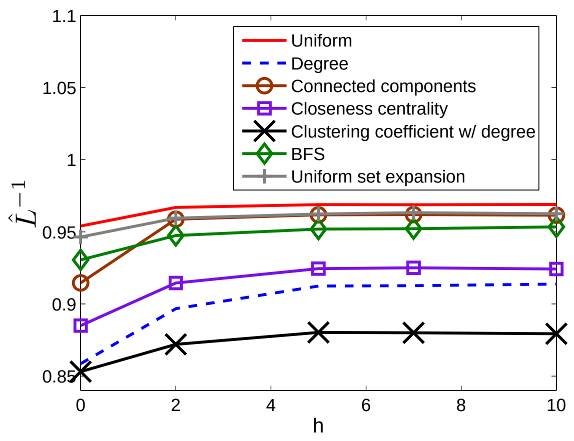
Figure 5.10: The average normalized inverse geodesic length after recovering from uniform or degree-based takedown, when 2% or 10% of the nodes are removed, for various values of the look-out distance h.

trend similar to the recovery from uniform and degree-based strategies, where networks with higher levels of assortativity experience continued recovery benefits with the look-out distance h (Figure 5.11(a)). Less assortative networks, on the other hand, do not benefit much after a look-out distance of 2 or 3 (Figure 5.11(b)). Regardless of the takedown strategy, assortative networks still have higher communication efficiency after recovery, in terms of $\hat{L}^{-1}$, than less assortative networks.

In addition to being one of the most effective strategies (see Section 5.3.2), we also find takedown attempts based on clustering coefficient with degree to be the most difficult one for a network to recover from, as shown by low values of $\hat{L}^{-1}$ in Figures 5.11 and 5.12. In fact, when 10% of the nodes are removed from the same network, the difference between $\hat{L}^{-1}$ after recovering from the uniform and the

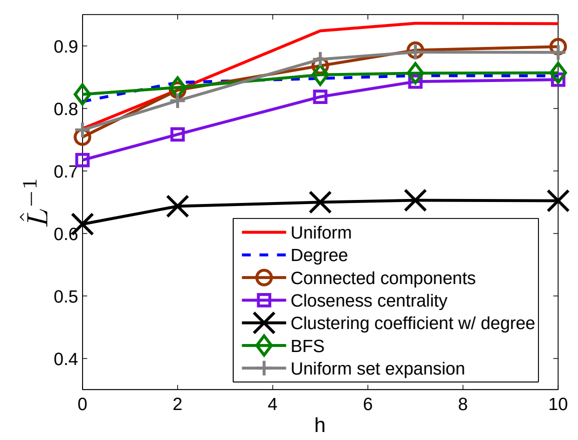


(a) Networks with assortativity at 0.87.

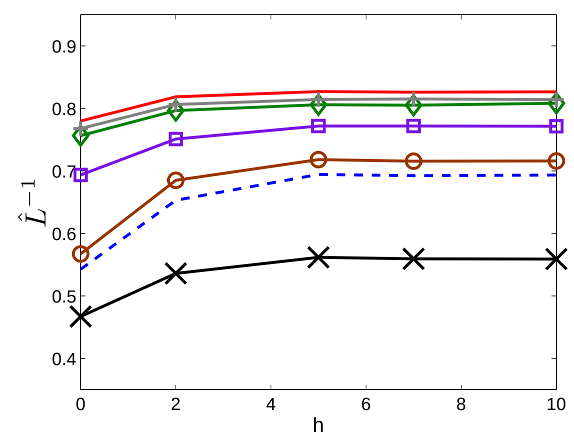


(b) Networks with assortativity at 0.04.

Figure 5.11: Recovery for networks of high and low assortativity when 2% of the nodes are removed according to each strategy.



(a) Networks with assortativity at 0.87.



(b) Networks with assortativity at 0.04.

Figure 5.12: Recovery for networks of high and low assortativity when 10% of the nodes are removed according to each strategy.

clustering coefficient strategies can be as much as 0.28 (Figure 5.12). Similarly, the $\hat{L}^{-1}$ after recovering from the clustering coefficient strategy is up to 0.2 less than the $\hat{L}^{-1}$ after recovering from the degree-based strategy. This shows that the clustering coefficient takedown strategy can be a better alternative to one based solely on degree.

5.5 Discussion

In this section, we discuss implementation challenges to takedown strategies studied in this chapter, and alternative methods of analytically modeling networks.

Applying takedown strategies in practice. Perhaps one of the reasons for the widespread study of the degree-based strategy is that it can be applied easily in practice. For example, if the degree of a node is interpreted as the number of hosts with which it communicates in some time interval, then identifying a node's degree can be performed on the basis of flow records (e.g., Cisco NetFlows [1]) that are collected from a router (or routers) that its traffic traverses. Notably, a node's degree can be determined solely by observing traffic to and from it, without requiring additional knowledge about the entity at the other end of the communication.

Other graph properties, however, may not be so straightforward to measure. For instance, takedown strategies based on clustering coefficient or neighborhood connected components depend on observing communications between the neighbors of a node, and may require collaboration between multiple administrative domains, such as that proposed by Xie et al. to trace the origin of worm propagations [152]. Another approach is to examine the peer-lists an infected host receives from its neighbors, assuming that such data can be captured (i.e., it is not sent encrypted, and full packet capture is enabled on the network). If a node u has two communicating neighbors, those nodes should be listed on each other's peer-lists, and so the fact that they communicate can be inferred by identifying overlaps between u's neighbors and peer-lists sent to u. Of course, in cases where communications between some neighbors of an infected node are visible neither directly nor by inference, takedown strategies requiring this information can be applied considering only those neighbors for which communications are visible.

Modeling networks analytically. Rather than assuming a particular network topology, e.g., random, scale-free, or small-world, or a specific level of assortativity, another approach to modeling networks is to specify a set of actions governing the behavior of nodes at each step in time, and analytically determine properties of the resulting network. This type of growing network models have been used extensively in the physics domain of complex networks [6, 42, 86, 100, 127]. Given knowledge of individual bot behaviors and how they interact with each other from P2P bot studies [17, 66, 116, 138], it seems likely that analytical network models from the physics literature can be adapted to characterize P2P botnets. In fact, a recent work by Li et al. [91] used this approach to derive the degree distribution of a botnet where new nodes join the network by "copying" the edges of an existing node that it chooses at random.

However, these analytical approaches do make other assumptions about the underlying network that they attempt to model in order to simplify calculations. Specifically, by assuming that both the age of the network t and the network size N is large, $t \to \infty$, $N \gg 1$, all actions experienced by a node are approximated by the *expected* action, e.g., when a node creates one edge at random, the degree of all other nodes increases by $1/N$, where the denominator N is also replaced by the expected value. These assumptions may not be applicable to botnets in practice, since 1) network administrators will be equally, if not more, concerned about infections in the early stages of a botnet when t is small; 2) botnets have been found to consist of a few hundred or thousand nodes only, and are commonly rented out in small numbers, e.g., for sending spam; 3) to a network administrator managing a local network, N certainly does not grow indefinitely; and 4) approximating aspects of network growth using expected values introduces error that could potentially be magnified by a bot designed counter to assumptions that these approximations imply.

As a simple demonstration of the separation between analytical models and actual network growth, we examine a derivation by Callaway et al. [19] of the assortativity of a simple network growth model. In each time step, the model assumes that one node joins the network, and with probability δ an edge forms between two nodes selected at random. Their derivation of the assortativity is based on a rate equation specifying the *expected* increase in the number of edges that connect nodes of remaining degree j and k at each time step, and makes the same assumptions as described above. Figure 5.13 shows the expected assortativity of the network as approximated by Callaway et al. for various values of δ. The actual average values from simulations are also plotted in the figure, with one standard deviation shown as error bars. To generate these values, we generated 50 networks for each value of δ, and set the number of time steps (i.e., number of nodes) to 1,000. Figure 5.13 shows that the expected assortativity as predicted by Callaway et al. can differ from the actual average assortativity by an amount that approaches or, in some cases, exceeds one standard deviation. This suggests that the simplifying assumptions typically employed in analytical models may cause nontrivial deviations from practice.

5.6 Chapter Summary

Peer-to-peer (P2P) botnets, in contrast to their centralized counterparts, do not have a single point-of-failure and are difficult to take down. Identifying and removing those nodes that are "important" to the connectivity or communication efficiency of a botnet is hence critical to disrupting its operation. Toward

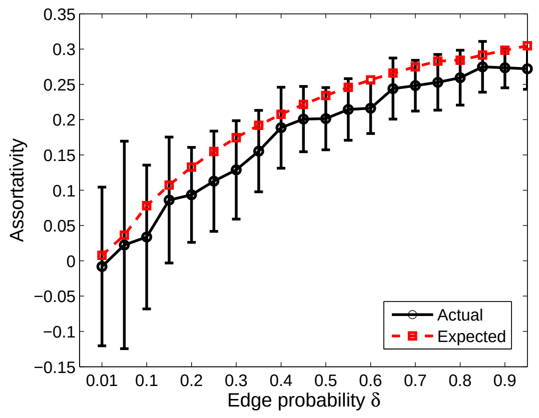


Figure 5.13: The expected assortativity, shown in the dashed line, versus the actual average value from simulations, with one standard deviation shown with error bars.

this goal, several previous works have modeled P2P botnets using theoretical network models [36, 41, 90]. These works compare the resilience of various network topologies to uniform or degree-based node removals, and quantify the effectiveness of these takedown strategies using graph properties, including the inverse geodesic length of the resulting network or the fraction of nodes in the largest connected component.

In this chapter, we observe that previous works do not consider an important structural property of networks, namely assortativity. Empirical measurements on network traffic from bots in a honeynet running in the wild suggest that this property can be quite high for botnets in practice. We show that in omitting the presence of assortativity in botnet models, and without considering the effect of dynamic networks actively recovering from node failures, previous works may have over-estimated the effectiveness of recommended takedown strategies. In addition, we identify alternative strategies that are more effective than those in previous works for botnets with high assortativity.

Chapter 6

Conclusions and Future Work

The work in this dissertation proposes new techniques to detecting hosts infected with stealthy malware. Infected hosts, i.e., bots, can exfiltrate sensitive data to adversaries, or lie in wait from commands from a bot-master to forward spam or launch denial-of-service attacks. However, it is difficult to detect bots, since their activities are subtle and do not disrupt the network. In addition, infected hosts can encrypt their traffic and utilize existing protocols for communication to further mask their activities.

Our key observation is that infected hosts exhibit similar characteristics in their network activities that are distinct from those of benign hosts. Our approach hence identifies bots by aggregating “similar” network traffic, which are collected in the form of flow records that contain coarse summaries of each connection. Under this framework, we present techniques to identify both infected hosts participating in centralized botnets and those that communicate over peer-to-peer networks. We further develop a passive browser fingerprinting method to detect malware that are not confined to hosts of a single operating system platform. To complement our empirical analyses, we also study peer-to-peer botnets analytically using models from network theory, and investigate how a structural characteristic of networks affects the effectiveness of botnet takedown strategies.

In this chapter, we summarize our contributions, and discuss limitations of our approach and potential future directions.

6.1 Contributions

Behavioral characteristics to detect stealthy malware. We identify characteristics in network traffic that can distinguish the behaviors of infected and benign hosts. In particular, we focus on characteristics that pertain to basic properties of malware operation, including the coordinated and automated nature of infected hosts, and unique goals and circumstances behind how they utilize existing communication protocols. Our approach hence does not depend on observing specific malware activities, e.g., sending spam or performing denial-of-service attacks, and has the potential to detect infected hosts prior to them engaging in such disruptive events. Moreover, we only utilize information contained in coarse traffic summaries, e.g., Cisco NetFlow [1] or Argus flows [1], to avoid bandwidth and storage requirements for capturing full packets.

In Chapter 2, we describe a system that detects hosts participating in centralized botnets. Our system, TĀMD, aggregates network traffic that share the same busier-than-usual destinations, that have similarly structured payload, and that are also associated with hosts running similar software platforms. TĀMD can reliably identify network traffic generated from real bot instances among all traffic crossing the border of our university campus network. Even for homogeneous networks where the majority of internal hosts are of the same platform (e.g., enterprise networks), forming traffic aggregates based only on the destination and payload characteristics would still yield accurate results.

Due to P2P technologies being used by both P2P bots and file-sharing hosts, botnet traffic will tend to "blend into" a background of P2P file-sharing. Being able to separate the two types of traffic hence becomes the main challenge in detecting P2P bots. In Chapter 4, we show that distinct goals and motivations driving the use of the same P2P network protocol will give rise to varying behaviors — particularly those associated with the hosts' volume of traffic, the amount of peer churn, and whether their activities are human-driven or machine-driven — that can be used to distinguish bot traffic from those of file-sharing hosts.

Efficient algorithms to analyze network traffic. In addition to identifying relevant behavioral characteristics that can detect stealthy malware, our contributions also include algorithms to analyze network traffic efficiently. The algorithms are drawn from diverse areas, including statistics, machine learning, and metric embeddings.

To our knowledge, we are the first to detect malicious traffic by computing a type of edit distance using techniques that can scale to high data rate environments (see Section 2.2.2). The TĀMD system performs this by first embedding the edit distance with moves metric into L1-space [30], and then deploying a near-neighbor search algorithm [39]. As a result, traffic with similar payloads can be found in time roughly

[1]`http://www.qosient.com/argus`

proportional to the number of flows.

We also explored various ways of representing hosts, or representing the characteristics by which we wish to aggregate network traffic. For example, a binary vector is used to denote the external destinations each host contacts (Section 2.2.1), allowing statistical methods to be applied to construct destination aggregates. In Section 4.3.3, we construct histograms of hosts' interstitial activity times to reflect the degree to which their behaviors are machine-like.

Adversarial cost to evade detection. In the arms race between attackers and network defenders, new malware variants have continued to demonstrate ways of circumventing existing intrusion detection systems. In our technique of detecting peer-to-peer bots by distinguishing them from peer-to-peer file-sharing hosts, we specifically quantify the operational cost for a bot to evade detection by masquerading as a file-sharing host (Section 4.5).

In particular, our evaluations show that a P2P bot will be able to escape detection by our tests, but at the cost of significantly changing its network behavior, e.g., generating more than 20 times its original traffic volume per flow, and increasing the fraction of new hosts it contacts by a factor of 1.5. Both evasion attempts can compromise the stealthiness of the bot, making their presence in the network observable through other means (e.g., scan detection) or even by the owner of the infected machine.

Infected hosts can also attempt to evade our technique to separate human-driven and machine-driven traffic by randomizing timing patterns in their communications, e.g., by adding random delays between its activities. This can introduce variances in the bots' interstitial time distributions, and cause our techniques to fail to correctly cluster them together. However, for this evasion attempt to be effective, our experiments show that the bots would need to add significant delay between their activities, e.g., up to minutes or hours, potentially making the botnet less responsive.

Passive application fingerprinting technique. We develop a new technique to infer the browser implementation on a host by passively observing its traffic to common sites (Chapter 3). In particular, our technique utilizes only information contained in coarse traffic summaries, including the byte and packet counts, and the start and end times of each flow. For each flow generated by a browser in retrieving the contents of a web page, we extract statistics related to its size and duration. The statistics are aimed at capturing differences related to the order by which browsers download objects on a given page, the number of objects retrieved in each connection, the number of parallel connections, and other discrepancies in browser implementations.

We show that our browser fingerprinting technique can be incorporated into TĀMD, allowing it to detect application-dependent malware in addition to those that are O/S-dependent. As a second application of browser fingerprinting, we demonstrate that knowledge of the browser implementation can yield a more precise deanonymization of the web *sites* than has previously been achievable from flow records. In our experiments, a per-browser website classifier achieves up to a 17% accuracy improvement to a generic website classifier.

Analytical botnet models. To complement our empirical analyses, we also study botnets analytically using models from network theory (Chapter 5). Each node in the network graph represents an infected host, and edges reflect communications between the hosts. Network models can allow the "usefulness" of a botnet to be quantified using graph properties (e.g., the number of hops between nodes, the size of the largest connected component), such that the effectiveness of various botnet takedown strategies can be evaluated.

We focus on studying an important structural property of networks — assortativity [102] — and its effect on the resilience and recovery ability of botnets. We show that, without considering this property, previous works on modeling botnets may have over-estimated the effectiveness of certain botnet takedown strategies. In addition, we identify alternative strategies that are more effective than those in previous works for botnets with high assortativity.

6.2 Limitations

Since our stealthy malware detection approach relies on observing similarities in hosts' network behaviors, it will be more successful in large networks consisting of hosts with diverse platforms and applications, and where more than one infected host are present. An enterprise network, for example, will likely have the majority of its internal hosts running the same operating system, potentially limiting the effectiveness of the platform aggregation in TĀMD. That said, in evaluations performed on traffic from our university campus network and from real bot instances, we find that TĀMD yields good results even with only destination and payload aggregation (see Section 2.4.3).

We also expect the infected hosts to react to bot-master commands or communicate with each other in a timely and roughly synchronized manner, e.g., active at least once an hour or once per day. Botnets that are less responsive will require longer observation windows, where analysis can be complicated by other factors, including a higher probability of treating traffic from multiple distinct machines as one, e.g., because of DHCP churn. That said, sparse or infrequent communications can also result in the bot-master's commands requiring more time to propagate through the botnet. This may limit the type of attacks that the bots can perform (for example, distributed denial-of-service attacks that rely on collected efforts from a large number of time-synchronized bots), and potentially the botnet size as well.

In cases where bot-masters plan their attacks well in advance (e.g., to allow time for commands to propagate to a large number of bots), or where the infected hosts are chosen selectively to perform specific attacks (and hence do not require frequent communications with the bot-master), activities from infected hosts can be few and far between. An example of this is the Stuxnet worm [45], which specifically aims to subvert control of industrial systems. This malware was able to stay under the radar by rate-limiting the number of hosts each bot infects, and interacting mostly with other vulnerable or infected hosts on the same local network. In particular, since the type of hosts it infects typically does not have Internet access, the operations of the Stuxnet botnet do not rely on bot-master command and control. Detecting this type of "advanced persistent threats" may require monitoring intra-network traffic, in addition to the deployment of host-based intrusion detection systems.

As bots become more sophisticated, their activities can become more "human-like" in the future, e.g., crafting their communications so that they are statistically similar to those of humans. This may challenge the effectiveness of behavior-based malware detection techniques, such as the human-driven versus machine-driven test described in Section 4.3.3. However, we believe that malware operations are still fundamentally distinct from benign user behaviors, particularly in how activities are triggered (e.g., the command-driven nature of malware activities) and their intended goals. Leveraging these underlying differences will be important for future malware detection systems. In particular, rather than relying on heuristics or empirical observations, a more formal representation of benign host behavior may be required to characterize the circumstances under which their activities originate.

6.3 Future Work

This section describes directions for future work that address some of our limitations discussed in the previous section, and potential methods to enhance stealthy malware detection systems.

Quantifying adversarial cost. Intrusion detection and prevention systems today are commonly evaluated by their false positive rate and the detection rate. However, such systems are typically tested only on particular datasets available to the researchers, e.g., due to difficulties in obtaining public data. It is hence difficult to generalize their effectiveness to other environments, or to compare them against each other without those systems being publicly available. Moreover, considering adaptive adversaries that modify their strategies based on the detection techniques deployed (as we have seen in the arms race between attackers and defenders), it is possible that a proposed intrusion detection or anomaly detection system can be circumvented by new malware variants.

In light of these observations, another useful metric for evaluating intrusion detection systems would be the operational cost required for the adversaries to evade detection. We explored this metric briefly in Chapter 4, where we quantified the increase in traffic volume and number of destinations that a Plotter must contact in order to masquerade as a Trader. This approach can be generalized into a constraint optimization framework, where the detection rules employed by the intrusion detection system serve as *constraints* imposed on the operations of the adversary, whose goal is to maximize certain aspects of the botnet, e.g., the number of infected hosts under the adversary's control, the combined bandwidth or computing power of the botnet, or the communication latency between infected hosts. This will allow anomaly detection systems to be evaluated from a more objective point of view that is dataset-independent.

Correlating host profile changes. Infected hosts are often used by the bot-master to provide rogue network services, such as HTTP proxies or DNS servers that redirect victims to malicious domains, SMTP servers sending spam emails, or FTP servers providing spam templates. Having bots participate in these services hides the backend server controlled by the bot-masters, and makes it difficult to disrupt their operation.

However, these activities are also likely to cause changes to the role that the infected host plays in a network, since such services are usually performed by designated machines. Upon infection, a host thus instructed by the bot-master is likely to switch from being a client to a server, or from one type of server to another. While benign hosts can exhibit behavioral changes as well, e.g., when new applications are installed, they are rarely synchronized in their activities (excluding popular events, such as connections to popular sites like `www.google.com`). For this reason, more suspicious are previously independent hosts that exhibit "coincidental" changes to their network behaviors.

As an example, a host's network behaviors can be profiled by its connection patterns (e.g., whether they accept inbound connections, the duration and size of the connections, or the number and type of the hosts with which it interacts). Examined over multiple time windows, "coincidental" changes involving multiple hosts can be detected by data mining techniques, such as association rule mining, that identify patterns frequently appearing together. Similar data mining approaches have been applied to identify communication rules in a network, which can be used for monitoring and diagnosis [76]. Based on changes to hosts' network profiles, this method of identifying infected hosts has the potential to detect previously unseen malware. It is also more general than methods that rely on signatures or behaviors deemed suspicious, which require labor-intensive reverse engineering efforts by human analysts.

Distributed Malware Detection. The malware detection approach described in this thesis makes the assumption that there are multiple infected hosts within the network. While many malware instances propagate locally, e.g., through open network shares (such as Stuxnet [45], Phatbot [135], Conficker [117]), and multiple infections are likely to be found in large networks, this assumption may not hold in smaller networks or be useful for detecting bots belonging to small botnets. In these cases, it is beneficial for multiple distributed networks to collaborate, so that activities from more than one bot may be observed.

Some of our data analysis algorithms can be readily adapted to distributed data. For example, payload aggregates in TĀMD (see Section 2.2.2) can be performed across multiple networks by sharing only the hash functions and the bucket numbers to which flow payloads are mapped. Distributed versions of statistical techniques used in this work are also available, e.g., distributed PCA [69]. The challenge in this setting, however, is in combining data from multiple sources in a privacy-preserving manner, e.g., similar to that proposed by Xie et al. to trace the origin of worm propagations [152].

Bibliography

[1] Introduction to Cisco IOS netflow. White paper, Cisco Systems, Inc., 2006.

[2] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck, and J. Van der Merwe. Analysis of communities of interest in data networks. In *Passive and Active Measurement Workshop*, 2005.

[3] R. Albert, H. Jeong, and A.-L. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406, 2000.

[4] L. Amaral, A. Scala, M. Barthelemy, and H. Stanley. Classes of small-world networks. *National Academy of Sciences of the United States of America*, 97(21), 2000.

[5] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[6] A.-L. Barabási, R. Albert, and H. Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272:173–187, 1999.

[7] P. Barford and V. Yegneswaran. An inside look at botnets. *Malware Detection*, pages 171–191, 2007.

[8] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *2nd ACM SIGCOMM Workshop on Internet Measurement*, 2002.

[9] G. Bartlett, J. Heidemann, and C. Papadopoulos. Inherent behaviors for on-line detection of peer-to-peer file sharing. In *IEEE Global Internet Symposium*, 2007.

[10] S. A. Baset and H. Schulzrinne. An analysis of the Skype Peer-to-Peer Internet telephony protocol. In *25th IEEE International Conference on Computer Communications*, 2006.

[11] F. Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, 2005.

[12] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[13] R. Beverly. A robust classifier for passive TCP/IP fingerprinting. In *Passive and Active Measurement Workshop*, 2004.

[14] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[15] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the Zeus botnet crimeware toolkit. In *8th IEEE Annual International Conference on Privacy, Security and Trust*, 2010.

[16] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing Skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review*, 37(4):37–48, 2007.

[17] L. Borup. Peer-to-peer botnets: A case study on Waledac. Master's thesis, Technical University of Denmark, 2009.

[18] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. RFC 2722, 1999.

[19] D. S. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. J. Newman, and S. H. Strogatz. Are randomly grown graphs really random? *Physical Review E*, 64(4):041902, 2001.

[20] S. Chang and T. E. Daniels. P2P botnet detection using behavior clustering & statistical tests. In *2nd ACM Workshop on Security and Artificial Intelligence*, 2009.

[21] Z. Chen, C. Chen, and Q. Wang. Delay-tolerant botnets. In *18th IEEE International Conference on Computer Communications and Networks*, 2009.

[22] D. Cheng, R. Kannan, S. Vempala, and G. Wang. A divide-and-merge methodology for clustering. *ACM Transactions on Database Systems*, 31(4), 2006.

[23] K. Chiang and L. Lloyd. A case study of the Rustock rootkit and spam bot. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[24] C. Cho, J. Caballero, C. Grier, V. Paxson, and D. Song. Insights from the inside: A view of botnet management from infiltration. In *3rd USENIX Conference on Large-scale Exploits and Emergent Threats*, 2010.

[25] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planet-Lab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[26] M. P. Collins and M. K. Reiter. Finding peer-to-peer file-sharing using coarse network behaviors. In *11th European Symposium on Research in Computer Security*, 2006.

[27] M. P. Collins and M. K. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *10th International Symposium on Recent Advances in Intrusion Detection*, 2007.

[28] D.E. Comer and J.C. Lin. *Probing TCP implementations*. Purdue University, Dept. of Computer Sciences, 1993.

[29] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *1st Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2005.

[30] G. Cormode and S. M. Muthukrishnan. The string edit distance matching problem with moves. In *13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[31] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[32] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, and M. K. Reiter. On web browsing privacy in anonymized NetFlows. In *16th USENIX Security Symposium*, 2007.

[33] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1), 2007.

[34] P. Crucitti, V. Latora, and M. Marchiori. Model for cascading failures in complex networks. *Physical Review E*, (69):045104, 2004.

[35] P. Crucitti, V. Latora, M. Marchiori, and A. Rapisarda. Error and attack tolerance of complex networks. *Physica A*, 340:388–394, 2004.

[36] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A taxonomy of botnet structures. In *24th Annual Computer Security Applications Conference*, 2007.

[37] G. B. Dantzig. Application of the simplex method to a transportation problem. In *Activity Analysis of Production and Allocation*, pages 359–373. John Wiley and Sons, 1951.

[38] N. Daswani, M. Stoppelman, the Google Click Quality, and Security Teams. The anatomy of clickbot.A. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[39] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *20th ACM Symposium on Computational Geometry*, 2004.

[40] C. Davis, J. Fernandez, S. Neville, and J. McHugh. Sybil attacks as a mitigation strategy against the Storm botnet. In *3rd IEEE International Conference on Malicious and Unwanted Software*, 2008.

[41] C. R. Davis, S. Neville, J. M. Fernandez, and J.-M. Robert. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *13th European Symposium on Research in Computer Security*, 2008.

[42] S. N. Dorogovtsev and J. F. F. Mendes. Scaling properties of scale-free evolving networks: Continuous approach. *Physical Review E*, 63:056125, 2001.

[43] P. Erdös and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[44] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *16th International World Wide Web Conference*, 2007.

[45] N. Falliere, L. O. Murchu, and E. Chien. W32.stuxnet dossier. White paper, Symantec Corp., Security Response, 2011.

[46] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM*, 2009.

[47] D. Freedman and P. Diaconis. On the histogram as a density estimator: L2 theory. *Probability Theory and Related Fields*, 57(4), 1981.

[48] C. Gates and B. Becknel. Host anomalies from network data. In *6th IEEE Systems, Man and Cybernetics Information Assurance Workshop*, 2005.

[49] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang. Measurement and classification of humans and bots in internet chat. In *17th USENIX Security Symposium*, 2008.

[50] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and K. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[51] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[52] J. Grizzard, V. Sharma, C. Nunnery, and B. Kang. Peer-to-peer botnets: Overview and case study. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[53] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *16th USENIX Security Symposium*, 2007.

[54] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *17th USENIX Security Symposium*, August 2008.

[55] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *15th ISOC Network and Distributed System Security Symposium*, February 2008.

[56] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee. Active botnet probing to identify obscure command and control channels. In *26th Annual Computer Security Applications Conference*, 2009.

[57] S. Guha, N. Daswani, and R. Jain. An experimental study of the Skype Peer-to-Peer VoIP system. In *5th International Workshop on Peer-to-Peer Systems*, 2006.

[58] K. Gummadi, R. Dunn, S. Saroiu, and S. Gribble. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *19th ACM Symposium on Operating Systems Principles*, 2003.

[59] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-bot: Improving service availability in the face of botnet attacks. In *6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[60] D. Ha, G. Yan, S. Eidenbenz, and H. Ngo. On the effectiveness of structural detection and defense against P2P-based botnets. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2009.

[61] S. Handelman, S. Stibler, N. Brownlee, and G. Ruth. New attributes for traffic flow measurement. RFC 2724, 1999.

[62] F. Hernandez-Campos, A. B. Nobel, F. D. Smith, and K. Jeffay. Understanding patterns of TCP connection usage with statistical clustering. In *13th IEEE Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005.

[63] P. Holme, B. Kim, C. Yoon, and S. Han. Attack vulnerability of complex networks. *Physical Review E*, 65:056109, 2002.

[64] T. Holz and J. Nazario. As the net churns: Fast-flux botnet observations. In *3rd IEEE International Conference on Malicious and Unwanted Software*, 2008.

[65] T. Holz, C. Gorecki, F. Freiling, and K. Rieck. Measuring and detecting fast-flux service networks. In *15th ISOC Network and Distributed System Security Symposium*, 2008.

[66] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on Storm worm. In *1st USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[67] Honeynet Project. Know your enemy: Fast-flux service networks. Technical report, The Honeynet Project and Research Alliance, 2008.

[68] X. Hu, M. Knysz, and K. Shin. RB-Seeker: auto-detection of redirection botnets. In *16th ISOC Network and Distributed System Security Symposium*, 2009.

[69] L. Huang, X. Nguyen, M. Garofalakis, M. Jordan, A. D. Joseph, and N. Taft. Distributed PCA and network anomaly detection. Technical report, EECS Department, University of California, Berkeley, 2006.

[70] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *7th ACM SIGCOMM Conference on Internet Measurement*, 2007.

[71] M. O. Jackson and B. W. Rogers. Meeting strangers and friends of friends: How random are social networks? *American Economic Review*, 97(3), 2007.

[72] M. Jelasity and V. Bilicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[73] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142, 1998.

[74] I. T. Jolliffe. *Principal Component Analysis*. Spring-Verlag, 1986.

[75] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *2nd USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[76] S. Kandula, R. Chandra, and D. Katabi. What's going on? learning communication rules in edge networks. In *ACM SIGCOMM*, 2008.

[77] B. Kang, E. Chan-Tin, C. Lee, J. Tyra, H. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim. Towards complete node enumeration in a peer-to-peer botnet. In *4th ACM International Symposium on Information, Computer and Communications Security*, 2009.

[78] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *ACM SIGCOMM*, 2005.

[79] V. Karamcheti, D. Geiger, Z. Kedem, and S. M. Muthukrishnan. Detecting malicious network traffic using inverse distributions of packet contents. In *ACM SIGCOMM Workshop on Mining Network Data*, 2005.

[80] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[81] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*. Wiley, 1990.

[82] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, 2004.

[83] S. S. Kim, A. L. N. Reddy, and M. Vannucci. Detecting traffic anomalies using discrete wavelet transform. In *International Conference on Information Networking*, 2004.

[84] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in IP traffic. *IEEE/ACM Transactions on Networking*, 14(6), 2006.

[85] D. Koukis, S. Antonatos, and K. Anagnostakis. On the privacy risks of publishing anonymized IP network traces. In *Communications and Multimedia Security*, 2006.

[86] P. Krapivsky and S. Redner. Organization of growing random networks. *Physical Review E*, 63: 066123, 2001.

[87] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[88] Y. Kulbak and D. Bickson. The eMule protocol specification. Technical report, School of Computer Science and Engineering, The Hebrew University of Jerusalem, 2005.

[89] A. Lakhina, K. Papagiannaki, and M. Crovella. Structural analysis of network traffic flows. In *International Conference on Measurement and Modeling of Computer Systems*, 2004.

[90] J. Li, T. Ehrenkranz, G. Kuenning, and P. Reiher. Simulation and analysis on the resiliency and efficiency of malnets. In *19th IEEE Workshop on Principles of Advanced and Distributed Simulation*, 2005.

[91] X. Li, H. Duan, W. Liu, and J. Wu. The growing model of botnets. In *IEEE International Conference on Green Circuits and Systems*, 2010.

[92] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein. Passive operating system identification from TCP/IP packet headers. In *Workshop on Data Mining for Computer Security*, 2003.

[93] C. Livadas, B. Walsh, D. Lapsley, and T. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security*, 2006.

[94] W. Lu, M. Tavallaee, and A. Ghorbani. Automatic discovery of botnet communities on large-scale communication networks. In *4th ACM International Symposium on Information, Computer and Communications Security*, 2009.

[95] W. Lu, M. Tavallaee, G. Rammidi, and A. Ghorbani. BotCop: An online botnet traffic classifier. In *7th IEEE Annual Communication Networks and Services Research Conference*, 2009.

[96] R. Matei, A. Iamnitchi, and P. Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6: 50–57, 2002.

[97] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[98] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *IEEE International Conference on Network Protocols*, 2002.

[99] A. Moore and K. Papagiannaki. Toward the accurate identification of network identifications. In *Passive and Active Measurement Workshop*, 2005.

[100] C. Moore, G. Ghoshal, and M. Newman. Exact solutions for models of evolving networks with addition and deletion of nodes. *Physical Review E*, 74:036121, 2006.

[101] D. Moore, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. In *10th USENIX Security Symposium*, 2001.

[102] M. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20), 2002.

[103] M. Newman. Mixing patterns in networks. *Physical Review E*, 67:026126, 2003.

[104] M. Newman and J. Park. Why social networks are different from other types of networks. *Physical Review E*, 68:036122, 2003.

[105] J. Newsome, B. Karp, and D. Song. Polygraph: Automatic signature generation for polymorphic worms. In *IEEE Symposium on Security and Privacy*, 2005.

[106] E. Osuna, R. Freund, and F. Girosit. Training support vector machines: an application to face detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997.

[107] J. Padhye and S. Floyd. On inferring TCP behavior. In *ACM SIGCOMM*, 2001.

[108] J. J. Parekh, K. Wang, and S. J. Stolfo. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *ACM SIGCOMM Workshop on Large-scale Attack Defense*, 2006.

[109] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. FluXOR : Detecting and monitoring fast-flux service networks. In *5th Conference on the Detection of Intrusions and Malware and Vulnerability Assessment*, 2008.

[110] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14), 2001.

[111] R. Pastor-Satorras, A. Vazquez, and A. Vespignani. Dynamical and correlation properties of the internet. *Physical Review Letters*, 87(25), 2001.

[112] V. Paxson. Automated packet trace analysis of TCP implementations. In *ACM SIGCOMM*, 1997.

[113] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31 (23-24):2435–2463, 1999.

[114] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems. In *6th IEEE International Conference on Data Mining*, 2006.

[115] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *USENIX Symp. Networked Systems Design and Implementation*, 2010.

[116] P. Porras, H. Saidi, and V. Yegneswaran. A multi-perspective analysis of the Storm (Peacomm) worm. Technical report, Computer Science Laboratory, SRI International, 2007.

[117] P. Porras, H. Saidi, and V. Yegneswaran. An analysis of Conficker's logic and rendezvous points. Technical report, Computer Science Laboratory, SRI International, 2009.

[118] N. Provos and P. Honeyman. Detecting steganographic content on the Internet. In *9th ISOC Network and Distributed System Security Symposium*, 2002.

[119] S. Racine. Analysis of Internet Relay Chat Usage by DDoS Zombies. Master's thesis, Swiss Federal Institute of Technology Zurich, 2004.

[120] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[121] A. Ramachandra, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *14th ACM Conference on Computer and Communications Security*, 2007.

[122] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. *ACM SIGCOMM Computer Communication Review*, 36(4):291–302, 2006.

[123] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[124] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: A statistical signature-based approach to IP classification. In *4th ACM SIGCOMM Conference on Internet measurement*, 2004.

[125] Y. Rubner, C. Tomasi, and L. Guibas. A metric for distributions with applications to image databases. In *IEEE International Conference on Computer Vision*, 1998.

[126] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, 2002.

[127] N. Sarshar and V. Roychowdhury. Scale-free and stable structures in complex ad hoc networks. *Physical Review E*, 69(2):026101, 2004.

[128] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[129] T. Schluessler, S. Goglin, and E. Johnson. Is a bot at the controls? Detecting input data attacks. In *6th ACM SIGCOMM Workshop on Network and Systems Support for Games*, 2007.

[130] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang. A multi-resolution approach for worm detection and containment. In *36th International Conference on Dependable Systems and Networks*, 2006.

[131] U. Shankar and V. Paxson. Active mapping: Resisting NIDS evasion without altering traffic. In *IEEE Symposium on Security and Privacy*, 2003.

[132] G. Sinclair, C. Nunnery, and B. B. Kang. The Waledac protocol: The how and why. In *4th IEEE International Conference on Malicious and Unwanted Software*, 2009.

[133] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *6th USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[134] M. Spiliopoulou, B. Mobasher, and B. Berendt. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS Journal on Computing*, 15(2), 2003.

[135] J. Stewart. This business of malware. *Information Security Technical Report*, 9(2):35–41, 2004.

[136] B. Stock, J. Göebel, M. Engelberth, F. C. Freiling, and T. Holz. Walowdac - analysis of a peer-to-peer botnet. In *European Conference on Computer Network Defense*, 2009.

[137] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *16th ACM Conference on Computer and Communications Security*, 2009.

[138] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. *USENIX ;login*, 32(6), 2007.

[139] S. Strogatz. Exploring complex networks. *Nature*, 410(268), 2001.

[140] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *6th ACM SIGCOMM Conference on Internet Measurement*, 2006.

[141] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and detecting relayed traffic: A case study using Skype. In *25th IEEE International Conference on Computer Communications*, 2006.

[142] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, 1997.

[143] C. Taylor and J. Alves-Foss. NATE - network analysis of anomalous traffic events, a low-cost approach. In *ACM Workshop on New Security Paradigms*, 2001.

[144] J. Terrell, L. Zhang, Z. Zhu, K. Jeffay, H. Shen, A. Nobel, and F. Donelson Smith. Multivariate SVD analyses for network anomaly detection. Poster, ACM SIGCOMM, 2005.

[145] R. Vogt, J. Aycock, and Jr. M. J. Jacobson. Army of botnets. In *14th ISOC Network and Distributed System Security Symposium*, 2007.

[146] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *9th International Symposium on Recent Advances in Intrusion Detection*, 2006.

[147] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[148] D. J. Watts. A simple model of global cascades on random networks. *National Academy of Sciences of the United States of America*, 99(9), 2002.

[149] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393, 1998.

[150] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[151] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *14th European Symposium on Research in Computer Security*, 2009.

[152] Y. Xie, V. Sekar, M. K. Reiter, and H. Zhang. Forensic analysis for epidemic attacks in federated networks. In *14th IEEE International Conference on Network Protocols*, 2006.

[153] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: Behavior models and applications. In *ACM SIGCOMM*, 2005.

[154] R. Xulvi-Brunet and I. Sokolov. Reshuffling scale-free networks: From random to assortative. *Physical Review E*, 70:066102, 2004.

[155] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *5th Conference on the Detection of Intrusions and Malware and Vulnerability Assessment*, 2008.

[156] T.-F. Yen and M. K. Reiter. Are your hosts trading or plotting? Telling P2P file-sharing and bots apart. In *30th International Conference on Distributed Computing Systems*, 2010.

[157] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter. Browser fingerprinting from coarse traffic summaries: Techniques and implications. In *6th Conference on the Detection of Intrusions and Malware and Vulnerability Assessment*, 2009.

[158] J. Yu, Z. Li, J. Hu, F. Liu, and L. Zhou. Using simulation to characterize topology of peer-to-peer botnets. In *IEEE International Conference on Computer Modeling and Simulation*, 2009.

[159] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *IEEE Conference on Local Computer Networks*, 2005.

[160] H. Zhang, A. C. Berg, M. Maire, and J. Malik. SVM-KNN: discriminative nearest neighbor classification for visual category recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

Appendix A

Applications of Browser Fingerprinting to Traffic Deanonymization

In Chapter 3, we developed a browser fingerprinting technique that infers the browser implementation on a remote host using flow records. In addition to enhancing an intrusion detection system to detect a wider range of malware, here we demonstrate a second application of browser identification to the deanonymization of web *sites* in flow records that have been anonymized.

In order to retain the utility of anonymized traffic traces for networking research, IP addresses are typically anonymized in a consistent fashion, i.e., so that the same real IP address is mapped consistently to the same pseudonym in the anonymized dataset. This enables the behaviors of the anonymous web servers to be examined, however, which can sometimes lead to their deanonymization. As a trivial example, the larger number of bytes typically transmitted from the main page of `cnn.com` would enable it to be differentiated from `google.com`. Moreover, since a page retrieval can involve connections to multiple physical servers (e.g., image servers or content distribution networks), Coull et al. [32] also found that the sequential order of the servers contacted to retrieve objects on a webpage can enable websites to be differentiated. While previous works placed emphasis on observing traffic behaviors of the websites [32, 85], to our knowledge, no study has accounted for this behavior as influenced by the particular implementation of their protocol peers, i.e., the browser. In what follows, we show that classifying the browser first can yield a more precise deanonymization of websites.

A.1 Feature Selection

We use the same dataset as that described in Section 3.2. Similar to our browser fingerprinting technique, we extract nine main flow features from each web page retrieval. While these features were previously calculated over all flows in a retrieval, here we calculate these features for all flows *per physical server*, for each of the first five servers contacted. The features are then arranged according to the order that the server was contacted, i.e., for retrieval r, the feature vector is $\{F_{r1}, ..., F_{r5}\}$, where F_{rj} refers to the features derived from the flows to physical server j, for website retrieval r. Breaking down the retrieval features by physical server provides a finer-grained representation of the retrieval and an order to the physical servers, both of which have been utilized in previous website deanonymization efforts (e.g., [32]). Furthermore, to eliminate redundancies and reduce dimensionality, we selected a subset of those features that are most relevant, specifically by computing the correlation of each feature from one day of retrievals to `cnn.com` in the PlanetLab-Native dataset to one day of such retrievals in the CMU dataset. This yielded nine features: the byte and packet counts to and from the first server contacted, and the number of flows to each of the first five servers contacted.

We focus on deanonymizing those websites that are "stable", as judged by their standard deviation for the total number of flows, bytes, and packets, and also those websites that are complex enough, as judged by the total number of flows. It has been previously established [32] that websites with a high variability in their contents (e.g., `espn.com`) or those that are too simple (e.g., `google.com`, `orkut.com`) will typically not be identified accurately. We determine a website to be complex and stable if the average number of flows from the first five servers contacted is greater than one, and the byte and packet counts to and from the first server has a small standard deviation, i.e., within twice the average value. In this way, we narrow down the list of websites that we will attempt to deanonymize in traffic traces to 52 of the top 100 websites in the U.S. according to `alexa.com`.

A.2 Website Classifier

We build our website classifiers using Bayesian belief networks, which have been shown to yield good results [32]. Given a test instance, the classifier outputs a probability for each class, which is the likelihood of the instance belonging to that class, according to the model built from training data. The class with the highest probability is taken as the classification of the test instance. This may not always yield optimal classification, for example, in cases where the probabilities for several classes are close to each other, or when all of the probabilities are small.

To establish some notion of "confidence" on the classification, one way is to let the classifier make a decision only from classes with probabilities greater than a cutoff value, and only when there exist probabilities above the cutoff. Although cases where multiple classes have similar probabilities may still result in ambiguities or misclassifications, the cutoff value can allow the classifier to provide answers based on more confident results, avoiding scenarios where uncertainty (small probabilities) are likely to cause incorrect classifications. The higher the cutoff parameter, the higher the probability of the test instance belonging to its class must be.

From the PlanetLab-QEMU dataset, we group the data by the browser that generated the traffic, as well as a combined group with traffic from all four browsers. This allows us to build four per-browser website classifiers (IE, Firefox, Opera, Safari), and one generic website classifier. The former are each trained on traffic from a single browser type, while the latter is trained on combined traffic from all browsers. In the following, we quantify the benefits of first classifying the browser in website deanonymization by applying these two types of classifier models separately and comparing their results. When testing with the CMU dataset, the browser type for each host is determined by our browser classifier developed in Section 3.3, using a confidence threshold set at 1.30. The per-browser website classifier is then applied to a website retrieval based on the browser determined for the host that performed the retrieval.

For each testing instance, i.e., each website retrieval, the classifier returns the class with the highest probability above the cutoff. If no probability larger than the cutoff exists, the instance is unclassified. Let the classification for retrieval r be $\mathsf{websiteguess}(r)$, and its actual website be $\mathsf{website}(r)$, where $\mathsf{website}(r) = \perp$ if the ground-truth website for retrieval r cannot be determined in the dataset (which only happens in the case of the CMU dataset). Then, the precision and recall are

$$\begin{aligned}
\mathsf{Precision} &= \Pr[\mathsf{website}(r) = s \mid \mathsf{websiteguess}(r) = s \neq \perp] \\
&= \frac{|\{r : \mathsf{websiteguess}(r) = \mathsf{website}(r)\}|}{|\{r : \mathsf{websiteguess}(r) \neq \perp\}|} \\
\mathsf{Recall} &= \Pr[\mathsf{websiteguess}(r) = s \mid \mathsf{website}(r) = s \neq \perp] \\
&= \frac{|\{r : \mathsf{websiteguess}(r) = \mathsf{website}(r)\}|}{|\{r : \mathsf{website}(r) \neq \perp\}|}
\end{aligned}$$

In the following tests, we only report results for cutoff values where the classifier is able to make at least thirty classifications. This is to avoid cases where not enough classifications can be made for the results to be representative.

A.3 Tests on PlanetLab-QEMU Dataset

Similar to the experiments described in Section 3.3.1, we first evaluate the results of website deanonymization under an ideal setting using the PlanetLab-QEMU dataset. In each experiment, the testing data consists of retrievals from one host, while the training data is from all other hosts. We apply each per-browser website classifier to retrievals determined to have been performed with that browser by our classifier in Section 3.3, to generate the per-browser results. We generate results for the generic website classifier by applying that classifier to all retrievals. Our tests are "closed-world", in the sense that only retrievals of the 52 selected websites (see Section A.1) are tested.

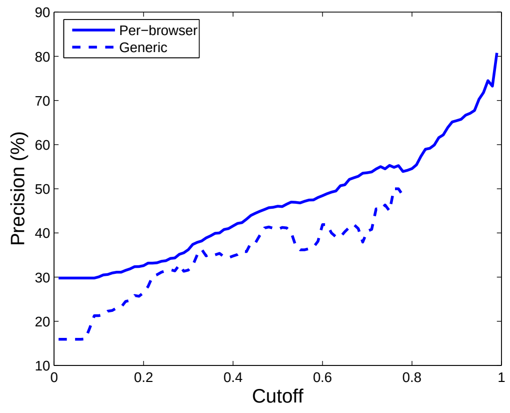


Figure A.1: Website classification precision on the PlanetLab-QEMU dataset.

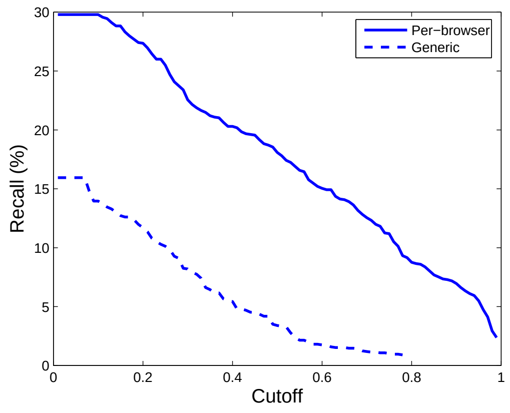


Figure A.2: Website classification recall on the PlanetLab-QEMU dataset.

Figure A.1 and A.2 show the precision and recall from the per-browser and generic website classifiers. Cutoff values range from 0.01 to 0.99, in steps of 0.01. The precision increases with the cutoff, but the recall decreases since some instances are not classified at higher cutoff values. The drops in precision are due to cases where correct classifications that do not have a high enough probability are filtered out by the cutoff value. The generic classifier was not able to classify more than thirty retrievals after the cutoff reaches 0.78, so we do not plot its results for cutoff values greater than 0.78.

To present an alternate view depicting our overall accuracy, let $\mathsf{Precision}(c)$ and $\mathsf{Recall}(c)$ be the precision and recall, respectively, when the cutoff is set to be c. We then define the precision "integral", over the range $[c_{\min}, c_{\max}]$, to be

$$\sum_{c=c_{\min}}^{c_{\max}} \mathsf{Precision}(c)$$

and we define the recall "integral" similarly. $c_{\min}$ and $c_{\max}$ are defined as the endpoints of the range where

Classifier	Precision	Recall
Generic	26.16	5.34
Per-browser	+6.01	+10.93

Table A.1: Comparing the precision and recall integrals on website classification on the PlanetLab-QEMU dataset.

both the per-browser and generic classifiers were able to make enough classifications. The integral is a measure of how the classifier performs across different cutoff values, in that larger integrals show higher precision (or recall) overall. The integral of precision and recall over $[0.01, 0.78]$, in steps of 0.01, are shown in Table A.1, with the generic case serving as baseline.

While website deanonymization remains a challenging problem in practice, we note that the improvement in recall between per-browser and generic classifiers remains significant, across all cutoff values, where the average difference is 14.01% and the maximum difference is 16.11%. On the other hand, the maximum difference in precision for per-browser and generic classifiers is 15.61%.

A.4 Tests on CMU Dataset

To evaluate the impact of first classifying the browser on website deanonymization in a more realistic setting, we turn to the CMU dataset, with the PlanetLab-Native dataset serving as training data. Since the IP addresses are anonymized in the CMU data, we have no direct knowledge of the websites contacted. So, to build ground truth for the classification, we examined information available in the first 64 bytes of each flow payload. Specifically, the "Host" field in HTTP requests are extracted to identify the domain name of the websites. Of the 52 websites targeted for identification, we found only 23 in the CMU dataset in this way, and so used only these retrievals for testing (while the training data still consists of traffic to the 52 websites). Only retrievals from hosts whose ground-truth browser type could be determined were used (see Section 3.3).

For each retrieval to one of the chosen 52 websites (see Section A.1) in the CMU dataset from a Firefox or Opera browser, we classify it using both the appropriate per-browser classifier and the generic website classifier, built using the PlanetLab-Native dataset. The results are shown in Figures A.3 and A.4. In particular, we test the per-browser website classifiers in two scenarios: (i) when our browser classifier from Section 3.3 is applied first, and (ii) when we assume perfect browser classification, i.e., the per-browser website classifier applied to a website retrieval is based on the actual browser that performed that

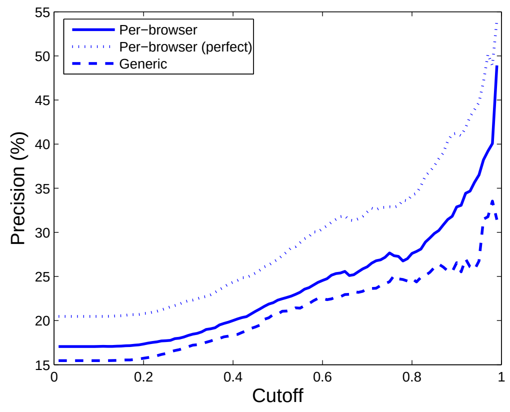


Figure A.3: Website classification precision on the CMU dataset (Train: PlanetLab-Native, Test: CMU).

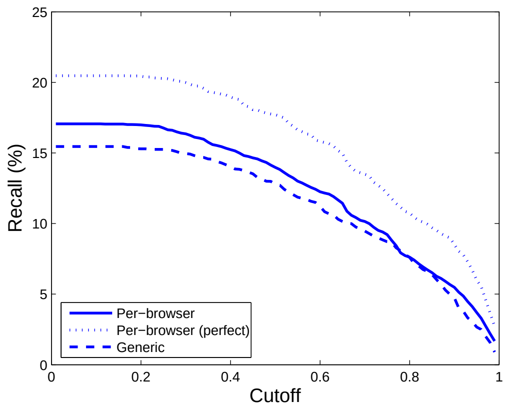


Figure A.4: Website classification recall on the CMU dataset (Train: PlanetLab-Native, Test: CMU).

Classifier	Precision	Recall
Generic	20.53	11.30
Per-browser	+2.73	+1.07
Per-browser (perfect)	+7.93	+4.38

Table A.2: The integral of precision and recall on website classification in the CMU dataset (Train: PlanetLab-Native, Test: CMU).

retrieval, as opposed to the browser determined by our classifier. When our browser classifier is applied, the difference in precision between the per-browser and generic classifiers can reach close to 17% at high cutoff values. Table A.2 shows the integral of precision and recall over cutoff values from 0.01 to 0.99, in steps of 0.01. The results in Figures A.3 and A.4 are calculated across all 52 websites.

However, for an attacker who is only interested in deanonymizing certain websites, such as those listed in Table A.3, a classifier that is able to classify those websites well would be more useful than a general website classifier. For example, the per-browser classifier has a 84.62% precision for `dailymotion.com`, a 27.57% improvement to the generic classifier. These results point out that in live network traffic, classifying the browser first can bring a non-trivial advantage to website deanonymization.

Website	Precision (%)		Recall (%)	
	Per-browser	Generic	Per-browser	Generic
adobe.com	17.59	0.00	9.55	0.00
dailymotion.com	84.62	57.05	50.00	44.95
nytimes.com	21.15	16.26	12.26	9.13
wordpress.com	13.98	0.00	7.15	0.00
yahoo.com	45.52	29.60	29.81	19.78

Table A.3: Comparing the precision and recall between the per-browser and generic website classifiers for the classification of selected websites in the CMU dataset, when our browser classifier from Section 3.3 is applied first (Train: PlanetLab-Native, Test: CMU).