ADVANCES IN MODERN BOTNET UNDERSTANDING AND THE ACCURATE ENUMERATION OF INFECTED HOSTS

by

Christopher Edward Nunnery

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Information Technology

Charlotte

2011

Approved by:

____

Dr. Brent ByungHoon Kang

____

Dr. Mohamed Shehab

____

Dr. Bill Chu

____

Dr. Min Shin

ABSTRACT

CHRISTOPHER EDWARD NUNNERY. Advances in modern botnet understanding and the accurate enumeration of infected hosts. (Under the direction of DR. BRENT BYUNGHOON KANG)

Botnets remain a potent threat due to evolving modern architectures, inadequate remediation methods, and inaccurate measurement techniques. In response, this research exposes the architectures and operations of two advanced botnets, techniques to enumerate infected hosts, and pursues the scientific refinement of infected-host enumeration data by recognizing network structures which distort measurement. This effort is motivated by the desire to reveal botnet behavior and trends for future mitigation, methods to discover infected hosts for remediation in real time and threat assessment, and the need to reveal the inaccuracy in population size estimation when only counting IP addresses. Following an explanation of theoretical enumeration techniques, the architectures, deployment methodologies, and malicious output for the Storm and Waledac botnets are presented. Several tools developed to enumerate these botnets are then assessed in terms of performance and yield. Finally, this study documents methods that were developed to discover the boundaries and impact of NAT and DHCP blocks in network populations along with a footprint measurement based on relative entropy which better describes how uniformly infections communicate through their IP addresses. Population data from the Waledac botnet was used to evaluate these techniques.

ACKNOWLEDGEMENTS

The mentoring and instruction from my advisor, Brent ByungHoon Kang was invaluable for the success of the research contained herein. The assistance of my research colleagues Greg Sinclair, Rhiannon Weaver, Gautam Singaraju and Ben Stock was also paramount in my ability to complete this study. I also wish to thank David Dagon and Chris Lee from the Georgia Institute of Technology, Yongdae Kim from the University of Minnesota and Jullian Grizzard from Johns Hopkins University. Finally, I must express my thanks to the developers of SiLK, a collection of traffic analysis tools developed by CERT which were used extensively to study botnet population data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Continuous increases in malware sophistication have resulted in tuned propagation vectors, formidable defensive technologies, and heightened architectural complexity when rallying behavior is implemented. The underground economy which now largely motivates malware development has acted as an evolutionary catalyst, where profit, efficiency, and infection longevity are encouraged. The end result of this activity is malicious code that spreads tenaciously throughout the Internet, enacts considerable harm, and resists removal or incapacitation.

While there are numerous fundamental types of malware, each deserving of attention with respect to understanding their behaviors and developing remediation techniques, the research presented in this thesis focuses on botnet malware. This focus is motivated by the generally nebulous understanding for modern botnet structures and the accompanying difficulties in estimating the threat a given botnet posses. Thus, this research intends to both refine architectural and behavioral knowledge for advanced botnets and exhibit methods to accurately evaluate their size. Distortion in botnet size estimation can occur through poorly designed enumeration tools which fail to cover all regions or tiers in a botnet, and is commonly introduced by DHCP churn and the presence of NAT devices, which cause over and under-counting, respectively.

The botnets surveyed in this study are Storm and Waledac which can be considered both modern and advanced given their topologies, defensive mechanisms, and communication protocols. Each use architectures and communication protocols with peer-to-peer elements and creative use of DNS. Storm and Waledac, with similar topologies and a probable shared lineage, fast-flux DNS schemes are used.

In this study, following an elucidation of motivations and related work in Chapter 1, methods to enumerate modern botnet architectures are defined and briefly described in terms of advantages, limitations and probable yield in Chapter 3. A deployment path for these tools given required knowledge and yield is then provided. The architectures and behaviors for Storm and Waledac are then presented in Chapter 4, providing illuminating details not found in other literature regarding protocols, defensive measures, and malicious output. This is followed by an attempt to characterize the performance and yield provided by several enumeration techniques when applied to these two botnets in Chapter 5.

This study then provides methods to discover NAT and DHCP boundaries in a botnet population, and proposes an entropy-based method to quantify IP address inflation in Chapter 6. Entropy-based inflation measurements are capable of describing the uniformity of the distribution of infected hosts, and thus differ from pure IP address to unique infection ID ratios. These methods are applied to data obtained from the Waledac botnet.

The sum total of the research presented in this thesis contributes significant advances in modern botnet understanding, the development and evaluation of enumeration technologies, and methods to discover and characterize distortion in population estimation.

CHAPTER 2: HISTORY AND MOTIVATION

This section provides information critical to the understanding of enumeration techniques and the motivation behind the research. Following a discussion of the nature of botnet malware, the topologies of two advanced botnets are described. Fast-flux DNS services are also elucidated, given the newness of this technology, their frequent deployment and use within botnets, and the potential to leverage this service for enumeration. This section concludes with a delineation of related research and publications which complement our work as well as form its underpinnings.

2.1 Botnets

Botnets may be loosely defined as collections of computing systems comprised by a common malware variant which can be instructed en masse by a framework or entity typically known as a "botmaster." The term "bot" is a colloquialism of 'the word "robot" which has garnered use given the autonomous nature of the individual infected nodes participating in these systems. The nature of the node organization and the methods of command distribution is referred to as the botnet's architecture. Botnets have traditionally employed central servers for command and control (C&C) functions, but have recently used more resilient peer-to-peer based topologies. This transition can be considered evolutionary in nature, and consistent with other types of malware in their consistent increases in sophistication.

The large number of compromised computing systems present in any given botnet provide the botmaster with substantial bandwidth, IP address diversity, data processing and storage, and potentially sensitive and valuable local information, such as usernames, passwords, and credit card data. These resources may be easily leveraged and exploited for malicious purposes. Functionally, botnets are most commonly used for unsolicited email dispersion and denial of service attacks. Fast-flux DNS resolution and network scanning activities may also occur by botnets but are generally for self-serving purposes. Network scanning is often seen in botnets that self-propogate with worm-like behavior. Spam email output may also be used for botnet self-propagation.

2.2 Infected Host Enumeration

The enumeration of infected hosts participating in a botnet is essential for threat estimation and often is a prelude to mitigation. Methods to enumerate hosts therefore deserve attention and critique to facilitate further development and refinement.

The knowledge of the IP addresses of infected systems is directly applicable to a number of remediation techniques. Namely, blacklisting of these systems is facilitated. Additionally, enumeration provides insight into the sizes and distribution of individual botnets, allowing the malware defense community to respond proportionately to threats. This study works under the assumption that other, more traditional methods of malware mitigation, such as signature-based anti-virus software are inadequate to remediate entire botnets. The aggressive polymorphism exhibited by all of the malware variants discussed in this study thwarts signature-based detection at a host level, unless current malware signatures are distributed extraordinarily quickly.

By discovering infected hosts participating in botnets through enumeration and limiting their traffic at core Internet routers or other major hops and their interaction with services available on the Internet (specifically mail servers), botnet activity can be more effectively and efficiently suppressed.

Room for improvement exists for current enumeration techniques, and new node-discovery techniques must be developed for novel botnet architectures, which this study intends to demonstrate. The evaluation of existing, improved, and newly developed enumeration methods in terms of their coverage and performance should highlight limitations and weaknesses while characterizing probable yield. This should allow readers to evaluate and improve their own implementations of enumeration methods, following what hopefully is accelerated development and deployment.

2.3 Related Work

Several works have been created in the last decade which have addressed the problem of fundamentally understanding the advent of bot networks and their behavior [26, 6]. Dagon et al. later provided a taxonomy of botnet structures [8]. These studies advanced the understanding of early botnet architectures and provided initial analysis and detection methodologies.

In an exploration of the history of botnets and their architectures, Grizzard et al. presented a case study on an early (January 2007) version of the peer-to-peer based Storm botnet [10]. Stewart, who presented the first exploration in Storm's intricacies [32] furthered the understanding of this botnet by later exposing its hierarchical nature [33]. The architectural characteristics of the decentralized Storm and Nugache botnets were compared by Stover et al. in terms of their command and control strategies, bootstrapping methods, encryption, DNS use, and update mechanisms [36]. The botnets presented in these studies demonstrated the kind of unbridled damage which can occur when a botnet introduces a previously unseen architecture to the Internet, and the inability of the network security community to immediately respond to these radically new and sophisticated botnet topologies.

Kanich et al. [14] and Kreibich et al. [16] explored the spamming model used by the Storm botnet, documenting the quantity of spam sent and the amount which resulted in "conversion," where a recipient executes a binary or makes a purchase on a website using a URL that was distributed from the botnet. Their method involved infiltrating the botnet and rewriting command and control data that passed through their custom nodes.

Bearing a strikingly similar architecture and behavioral characteristics to Storm, the communication protocol and operations of the Waledac botnet may be found various published work [31, 5, 34, 24]. This research helped define the botnet's communication protocol and exposed its vulnerabilities. The work by Sinclair et al. also discussed weaknesses in its architecture which would allow for the possibility of direct remediation and incapacitation of the network [31].

In response to the rapid advances in botnet sophistication primarily the advent of the Storm botnet, several research groups have pursued possible mitigation strategies for these formidable types of botnets. Holz et al. presented an initial attempt to enumerate and mitigate the Storm botnet [12]. For enumeration, Sybil nodes were deployed to passively measure nodes present in the network. A crawler was also used to repeatedly send route requests to also measure the number of present bot nodes. Two mitigation techniques were also proposed, one based on publishing erroneous hashes on the network in bulk and another using Sybil “Eclipse” poisoning. These techniques were mildly successful.

Host-based mitigation technologies have also been crafted to remediate the botnet threat. Bothunter [11] is one such host-based protection method, which employs IDS (Intrusion Detection System) functionality in recognizing established Storm-related network flows. Host-based IDS systems would prove to be effective in mitigating botnet threats for small networks, but like all host-based solutions, scalability is difficult. Further complicating the ease of use and effectiveness of host-based solutions is the need to propagate active and accurate traffic signatures.

Characteristics and behaviors of fast-flux DNS systems in botnets were explored by Nazario et al. [23]. The authors noted that by using active DNS mining, insight into the sizes of botnet fast-flux operations and botnets themselves could be gained. This technique used repeated, not recursive, queries. Based on enumeration data from a network crawler, the authors estimated that approximately 1% of the nodes in the Storm botnet participated in its fast-flux DNS scheme.

Given the flurry of research on the Storm botnet and the infiltration attempts made by several large universities, Kanich et al. presented a study addressing the problem of filtering nebulous traffic from gathered data when crawling this network [15]. Completeness in enumeration for this botnet was pursued by Kang et al. [13] where the authors demonstrated that network crawlers could not discover bot nodes behind firewalls or those residing in NAT networks. This work also studied how many Sybil nodes were needed to achieve complete enumeration through passive monitoring.

Over and under-counting in a botnet population was briefly discussed by Stone-Gross et al. in an exploration of the Torpig botnet [35]. The authors also use the methods in Rajab et al [4] to describe the differences between a "live" size and a total footprint or node count in the aggregate. Kang et al. and Kanich et al. described methods to improve the completeness or accuracy of enumeration efforts for the Storm botnet [13, 15].

Weaver proposed a probabilistic model for the C variant of Conficker to estimate the total population given the scanning behavior of a single node [41], and showed how to use correlation among activity in adjacent net blocks to discover large regions of DCHP activity [42]. Xie et al. created a tool, *UDmap* [44], to process server logs, identify dynamic IP addresses, and track IP inflation among web users. Yu et al. use publicly available statistics on Internet usage and infection longevity to adjust for NAT and DHCP in ranking countries by prevalence of infection [45]. D'Acunto et al. built a Bittorrent-based, P2P system that interacts with STUN servers to categorize UDP traffic behavior of NATs and firewalls on the Internet [7].

Relative entropy, used in the calculation of IP inflation in this study, has been previously used in anomaly detection as a measure of divergence between an observed stochastic traffic profile and a baseline, that is often useful for detecting botnet scanning activity [40, 25].

CHAPTER 3: ENUMERATION TECHNOLOGIES

3.1 Methods

This section provides an overview enumeration techniques to discover nodes participating in botnets with architectures employing peer-to-peer routing. Probable coverage provided by each technique is discussed, in addition to general limitations and advantages specific to each method. These methods, some of which have been explored in related work, were refined during the study of two modern botnets with advanced architectures: Storm and Waledac. In Chapter 5, these methods are comparatively evaluated in terms of coverage and deployment time.

3.1.1 Understanding Bootstrapping

As several of the enumeration techniques described in this section begin by exploiting *bootstrapping* mechanisms available to a botnet, a brief description of fundamental bootstrapping methodologies is warranted. The bootstrapping process occurs early in bot malware execution, and describes the means a node employs to discover one or more nodes already active or recently active in the botnet it wishes to join. Two types of bootstrapping are used: hard-coded *IP addresses* or hard-coded *domain names.* Some malware variants use both techniques. Bot nodes then integrate themselves into a network using these data or peer retrieval points.

IP addresses hardcoded in bot binaries generally belong to reliable systems with high uptime, command and control servers, or recently seen peers in the network which will likely be available when then binaries are executed. The process of frequent binary repacking occurring in propagation campaigns provides the opportunity for botnet operators to inject “fresh” IP addresses in repacked binaries with each new build. A limitation of hard-coded IP addresses for bootstrapping is the possibility that the systems associated with these addresses will become unavailable, particularly if a binary is executed long after distribution. As a result, binaries often use embedded domain names as a redundant bootstrap mechanism.

Hard-coded domains names provide the botnet operator with the ability to change IP addresses for a rallying or bootstrap point. Further, these domains may be a part of a fast-flux DNS scheme, where corresponding A-records for a domain change rapidly. Unlike hardcoded IP addresses, hardcoded domains, particularly those with fast-flux behavior, provide a means for enumeration when domains resolve to IP addresses for bot nodes. Fast-flux DNS functionality and the potential for enumeration is described in Section 3.1.4.

3.1.2 Passive Infiltration-Based Peer-Monitoring

In peer-to-peer based botnets which utilize participating nodes for traffic routing or indexing, enumeration and other intelligence gathering can occur by deploying Sybil nodes throughout the network. In architectures employing distributed hash tables (DHT) as seen in the Storm botnet, monitoring nodes can be deployed evenly throughout the key-space. When a sufficient number of monitoring nodes are deployed throughout an addressing range, one can effectively discover all the bot nodes present in a botnet based on incoming searches and other data the monitoring nodes are tasked with routing.

This enumeration strategy is least disruptive and noticeable when employed in a passive mode. The deployed nodes which gather bot identities can avoid seeking out other nodes. Enumeration occurs via the logging of inbound traffic. This form of enumeration is relatively stealthy as aggressive botnet address traversals are not required and legitimate participating nodes do not need to be interrogated. Enumeration may occur when one is familiar with a popular peer-to-peer protocol and associated network which a botnet uses. Knowledge of specific botnet protocols is not necessarily required, but is needed when botnets use atypical, custom peer-to-peer protocols.

With an *active* enumeration methodology via botnet infiltration, masquerading nodes can aggressively insert their identifying information into peer lists to increase their network presence and traffic within a network and learn more about other botnet participants. This is due to the expectation that more routing and search traffic will be sent through the infiltrating nodes. Active enumeration can also occur by directly querying nodes for their identifying information, but this is generally referred to as network crawling, discussed in the next section. Note that these two enumeration strategies, crawling and enumeration via infiltration can overlap in functionality.

The primary advantage of this enumeration method is the ability to view most of the traffic within a botnet with sufficient Sybil distribution. This is possible even in the face of partitioning, assuming the area(s) of the botnet which use peer-to-peer communication are tasked with data distribution and service hosting for other nodes in the botnet beyond partitioning layers.

The main disadvantages of this technique are the potential necessity for knowledge of communication protocols and the need for a large number of deployed monitoring nodes which are evenly distributed across the network addressing system. Virtualization or running numerous instances of Sybil nodes on a system can nullify this limitation. The possession of large numbers of diverse IP addresses may also be required, as Sybili attack activity may otherwise be noticeable, particularly in botnets with small populations. Finally, node-vetting may be used to thwart mass Sybil node ingress by establishing a trust or reputation system which necessitates that the nodes already participating or a vetting agent approve of a joining node. An additional disadvantage of this technique is the need to expose ones systems to the network, and, in turn, the botnet owners. Direct retaliation can result.

For passive infiltration-based enumeration, botnet coverage is a function of time, node distribution across indexing space, node quantity, and traffic rates, which are dependent on botnet communication protocols and traffic routing behavior. The architecture of a given botnet also plays a role, as partitioning or tiering can limit the ability to discover all nodes regardless of these other characteristics.

In this dissertation, data is exhibited from passive enumeration attempts through infiltration on the Storm and Waledac botnets.

3.1.3 Routing-Table Crawling

In architectures that allow participating nodes to ask one-another for routing-table data or identities of peers in other various formats, enumeration can occur by methodically querying nodes within the network, requesting data which contains peer identities. This enumeration method can be used in architectures with established peer-to-peer protocols traditionally used in benign networks (such as Overnet) as well as custom protocols specific to individual botnets.

As a general rule, diversity should be stressed when requesting routing-table or other peer-identity data. As bot node partitioning may occur in botnet design, where any given node is not omniscient, or aware of the existence of all other nodes within the network, repeatedly querying a single node would yield only addresses for a subset of the total population. Thus, as a general methodology, enumeration by this method should occur by first querying a node, then querying any nodes returned for their peer-lists. Any new nodes discovered in each peer-list request should also be queried. When the known addresses of bot nodes have been queried, the process should be repeated ad infinitum. As partitioning still may occur based on query locality, peer-list requests should originate from a diverse range of IP addresses if possible. The general methodology for this technique is shown in Figure 1.

The main limitation of this technique is the need for knowledge of botnet-specific protocols when custom communication schemes are employed. Means of obtaining identity information for other peers must be determined prior to network enumeration. Accordingly, tools written to “crawl” a given botnet generally must be modified for other newly discovered botnets, and in some cases, different variants of the same botnets due to evolving protocols. The short life-cycles for botnets and their communication protocols exacerbates this limitation. A second potential limitation exists when crawling algorithms are developed to be overly aggressive, revealing the enumeration activity to the botnet operators.

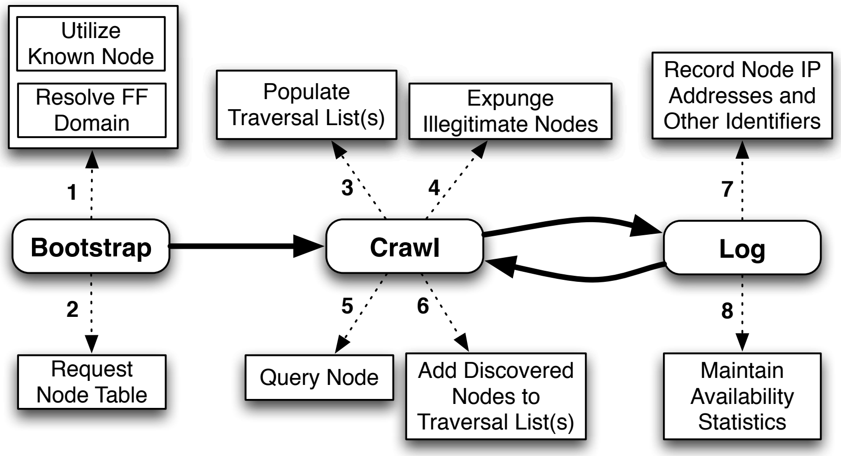

Utilize Known Node
Resolve FF Domain
Populate Traversal List(s)
Expunge Illegitimate Nodes
Record Node IP Addresses and Other Identifiers
1
3
4
7
Bootstrap
Crawl
Log
2
5
6
8
Request Node Table
Query Node
Add Discovered Nodes to Traversal List(s)
Maintain Availability Statistics

Figure 1: A general methodology for enumerating nodes in a botnet with peer-identity retrieval mechanisms. After generating a list of nodes through a bootstrapping process, bot nodes are queried repeatedly for information about other peers. Multi-threaded crawling can occur when traversal lists are divided and distributed. All known nodes may be queried during each crawl cycle, or undesirable nodes may be filtered out using heuristics based on availability and perceived legitimacy.

When crawling algorithms are well-designed, where protocols and other bot behaviors are precisely emulated and use a query rate which is at or below that of legitimate bot nodes, this enumeration strategy can enjoy respectable stealth. A second benefit of this method is the likelihood of obtaining identifying information for nodes beyond IP addresses. Node IDs or hashes may also be discovered along with IP addresses, providing a means to identify unique nodes despite Internet volatility (DHCP churn) and the presence of NAT devices.

Botnet coverage for this enumeration method is a function of time, query rate and returned peer count. Like other enumeration strategies, coverage provided by network crawling can be limited by architectural characteristics such as partitioning. For example, crawling Waledac only reveals nodes in a single tier in the architecture, as the other tiers do not participate in the peer-to-peer communication. The results from the deployment of a network crawler in the Waledac botnet is presented in Chapter 5.

3.1.4 Fast-Flux DNS Exploitation

A brief discussion of fast-flux DNS is warranted in this study, as it is a relatively new technology and understanding how it functions and is deployled within botnets is critical to our enumeration efforts. Domains in *single-flux* fast-flux DNS systems are designed to resolve to a rapidly changing set of IP addresses. In botnets, bot nodes register their IP addresses for domain A records with short TTL values. In *double-flux* systems, nodes register and register themselves as NS records for a DNS zone.

In botnets, a select subset of the participating nodes can be used in fast-flux DNS schemes. In this role, these systems are used as a distributed hosting environment for botnet data, such as binaries or spam-campaign information. Nodes participating in the fast-flux network can either host content directly or forward requests to other computers providing data hosting services. As each node in the fast-flux scheme is capable of delivering content to requesting nodes, considerable resilience is provided. Obfuscation can also be provided by these fast-flux systems. The opaque fast-flux layer can occlude the existence of systems which actually host data as they act as proxies.

Traditional "takedown" efforts which target a single server associated with a domain are no longer effective, as the number of A-records for a fast-flux domain are abound. Moreover, for each resolution attempt, not all A-records associated with a fast-flux domain (for its lifetime) are returned. Take-down attempts for the large and nebulous set of addresses is, as a result, quite difficult.

By exploiting the fast-flux DNS used by botnets, node enumeration is possible. Specifically, domain-names associated with botnets can be repeatedly resolved. IP addresses in returned A-records can then be used as DNS servers themselves in further domain resolution. With short TTL values for fast-flux domain A-records, enumeration can occur rapidly.

The main advantage of this enumeration technique is the ability to enumerate without intimate and precise knowledge of botnet communication protocols. Only the fast-flux domain name(s) associated with a botnet are needed for enumeration to begin. Enumeration with a short lead-time is therefore possible. Further, this node-discovery method is considerably stealthy, as enumeration is simply an exploitation of an external botnet service. The botmaster *intends* for systems outside of the botnet to be able to resolve fast-flux domains and discover participating hosts to allow data retrieval from nodes or bootstrapping. This service, as a result, can generally be exploited without fear of alerting a botnet operator.

Limitations of this node discovery method include the inability to discover nodes not participating in the fast-flux scheme and the lack of desirable identifying information given in domain resolution beyond simple IP addresses. In our research we have found that nodes participating in botnet fast-flux schemes do not constitute a majority of the total population. The lack of identifying information such as peer hashes is also unfortunate, given the potential for false-positives in mitigation and significant distortion in population estimation.

Coverage for this technique is a function of time, domain resolution rate, TTL values for A-records, and the percentage of total population participating in the fast-flux DNS scheme. Partitioning can affect coverage also, if multiple domains are used and a given domain only resolves to a fraction of the nodes in the complete fast-flux scheme. In Chapter 5, this efficacy of this method, and its coverage is compared to a network crawler for the Waledac botnet.

3.1.5 DNS Sinkholes

Our final two enumeration techniques proposed in this study originated during our research on the Confiker worm and its associated botnet. While much of the botnet architecture used by this malware variant is nebulous, two facets of its operations allow researchers to enumerate its participating systems.

For current data discovery, Conficker first utilizes a hard-coded algorithm to generate numerous domain names based on dates and then sends queries to the IP addresses associated with these domain names. These domains, which appear random, are registered by the authors of this malware prior to their appearance in output of the domain-generator algorithm.

Based on this behavior, it is possible for malware researchers to register these domains yet further in advance, prior to their registration by the malware authors. Monitoring systems may then be deployed with the IP addresses of the *A-records* of these domains, and inbound traffic may be logged. As the domains generated by the Conficker algorithm are considerably awkward in terms of composition (neither pronounceable nor originating in any modern languages), one can make the assumption that data-requests inbound to the systems associated with those domains do indeed originate from systems infected with Conficker. The domain names were likely not accidentally typed by a user attempting to navigate to a website or request other non-malicious Internet-based services. With this monitoring, enumeration can occur.

The primary advantage of this technique is the ability to capture all communication intended for hosts associated with bot domains without knowledge of botnet protocol details. Only knowledge of botnet domains is needed. This would constitute either a static set of domains or possession of the algorithm used to generate pseudo-random domains. A limitation is a potentially slow discovery rate given unpredictable sinkhole contact events, which would be dependent on botnet communication schemes.

Botnet coverage provided by this technique is a function of time, the percentage of total domains "monitored" out of all potential domains, and inbound traffic rates.

3.1.6 Darkspace Monitoring

A second enumeration technique which arose during our study of Conficker is based on the worm's use of random Internet scans in IPv4 address space. The scanning logic in Confiker does not exclude IPv4 addresses which are unreachable, those falling in *darkspace* ranges. By deploying monitoring systems designed to receive traffic destined to addresses in these ranges, we can observe Conficker activity. With time, a significant number of computers infected with Conficker can be enumerated.

Though this enumeration method was crafted as an effective means of monitoring Conficker, it is applicable to other forms of malware which scan Internet addresses ranges, As peer-to-peer botnets can employ this behavior to discover nodes in purely decentralized architectures, we include darkspace monitoring as a general enumeration technique for these types of advanced peer-to-peer botnets.

The primary advantage of this node discovery method is the provision of complete stealth. A limitation is the need for large regions of darkspace to catch a significant portion of botnet traffic. Coverage is a function of time, size of monitored IP address space, bot scanning selection algorithms, and scanning rates. Coverage would also depend on botnet architectural characteristics, where only nodes which engage in scanning activities may be discovered.

3.1.7 Local Execution Monitoring

One of the most basic forms of botnet enumeration can occur through the monitoring of a honeypot or a system intentionally infected with a bot binary. This allows one to discover nodes participating in a botnet via the bootstrapping process and various types of information exchanges with other nodes.

The primary limitation of this method is difficulty in scalability and the limited coverage provided. This method does not scale well, considering that a single infected node likely does not interact with a significant percentage of the total bot population during routine execution. It is also unlikely that a single node would otherwise be aware of the total population, unless full peer lists are distributed to all participating nodes, as per the case with the Mayday botnet. This technique can scale with multiple infected hosts, but these systems would likely need to be virtualized. A benefit of this method relates to the botnet communication protocol, in that it may be immediately and precisely implemented without reverse-engineering or observation followed by programming.

The coverage provided by this technique is a function of time, population visible by an individual node, and the number of infected systems deployed.

3.1.8 Sacrificial Attack Recipients

A rudimentary and potentially harmful method to enumerate nodes participating in a botnet consists of intentionally provoking a denial of service attack on oneself from a given botnet. Such an attack may be instigated by aggressively crawling a botnet or otherwise performing intrusive activities which a botnet operator views as undesirable tampering. This enumeration method related to darkspace monitoring and sinkhole-based techniques.

This enumeration technique is exceedingly simplistic, but requires an adequate network infrastructure to monitor and log inbound attack traffic. Additionally, it is not guaranteed that systems participating in the attack belong to the botnet one desires to enumerate. If botnet operators are not inclined to defend themselves with their own resources (i.e. the nodes within their own botnets), other botnets may be leased to perform distributed denial of service attacks. If a botnet one desires to enumerate does, in fact, perform a denial of service attack, not all of the nodes in the botnet may participate.

Further, such attacks may be difficult to incite, and the exposure provided by the mechanisms to trigger the attacks may not be desirable. Finally, attacks performed by botnets may not precisely target the systems you intended to become recipients of these attacks. Routers and other systems tied to an organization a researcher belongs to may be affected.

Given the limitations of this enumeration method, coverage can be considered fundamentally stochastic. However, coverage could be determined if one knew the participation percentage of the the nodes in a botnet engaging in the attack, the duration of the attack, and the rate of the attack.

3.2 Performance Evaluation Criteria

This section briefly describes evaluation criteria which can be used to evaluate the performance of enumeration tools: accuracy, robustness, and stealthiness.

3.2.1 Accuracy

Accuracy in botnet enumeration refers to correctly distinguish bot nodes from innocent network participants or fellow researchers. This is particularly important when a botnet co-opts an existing innocuous peer-to-peer network. Accuracy may also refer to the ability of a tool or methodology to enumerate infections rather than IP addresses. When possible, unique identifiers specific to infected hosts should be retained in addition to network information, as IP addresses are unreliable due to DHCP churn and NAT devices.

3.2.2 Robustness

Botnet monitoring and measurement tools or methodologies should be robust and resistant to protocol changes. The degree of robustness can result in an enumeration method remaining effective for a single malware family experiencing protocol changes or across numerous malware families with vastly different behavior. Regretfully, as this study demonstrates, tools which emulate botnet protocols are the most effective in terms of completely enumerating a bot network are the least robust.

3.2.3 Stealthiness

Finally, stealthiness must be emphasized when designing enumeration tools, as combative measures are often found in botnets where nodes which do not accurately emulate legitimate infections are expunged or attacked. The absence of stealthiness in tool design can also apply evolutionary pressure to malware, acting as a catalyzing influences in malware design where additional defensive technologies might be incorporated in future revisions of a malware family.

CHAPTER 4: ADVANCED BOTNET ARCHITECTURES

This chapter delineates two advanced botnets which have attained notoriety in the information security community given their novel architectures, formidable defenses, and unique daily operations. Achieving comprehensive understanding of these architectures is critical for the development and refinement of the node-discover techniques described in this study. The Storm botnet is presented in Section 4.1. The Waledac botnet is documented in Section 4.2.

4.1 The Storm Architecture

The Storm botnet, which was first discovered in January of 2007, utilized peer-to-peer communication in its design, which marked a radical departure from centralized IRC-based architectures. While Storm was not the first botnet to employ a partially decentralized design, it was perhaps the most successful to date, given its estimated size, spam-output, and longevity. Despite its unique architecture, the behavior of Storm as a botnet is similar to that seen in more familiar botnet topologies. Through its life, the botnets was used for the distribution of spam emails and distributed denial of service attacks against a variety of targets, most commonly spam blacklisting services and anti-malware researchers [32, 14].

While Storm was initially entirely peer-to-peer based, a tiered topology was later introduced, with only the lower three layers engaging in a Distributed Hash Table

(DHT) based peer-to-peer communication scheme. This final hybrid architecture is shown in Figure 2. The vernacular used to describe the layers of this network was originally proposed by Stewart [33]. This bodysection describes the experiments and research conducted to discern the botnet's structure and behavior followed by a delineation of it's communication protocol.

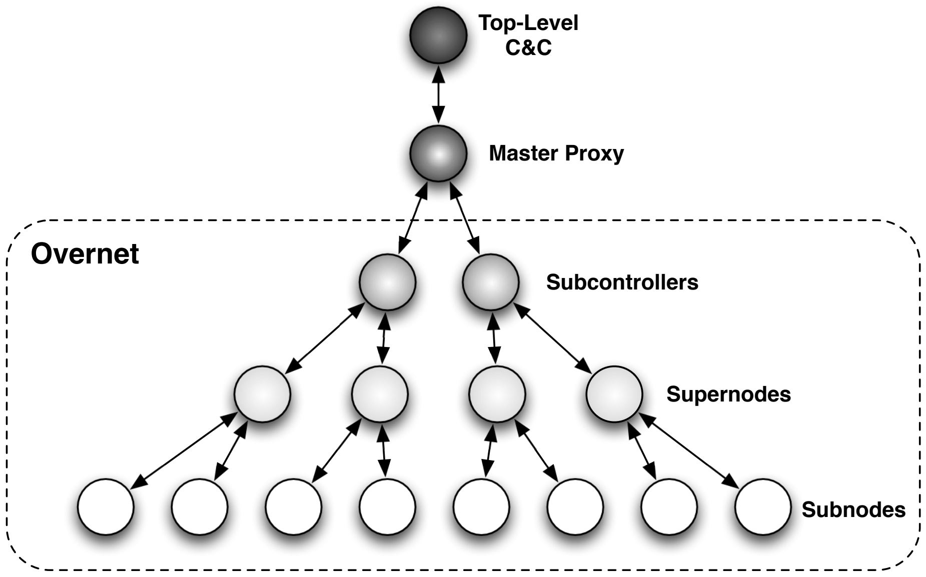


Figure 2: The Storm botnet architecture. A hybrid architecture featuring DHT-based P2P communication with an overall tiered structure.

4.1.1 Analysis Methodology

The structure and behavior of the Storm botnet were partly discovered by intentionally infecting four computers running Windows XP SP2 using Storm binaries. These are referred to these as *bare-metal* systems, as the execution environment was native and not virtualized. Possible detection of VMWare components by Storm binaries influences this decision. Two of these systems resided in a university network and two were connected using a residential ISP.

A Linux-based server, acting as a bridge, was been placed between these machines and the Internet allowed for the capture of incoming and outgoing traffic as well as blocking of certain types of malicious traffic (e.g., SMTP used in bot email spamming). In some instances, malicious traffic may be redirected to a sink-hole. In the research presented in this dissertation, experiments were designed to avoid significantly contributing to the malicious output of this botnet.

Throughout the experiment, secondary injection updates, including adaptation to a new XOR encrypted communication channel, occurred without issue. The network traffic generated by the bare-metal honeypots running real Storm binaries provides a ground-truth for the botnet's activities. Unfortunately, this does not allow for the discovery and analysis of systems deployed by the botnet operator for command and control functions.

Perceived functionality and behaviors viewed in traffic were confirmed and more extensively explored through the reverse-engineering of Storm binaries. This also provided a means to discover the bot's hash-generation algorithm, encryption keys, and other behavior that could not be discerned through non-static analysis techniques.

4.1.2 Communication Protocol

Storm's communication protocol used an implementation of Overnet. Thus, this section briefly describes the functionality of the original Overnet protocol, then delineates the modifications used within Storm.

4.1.2.1 Overnet

Initial exploration of network traces from Storm binaries indicated the use of the Kademlia protocol and participation in the Overnet file-sharing network. Overnet was originally designed for file sharing and implements the Kademlia protocol [21]. The network uses *Distributed Hash Tables* (DHT) to index files and facilitate searches. Within this network there is no hierarchy. Each node participates equally in routing search traffic and possesses an arbitrarily generated 128-bit identifier. A ring architecture is essentially formed which encompasses the complete ID key-space.

The DHT interface allows a user to publish [`key, value`] bindings, where *keys* are constrained to the ID-space within the network and *values* are arbitrary strings. Searches are performed by computing the cryptographic hash (*MD4*) of a keyword; this hash is bound to the cryptographic hash of a file, which is in turn bound to a string containing metadata about the file. When joining the network peers publish information about files they own. Other peers determined to be *close* in terms of key-space to the hash of the file are tasked with indexing this data, which includes the name, length, and the IP addresses of peers that posses those files. In this network "close" does not refer to physical proximity but rather distance as calculated by a bitwise XOR of two hash IDs.

Each node in the network also maintains a large list of neighboring peers which effectively functions as the node's routing table. A Searches for a given key κ are accomplished using "iterative prefix matching." Starting from the routing table, the peer repeatedly asks the $\alpha = 3$ nodes it knows with IDs closest to κ for their α nodes closest to κ, which are then added to the list of known peers, until it finds a *replica root*, or a node closest to the desired key. These nodes may be found with logarithmic efficiency.

The types of messages in the Overnet protocol are delineated in Appendix A.2.2.

4.1.2.2 Storm Protocol

Storm originally co-opted Overnet from January to October of 2007, and connected to this network used the same ID space, message types, and semantics as legitimate Overnet clients. From October 2007 until the botnet's demise in late 2008, however, Storm bots joined an "encrypted" network that follows the same set of protocols as the Overnet but encrypts packets at the application level using a simple XOR cipher with a 320-bit key. Since the packets were encrypted, this network no longer interacted with Overnet. While this allowed Storm nodes to longer be tasked with routing traffic associated with legitimate peer-to-peer network activity in Overnet, it also facilitated the accurate enumeration of infected nodes, as the newer encrypted network contained only legitimate Storm nodes and researchers.

Several idiosyncrasies exist in the Storm implementation of the Overnet protocol which distinguish it from legitimate P2P traffic. The number of search messages sent by a legitimate client is 3 for any given search. In Storm, however, 20 were observed. This was most likely an attempt by the bot to increase the odds and speed of search results. Publish messages sent by the bot also did not contain any search bindings. Typically, publish messages contain 2 hashes in addition to strings. This created Storm publish packets which were fixed at 36 bytes.

4.1.2.3 Network Bootstrapping and Locating Data

A bot began by sending *OvernetPublicize* messages to IP addresses and ports hard-coded in the binary in order to find a live node participating in the network. Upon receiving an *OvernetPublicizeAck*, an incoming bot continued to talk to this node to build up a large number of IP addresses in its routing table. While these peer identities were stored in memory, periodically, a small number of peer IP addresses were also written to disk in `spooldr.ini`. Peer data in this file was formatted as `<32 CharacterHashID>=<8CharacterIPAddress><4CharacterPort>00`, with the IP address and port in a hexadecimal format. An excerpt of a Storm bootstrap file is shown in Figure 3. When a bot has a sufficiently large number of routing table entries, it uses *OvernetPublicize* messages and performs searches for its own ID to maintain a list of active peers.

```
[config]
ID=930892018
[local]
uport=32932
[peers]
0000CE0DFF15B0147F656014084C9E1F=7AA517D0308200
0100DF1E00362069A4001B189D259177=5D50A6A76E9D00
0200BC6FD7649D3535182302EC1DC551=62C7ADA5184400
0300D0245D135C365950A24C793F8B1C=C4C026C00D6D00
...
...
```

Figure 3: Excerpt from a sample Storm bootstrap file.

To locate data in the network a bot uses iterative parallel routing by repeatedly sending *OvernetSearch* messages to get closer to a target hash. In each hop, it sends 20 *OvernetSearch* messages and, upon receiving sufficient *OvernetSearchReply* messages, it chooses another 20 nodes closer to a key. When the search is routed to a potential root node, a *OvernetGetSearchResults* request is sent. The *OvernetSearchResults* messages sent as a response initially contained information pertinent to malicious campaigns, such as sending spam or participating in denial of service attacks. In later versions of the botnet, this search functionality served to help a bot locate IP addresses of nodes in the hierarchical architecture.

Target hashes can be one of 32 unique strings generated daily. The hash-generation algorithm is hard-coded in each bot binary, and uses the current data and a random integer between 0 and 31 as input. To ensure that the correct hashes are being generated, Storm infections perform an `NTP` query to synchronize the local system clock with the correct date. To monitor the botnet, the activity associated with these hashes can be recorded.

4.1.3 Tiered Structure

In early versions of Storm, infected hosts participated equally in the malicious output of the botnet, primarily spam dispersion, but following the introduction of the tiered structure to the network, bots were relegated to two discrete layers which distinguished their roles. These layers have been described as *Subnodes* and *Supernodes* [33] or alternatively *Worker bots* and *Proxy bots* [14]. Supernodes were externally available, while nodes behind NAT or firewall devices became Subnodes. This decision was made after a test to determine if a node could be reached externally.

Supernodes were activated when they received an RSA-encrypted packet known as a "Breath of Life." These packets were sent by nodes higher in the botnet hierarchy which have been called Subcontrollers [33] and contained a lists of these same hosts which allowed the recipient of the BoL packet to act as proxies for the botmaster systems. Communication between Subnodes and Supernodes was HTTP-based and `zlib` [28] compressed.

At least one additional tier existed above these layers which provided tasks to the botnet and logged status reports. Servers residing in tiers above the layers populated with infected-hosts in Storm were deployed by the operators of the botnet and could not be inspected to confirm their existence or discern their configurations.

4.1.4 Infected-Host Activities

Supernodes, which were publicly accessible, were used in the botnet's HTTP proxy system and fast-flux DNS scheme. These nodes listened for HTTP traffic on port 80, specifically requests for bot binaries generated when Internet users clicked on URLs in emails distributed during malware propagation campaigns. Requests for these binaries were forwarded up to the Subcontroller tier, which returned the requested binary. It is unknown whether hosts in the Subcontrollers tier actually hosted these binaries or if additional request forwarding was performed. Once a Supernode receives a binary, it responds to the original binary request. From a user's perspective, the requested binary appears to be hosted on the Supernode. By computing MD5 hash sums for gathered binaries, it was observed that these executables were not static. Though the functionality of these binaries is consistent, their MD5 signatures change every 10 minutes.

Subnodes, the bots in the lowest layer of the hierarchical architecture, were tasked with sending spam and participating in denial of service attacks. Spam emails were only sent, however, if an initial test to see if the bot can connect to an external SMTP server was successful. Subnodes communicate through Supernodes, which are discovered using the Overnet search process described above. Supernodes act as intermediaries (proxies) between the Subnodes and Subcontrollers. As the Subnodes in the network request spam campaign data, this can be considered a *pull* rather than *push* command and control scheme.

Subnodes requested three types of data over TCP prior to participating in the spam distribution campaign: templates, raw-text dictionaries, and a list of email addresses to be used as mail recipients. The raw-text dictionaries were used to complete fields in the templates and produce more dynamic spam less likely to be blocked by filters in-transit. This can be referred to as spam *polymorphism.* After unique messages are crafted for each recipient in the delivery list, Subnodes send these messages to the servers found using the MX records. After completing the spam workloads, Subnodes would send detailed statistics related to delivery success back to the botmaster via Supernodes.

4.1.5 Demise

The Storm botnet thrived from January 2007 until September 2008 when it inexplicably stopped functioning. Numerous mitigation efforts from several universities and private-sector companies were ongoing at the time, but it is unknown whether the botnet met its end due to that activity or intentional destruction by the botnet operator(s). The emergence of a new, highly similar botnet, however, in December of 2008 suggests the latter of these two possibilities may be true. Assuming the botnets are related, which this research proposes, it is probable that Waledac was designed as a replacement.

4.2 The Waledac Architecture

Featuring a highly similar topology to Storm, the Waledac botnet emerged in late 2008. Waledac also employes a composite architecture which blends peer-to-peer communication with an overall tiered structure. The binaries which infected hosts and formed this botnet were spread via email. The botnet primarily functioned to send spam and harvest sensitive data from bot participants. This chapter documents this botnet's structure, communication protocol, malicious output, and the deployment specifics of the command and control servers.

4.2.1 Layers in the Tiered Topology

The determination of Waledac's structure and many of its behaviors was achieved through reverse engineering of bot binaries, binary execution, and live network exploration. Discovering the configuration and behavior of the nebulous systems deployed by the botmaster required cooperation of the ISPs hosting these systems. Two ISPs in the Netherlands provided network traces and file-system artifacts from nodes in Waledac's top two tiers. The botnet's communication protocol was determined through reverse-engineering and analysis of network traffic from Waledac-infected hosts.

While active, the Waledac botnet was composed of four tiers, each containing nodes with clearly defined roles. Infected hosts were relegated to the lower two tiers, while systems deployed by the operators of the botnet were hosted in the top two layers. This topology is shown in Figure 4. Unlike Storm, a complex peer network which comprises multiple tiers is not found; peer-to-peer communication is relegated to a single tier. Comparatively, Waledac is more refined and simple, the result of a devolutionary process.

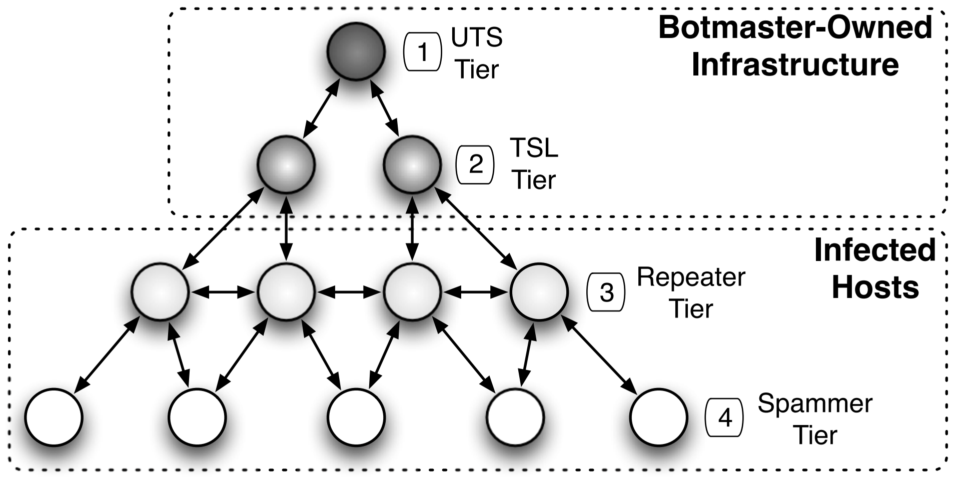


Figure 4: The hierarchical topology of Waledac. Peer-to-peer communication is relegated to a single tier.

4.2.1.1 Tier 1: UTS

The highest layer in Waledac contained a single node throughout its lifetime, using the IP address 85.17.143.66. This system, hosted in the Netherlands, is referred to as the *UTS*, or Upper Tier Server. This system functioned as the primary C&C server for the botnet, performing the following functions:

- Task Storage
- Bootstrap List Hosting
- Binary Hosting (For Spam and Propagation Campaigns)
- Infected Host Monitoring, Heath Tests / Auditing
- Botnet Log Repository
- Interface for Affiliates

By analyzing network traces from this system, its operating system could be determined. As part of the request to the `yum` repository found in Linux distribution based on Red Hat, `yum` sent out a XML request containing the operating system and platform. The UTS server under observation sent a standard HTTP request containing `GET /pub/centos/5.3/os/x86_64/repodata/repomd.xml`. This indicates that the UTS server was based on CentOS 5.3 running on a 64-bit platform.

By inspecting fragments of file-system data, it was determined that this server used 97 PHP files, 75 bash scripts, and numerous flat text files to perform its daily operations. A central database for configuration data, such as *Oracle* or *MySQL* was not employed. The main PHP files which interfaces with the lower tiers of the network is called `main.php`. This script responded to queries from infected-hosts in the network and dispatched an appropriate handler for the type of request. Automated tasks on the server were performed with `while 1` loops and `sleep` commands rather than `cron` jobs.

To interact with this server, the botnet operator relied on a command-line interface. A graphical interface does exist but is not interactive; it provides visualization for botnet statistics using RRD. This monitoring interface is described in Section 4.2.6.

The UTS node periodically interacted with several 3rd-party servers external to Waledac's tiered topology. These external servers provided binary repacking services, URL data for spamming campaigns, and *RogueAV* software to be installed on infected hosts. These activities are described in Sections 4.2.4 and 4.2.5.

4.2.1.2 Tier 2: TSL

The TSL layer, also containing systems deployed by the botnet operators, is the last victim exposed tier and the first obfuscated tier in the Waledac infrastructure. This layer is named after the network's distribution of the identities of these systems in a format dubbed "The Server List."

From the perspective of nodes below it, this tier functions as a simple obfuscation layer, occluding the location and existence of the UTS tier. Forensic analysis of these systems, however, reveals that the TSL tier also plays a role in the botnet's production of spam. The use of this tier in Waledac's malicious output is discussed in section 4.2.4. While the UTS layer contained only a single node, the TSL hosted, at a given time, between 5 and 7 servers. These severs constantly revolved through the botnet's lifetime, as the botnet operator would bring new servers online when TSL systems were disabled by their ISPs.

By examining the *Kickstart* configuration file, it was determined that these servers contain an absolutely minimal base image install (referred to in the configuration as simply @`core`). This stripped down install reduces the amount of work the botmaster must do to secure the server when removing unnecessary services.

Inspecting the .bash_history file for each of the TSL images, it was possible to determine the approximate order in which services were installed and executed during the creation of the TSL server by the botmaster. Following the installation of `mc`, the botmaster installed several standard services and applications such as the Network Time Protocol (`ntp`) daemon, the DNS server BIND, PHP, OpenVPN, BZip2, and the nginx [38] proxy.

The botmaster used two different approaches for obtaining the necessary services and applications to bring a TSL online. The first approach relied on the standard `yum` application. This approach allowed the botmaster to use the database of precompiled applications available in the CentOS software repository. The second approach, the approach most heavily relied upon, involved having a prepackaged collection of software installation archives that require individual installation and compilation. As a defensive mechanism intended to make data forensics more difficult should a TSL system become captured, a custom script was found on this system which deleted incriminating logs in `/var/log/` every hour.

```
/root/pack:
-rw-r--r-- 1 root root   18 Jul 10 09:13 PERSONAL_IPTABLES!
-rwxr-xr-x 1 root root  147 Jul 23 09:20 clean_log.sh
-rwxr-xr-x 1 root root  185 Dec 15  2008 do.sh
-rw-r--r-- 1 root root   41 Dec 15  2008 i18n
-rwxr-xr-x 1 root root 3689 Jul 23 09:20 iptables
-rw-r--r-- 1 root root 4123 Dec 17  2008 nginx.conf
-rw-r--r-- 1 root root   57 Dec 15  2008 rc.local
-rwxr-xr-x 1 root root   81 Dec 22  2008 screen.sh
-rwxr-xr-x 1 root root   84 Dec 15  2008 time.sh

/root/src:
-rw-r--r--  1 root root   6724683 Jun 12 02:23 bind-9.6.1.tar.gz
-rw-r--r--  1 root root    895713 Jun 15 19:59 dante-1.1.19.tar.gz
-rw-r--r--  1 root root    555631 Jun 23 21:40 eaccelerator-0.9.5.3.tar.bz2
-rw-r--r--  1 root root    414870 Jun 13 16:45 htop-0.8.1.tar.gz
-rw-r--r--  1 root root    428061 Jun 23 14:05 htop-0.8.3.tar.gz
-rw-r--r--  1 root root    524667 Nov 27  2008 nginx-0.6.34.tar.gz
-rw-r--r--  1 root root    593586 May 25 10:00 nginx-0.7.59.tar.gz
-rw-r--r--  1 root root    595557 Jul 13 11:48 nginx-0.8.5.tar.gz
-rw-r--r--  1 root root   1168513 Sep  5  2008 pcre-7.8.tar.gz
-rw-r--r--  1 root root  11433921 Jun 17 12:43 php-5.2.10.tar.gz
-rw-r--r--  1 root root  12427411 Jun 13 16:45 php-5.2.8.tar.gz
-rw-r--r--  1 root root  13239065 Jun 29 21:36 php-5.3.0.tar.gz
-rw-r--r--  1 root root    327057 Jun 23 21:40 proxychains-3.1.tar.gz
-rw-r--r--  1 root root     83928 Jun 16 09:52 tsocks-1.8beta5.tar.gz
```

Figure 5: TSL *pack* and *src* directories as observed on a TSL server. These directories contain the necessary services and configuration files to deploy a TSL server.

The network identities of TSL systems are encrypted and distributed to nodes in the next layer down, Repeaters, shortly after these systems join the botnet. While the peer-lists the Repeater nodes exchange are unauthenticated, the list of TSL IP addresses is signed using a public/private key signature.

4.2.1.3 Tier 3: Repeater

The *Repeater* tier contains systems infected with Waledac binaries which feature non-private IP addresses, and has the distinction of being the highest tier which contains infected-hosts and the only tier which uses peer-to-peer routing. This tier is similar to the *Supernode* layer in the Storm botnet. Nodes in the Repeater tier are aware of one another as the result of the exchange of XML-formatted peer lists. The specifics of this peer communication are discussed in Section 4.2.3.3. Peer lists are initially seeded by querying a fast-flux domain, which points to a Repeater node. Repeater nodes can serve as HTTP proxies, SOCKS proxies, and DNS servers, participating in the fast-flux botnet infrastructure. These nodes also harvest local emails and use a *library* to sniff network traffic.

When sending data to the C&C server, Repeaters communicate through other Repeaters. In terms of routing, data is sent laterally to a neighboring Repeater node before it is sent up. Spammer nodes in the tier immediately below this layer also rely on Repeaters to send data to the UTS tier. Given that they are publicly accessible and are required for routing data, Repeater nodes are considerable more valuable than the infected hosts in the Spammer tier.

4.2.1.4 Tier 4: Spammer

The lowest layer in Waledac is populated with *Spammers*, named after their primary functionality and the use of *S* in Waledac's own C&C logs. Incoming nodes are relegated to this tier when the possess private IP addresses as dictated by the RFC 1918 [27] specifications. Nodes in this tier sent unauthenticated spam, participated in denial-of-service attacks, and harvested local sensitive network and file-system data. Spammers communicated through Repeaters for all types of communication.

A *pcap* library was used to sniff local network traffic, looking for SMTP credentials. The file-system harvesting activity searched for email addresses, locating them by looking for the '@' character, then scanning backwards until the first non-printable ASCII character was reached, followed by a forward search using the same methodology. A limit of 256 characters is is used for both searches. Top Level Domains (TLDs) less than 2 characters and usernames less than 4 characters were rejected. Discovered address were stored internally for later transmission to the UTS. Probing of the UTS, described in Section 4.2.6 revealed 14 GB of email addresses. Only files with certain extensions were scanned. The email harvesting routine ignored files with the following extensions: `avi`, `mov`, `wmv`, `mp3`, `wave`, `wav`, `wma`, `ogg`, `vob`, `jpg`, `jpeg`, `gif`, `bmp`, `exe`, `dll`, `ocx`, `class`, `msi`, `zip`, `rar`, `jar`, `hxw`, `hxh`, `hxn`, and `hxd`.

4.2.2 Trans-Tier Command Marshaling

By default, the *ngnix.conf* file, seen in Figure 6, contains a simple set of proxy transformations. The primary function of the proxy transformations is the translation of requests from the public side of the TSL tier to a format acceptable to the higher tiers of the botnet. These transformations focus primarily on ensuring that the request originated from within the Repeater tier of the botnet, as indicated by the user-agent field of the HTTP request containing the string `LMK`. With three exceptions (`/pr/`, `/lm/`, and `/tds/`), the proxy will return a HTTP 404 error code if the user-agent does not contain the `LMK` substring. This effectively weeds out non-Repeater tier originating requests while at the same time preventing additional work for the UTS tier.

The three exceptions to the `LMK` rule relate to traffic originating from outside of the Repeater tier. These exceptions establish the fact that what was originally considered a simple proxy tier is actually an entry point for third party access. The exceptions allow third party actors (such as affiliates) to interface with the Waledac botnet in order to facilitate the underground commerce the Waledac botnet generates. The `/pr/` exception allows the botmaster to transfer content between the botmaster-controlled tiers (TSL and UTS) without significant overhead and provides a means for phishing webpages to serve content such as graphics and executables. This data revealed in this configuration file allowed for the direct interaction with the UTS in probing experiments, and the retrieval of back-end data, including email addresses, raw log files, and other various configuration files.

4.2.3 Communication Protocol

This section describes the nuances of Waledac's communication protocol, determined through reverse-engineering of bot binaries and the analysis of network traffic from Spammer and Repeater nodes, which were run in a virtual environment throughout Waledac's lifetime (December 2008 - February 2010).

4.2.3.1 Bootstrapping

A new infection joining the Waledac network relies on two mechanisms to discover active nodes already participating in the botnet. First, a new infection utilized a hardcoded list of IP addresses. Should the incoming node fail to find an active node

```
location /mr.txt {
     proxy_pass http://85.x.x.x/lm/data/hosting/mr.txt;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location /pr/ {
     proxy_pass http://85.x.x.x/lm/data/hosting/partnerka/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location /tds/ {
     proxy_pass http://{removed}.name/tds/;
     proxy_redirect off;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header User-Agent $http_user_agent;
     proxy_set_header Referer $http_referer;
     proxy_pass_header Client-Host;
}
location / {
     if ($http_user_agent !~ (.+)LMK$) {
          error_page  403 404 500 502 503 504 /404.html;
          return 404;
     }
     proxy_pass http://85.x.x.x/lm/data/hosting/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location ~ ^/[a-z]*\.(png|htm)$ {
     if ($http_user_agent !~ (.+)LMK$) {
          error_page  403 404 500 502 503 504 /404.html;
          return 404;
     }
     rewrite ^/[a-z]*\.(png|htm)$ /lm/main.php last;
}
location /lm/ {
     if ($http_user_agent !~ (.+)LMK$) {
          return 404;
          error_page  403 404 500 502 503 504  /404.html;

     }
     proxy_pass     http://85.x.x.x/lm/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP    $remote_addr;
}
```

Figure 6: The default TSL *nginx.conf* configuration for the TSL servers defines the translation of HTTP requests from the public interface of the TSL to the next tier in the hierarchy (UTS). This configuration specifies how Repeater nodes must conform to a specific user-agent in order to pass traffic through the TSL tier.

in that list, a hardcoded URL in the botnet's fast-flux scheme was used, as resolving this domain returned IP addresses of bot nodes. After the new bot discovered a live node, it sent it a copy of a unique certificate generated at runtime. This certificate was forwarded to the UTS server, which used it to encrypt the botnet's current private encryption key. This encrypted key was then returned to the new bot. This new infection was then capable of participating in the botnet.

4.2.3.2 Commands and Encoding Scheme

Waledac used 5 different methods to encode data, shown in Table 1. This table documents the command types, purpose, and encryption schemes used. Note that most encoding methods involve `bzip2` compression and AES encryption with one of three unique keys. All types of communication are Base64 encoded prior to being transmitted.

Table 1: Waledac Encoding Schemes

Comm. Type	**Used For**	**Encoding Transform**
1	Node List Updating via .php Page	Base64(AES.key2(XML))
2	"GetKey" Information Exchange	Base64({header}{AES.key2(BZip2(XML))})
3	Commands 1-7	Base64({header}{AES.key0(BZip2(XML))})
4	Node List Updating using "X-Request-Kind-Code: nodes" HTTP Request	Base64(AES.key1(BZip2(XML)))
5	TSL List Updating using "X-Request-Kind-Code: servers" HTTP Request	Base64({header}{RSA/SHA1 Signature} {header} {Timestamp} {Entry Count} {IP:Port Pairs})

4.2.3.3 Peer-to-Peer Communication

Nodes in the Repeater layer were aware of one another due to the presence of XML-formatted peer lists. In early versions of the network, these were exchanged between

Repeater nodes and included nodes each peer was aware of. In later versions, however, these lists originated from the UTS and were less dynamic. The format of these peer lists is reproduced below.

```
 <lm><localtime>EPOCHTIME</localtime><nodes>
<node ip="IP" port="80" time="EPOCHTIME">HASHID</node>
<node ip="IP" port="80" time="EPOCHTIME">HASHID</node>
...
<node ip="IP" port="80" time="EPOCHTIME">HASHID</node>
</nodes></lm>
```

4.2.3.4 Hash ID Generation

Waledac hash IDs are hexadecimal strings ranging from 30 to 44 characters in length. This length varied over time through different version of binaries. The hash generation algorithm used by Waledac's infected hosts produces sufficiently random node IDs, but does small posses flaws.

The `rand()` function, reproduced below, is initialized with a call to `srand()`, and is seeded with the `tickcount()`. The TickCount [22] function returns a 32-bit value representing the number of milliseconds that have elapsed since a computer was booted. The resolution of this 8-byte value is usually between 10 and 16 millimeters, and is limited to 49.7 days. When this limit is reached the value wraps to 0. The resolution of this counter has, according to MSDN, only a resolution of 10 to 16 milliseconds or 2^{28} bits. While a hash ID is generated as an array of integers, it is converted to an array of characters. A predictable flaw exists as a result of this conversion, where every 3rd character falls in the [0..7] range. As a result, hash collisions are a remote possibility.

```
rand()
```

```
tiddata = _getptd();
newseed = 0x343FD * tiddata->_holdrand + 0x269EC3;
tiddata->_holdrand = newseed;
return (newseed >> 16) & 0x7FFF;
```

These hash IDs are stored locally in non-volatile memory in the registry branch `HKCU/Software/Microsoft/Windows/CurrentVersion/` in the key `MyID`. These values are persistent across system reboots and binary updates, allowing them to function as true unique identifiers. These hash IDs are only generated by the malware when the registry key is not found.

4.2.4 Automated Administrative Activity

This section documents the Waledac's automated activities mostly initiated by the systems deployed bot the operator(s) of the botnet. The node-auditing techniques, third-party binary repacking services, and fast-flux DNS operations are discussed.

4.2.4.1 Node Auditing

The Waledac botnet is open to observation as this document and others related to this topic have shown [30, 31, 5, 34]. The botnet has limited protection from poisoning attacks at the Repeater tier. To monitor and prevent such attacks, the botmaster uses the UTS as a self-auditing component to ensure that only legitimate Waledac bots are introducing traffic into the botnet. Simulating the behavior of a Waledac Repeater node is possible given the open XML format the botnet uses for communication. Provided that the simulated Repeater node properly handles the encryption and compression required to transmit the XML through the botnet, the construction of a simulated Repeater node that appears to be a legitimate Repeater node is trivial. The Waledac botmaster has developed creative solutions to distinguish simulated (illegitimate) Repeater nodes from real bots.

The first test performed by a UTS server when auditing a node can be called the Executable Request Proxy (*ERP*) test. When developing a simulated node, it is conceivable that the researcher would prevent the node from being used to propagate Waledac or other malicious nodes. As such, the node would drop any request for an executable by an outside (victim) entity. The *ERP* test plays against this fact by having the UTS issue a request for a specific file named *readme.exe*. The UTS will directly contact the node under audit with the URL `/readme.exe`. A real node will pass this request to the TSL server which will in turn pass the request to the UTS server. Therefore, it is possible for the UTS to track from start to finish the request and reply for *readme.exe*. The contents of readme.exe consist of two bytes which simulate the DOS header of a PE/COFF file, the letters `MZ`. A variation of the *ERP* test is also performed randomly when the UTS requests *readme.txt* instead of *readme.exe*. The reply to this variation of the *ERP* test is the string `Hello`. During a two hour period, the observed UTS server issued 597 *ERP* tests.

The second test performed by a UTS server focuses on the DNS component of a Repeater node. Since a simulated Repeater node would not necessarily need to participate in the DNS portion of the Waledac fast-flux network, it is conceivable that researchers would simply ignore DNS requests. To test for this possibility, the botmaster introduced a new domain into the Waledac fast-flux configuration named *hellohello123.com* in August of 2009. The domain currently does not have an associated name server and as such cannot be resolved though the *.com* Top Level Domain

(TLD). The Domain Response (*DR*) test uses the fast-flux network configuration in order to determine the validity of the audited node. The UTS issues a DNS lookup for *hellohello123.com* by querying the node under review. Since *hellohello123.com* is part of the fast-flux network configuration, a valid repeater would return one of the predefined IP addresses from the configuration data. A simulated repeater would potentially fail this test by either returning invalid information or not responding at all. Therefore the *DR* test can identify invalid Repeater nodes based solely on their response to a specific, non-resolvable domain query. The UTS issued 693 *DR* tests during a two hour period of observation.

4.2.4.2 Third-Party Repacking

Waledac did not employ rootkits in order to hide from antivirus applications, but rather it used a constantly changing set of packed binaries to avoid signature detection. There are approximately 50 known versions of Waledac in the wild, but there are over 3200 different binaries for these 50 versions [37]. The Waledac binary is routinely repacked resulting in the large number of binaries each with a unique MD5 hash (or signature). The frequency at which these binaries are repacked is exceedingly high and requires automation. The UTS employed a third party service provider at *crypt.j-roger.com* and *cservice.j-roger.com* to repack Waledac binaries. In order to repack a binary, the UTS system sends a POST request to one of the two URLs `crypt.j-roger.com/api/apicrypt2/ {16 hexadecimal digit hash}` or `cservice.j-roger.com/api/apicrypt2/ {16 hexadecimal digit hash}`. Contained within the POST is an action form detailing the specifics of the repacking request along with the binary to pack in a modified version of Base64. Figure 7 illustrates an example, pulled from the UTS network traces, of the POST payload sent by the UTS during a request for a repack of a Waledac loader binary.

```
files={"0":{"filename":"loaders\/
gera.exe","filebody":"TVqQAAMAAAAEAAAA….AAA=","packer_before":"none","pac
ker_after":"none","iconname":"","iconbody":""},"files_count":
1,"profiling":{"client_start_transmitting":1251448624}}
```

Figure 7: UTS requesting a binary repack.

On average, the packing service at *j-roger.com* returned a repacked binary in 4 seconds. This allows the UTS to repack multiple binaries in a very short period of time. During a two hour period, Waledac was observed requesting (and receiving) 157 binaries through the *j-roger.com* service. When the service returns the binary to the UTS, the server uses a similar format as the request as seen in Figure 8.

```
{"files":[{"systemid":"213555","filename":"loaders\/
gera.exe","filesize":"18944","packer_before":"none","packer_after":"none"
,"description":"","upload_status":true,"crypt_status":"true","crypt_times
tamp":"2009-08-28
12:37:08","crypt_md5_hash":"6a8c37c99f5cae04818fcb95008c63ed","crypt_body
":"TVqQAAMAAAAEAAAA…AAA="}],"files_count":1,"profiling":
{"server_after_receive":1251448624,"server_after_checking":
1251448624,"server_after_parsing":1251448624,"client_start_transmitting":
1251448624,"server_after_crypting":{"0-loaders\/gera.exe":
1251448628},"server_before_sending":1251448628}}
```

Figure 8: Reply from *j-roger.com* containing the repacked binary.

4.2.4.3 Fast-Flux DNS

The Repeater layer in the Waledac botnet participated in a fast-flux DNS scheme managed by the UTS node. Using an automated script, the UTS sends updates through the Repeater tier, functioning as a SOCKS proxy, to the *xinnet* master name server for Waledac domains. The script pulls a list of active Repeater IP addresses and updates NS records accordingly. These fast-flux domains are used in spamming and propagation campaigns. By using the SOCKS proxy layer within Waledac, the UTS server does not reveal its location to the external DNS server.

4.2.5 Malicious Output

The Waledac botnet relied chiefly on spam dissemination for infected-host monetization. Ultimately, the UTS tier is responsible for acquiring the information to place in the spam campaigns. Evidence of this behavior is found in a series of requests to the spam warehouse website at *spamit.com* [29]. The Spamit system is a known clearinghouse for so-called Canadian Pharmacy websites. On multiple occasions in a very short period of time (less than one hour) the UTS used the `wget` application to query the Spamit website for new domains to enter into the current spam campaign. The UTS queries the spamit.com server using a simple HTTP GET request that takes the form of `GET /export.php?aid={affliateID}&mode=personal&design=blue&secure={hash token}`. The request generates a simple list of domain names such as *http://offerled.com*, *http://toldtool.com* and *http://hourshine.com* with each domain name separated by a newline break. This information is disseminated downward into the botnet for use in different spamming campaigns.

A unique feature of this botnet was the presence of differentiated spam campaigns. In addition to the more traditional method of using open mail relays or sending mail directly, Waledac possessed the ability to send authenticated spam using SMTP-AUTH. These campaigns can be referred to *LQS* (Low Quality Spam) and *HQS* (High Quality Spam), respectively.

The majority of Waledac's spam was sent from the Spammer tier. This campaign type was characterized by bulk spam with a higher probability of being blacklisted due to the originating IP addresses being dynamically assigned (e.g. residential cable modems or DSL services). The Spammer nodes that ultimately transmitted this type of Low Quality Spam (*LQS*) kept detailed statistics on if a particular piece of spam from a particular campaign destined for a particular email address was successfully transmitted. This operation is illustrated in 9.

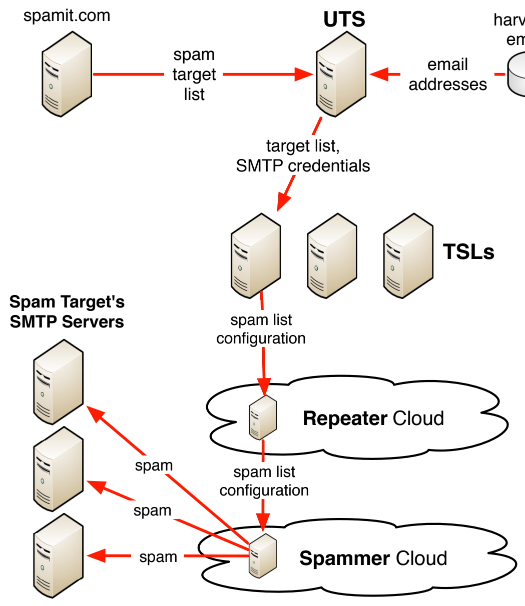


Figure 9: Waledac's low-quality bulk spam generation model, where delivery success is not of high importance. This is referred to as *Low Quality Spam* (LQS).

As LQS is likely to run into blacklists during mail dissemination, the operators of this botnet developed a method of sending mail using legitimate email accounts with their associated SMTP servers. Numerous tiers in the Waledac botnet are leveraged to produce this type of mail. While the TSL tier has been described in prior research as a simple proxy or obfuscation layer serving to hide the location and existence of nodes above it, nodes in this tier also function as secondary command and control servers in the HQS campaigns.

The botmaster, as part of the initial deployment of a TSL server, installs a custom PHP application called `php_mailer`. This application is a simple bulk mailer that is coupled with an open source package known as *proxychains* [2]. Equipped with a collection of validated SMTP login credentials, the botmaster generates between 100 and 300 instances of `php_mailer`. The bulk mailer connects to a cloud of proxy servers via SOCKS5. These proxies in turn connect to the specified SMTP server via TCP port 25. The `php_mailer` application uses valid login credentials to authenticate with the SMTP server before sending multiple spam emails from the victim's account. The result of this HQS scheme, shown in Figure 10 was spam with a higher probability of successful delivery.

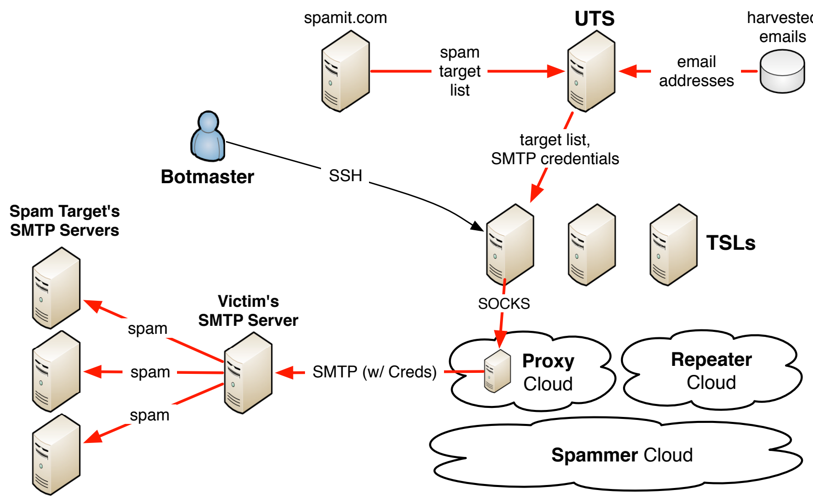


Figure 10: Waledac supports the unique ability to send spam using stolen email credentials. This provides a high delivery success rate. This is referred to as *High Quality Spam* (HQS). In the *Low Quality Spam* (LQS) campaigns, nodes in the Spammer tiers send the more common type of unauthenticated spam.

4.2.6 Monitoring Interface

By carefully examining the `nginx.conf` file, a mechanism for interacting with the UTS was discovered, which allowed for the HTTP-based retrieval of botnet logs and the ability to observe graphs created with RRD-tool. This log-retrieval method is shown in Figure 11. A log excerpt is shown in Figure 12. Four types of logs were available: `1-first.txt`, `2-notify.txt`, `4-words.txt`, and `6-httpstats.txt` and followed a similar format. The logs correspond with command types used by the botnet when infected systems report-in or request data from the network. Common fields include dates, timestamps, IP addresses of infected systems and intermediary nodes (used in traffic forwarding), hash IDs, node types (Repeater of Spammer) and affiliate IDs. Fields were delimited with space characters.

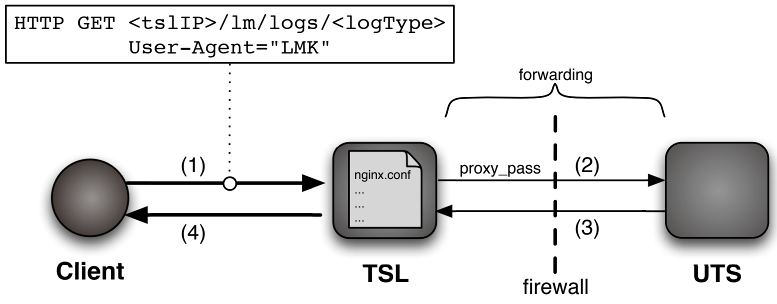

```
HTTP GET <tslIP>/lm/logs/<logType>
        User-Agent="LMK"
```

Figure 11: Retrieval method used to access logs stored on the main Waledac command and control server.

In total, 5.1 GB of plain-text logs were collected from December 4th through the 22nd in 2009. These logs provide total coverage in terms of node enumeration, and were used in later research pursing the relationships between network addresses and unique infection IDs. It should be noted that spoofed network traffic can arrive at the UTS node and, as a result, can be written to disk in these logs. This requires a moderate amount of knowledge on the botnet's communication protocol, however, as various correct encoding methods must be used.

```
04/12/2009  10:42:41  121.190.114.138   97.100.150.154  89.149.226.65   51  5e300d441c49064db74d4715de54b3  R  5.1.2600  sware51
04/12/2009  10:42:44  121.162.27.26     118.36.212.192  217.23.3.226    51  e33555107c4aff1f3e321a27f1599d  S  5.1.2600  sware51
04/12/2009  10:42:46  59.94.240.71      121.140.186.12  95.211.8.161    51  1c11313dbb59af3c4d597525a4430e  S  5.1.2600  sware51
04/12/2009  10:42:48  125.177.44.191    121.140.186.12  95.211.8.161    51  2a5c845c9862a4749964a2293f46ed  S  6.0.6002  sware51
04/12/2009  10:42:51  119.149.86.34     24.124.69.193   89.149.208.241  51  5e300d441c49064db74d4715de54b3  R  5.1.2600  sware51
04/12/2009  10:42:56  122.42.104.16     121.140.186.12  95.211.8.161    51  4342577bd239a71e06374062bc683d  S  5.1.2600  sware51
04/12/2009  10:43:00  203.78.118.6      93.177.154.174  89.149.208.241  51  612c1c60b675ba71a8199e166b272a  S  5.1.2600  sware51
04/12/2009  10:43:01  60.191.116.66     121.140.186.12  89.149.242.175  51  5964dd1c9602512e5934c366712152  S  5.1.2600  sware51
04/12/2009  10:43:05  94.96.220.126     97.100.150.154  217.23.3.226    51  46545e1ad05eb725f84a43096a304f  S  5.1.2600  sware51
04/12/2009  10:43:06  118.101.29.234    117.123.130.47  93.174.93.73    51  453e232af214bc666d222265dc6091  S  5.1.2600  sware51
04/12/2009  10:43:08  117.204.97.223    68.61.124.17    89.149.208.241  51  cc654a169351df313831600e7971ef  S  5.1.2600  sware51
```

Figure 12: Excerpt from `1-first.txt` retrieved from the UTS node. The fields are date, time (GMT+3), reporting node IP address, repeater address used in forwarding, TSL address, node binary version, node hash ID, node operating mode (Spammer or Repeater), Windows version, and affiliate ID string.

4.2.7 Business Model

The affiliate program, or *partnerka* used by Waledac is known internally as the *FairMoney* system. Details regarding FairMoney were stored on the UTS system and discovered during file-system analysis. In this program, binaries were tagged with unique affiliate IDs, and affiliates were tasked with the propagation of binaries. They received funds through the Webmoney [3] based on the number of infections that resulted and their longevity. An excerpt of the contract affiliates agreed to (which could be found at `http://<waledacdomain>/pr/rules.html` which describes this reward system is reproduced below. Note that this was translated from Russian.

> "Average bot lifetime lifetime of bots at least 2 hours (120 minutes) - not paid
> Average lifetime bot 2 hours (120 minutes) and more - $25 per 1000 downloads
> Average lifetime bots 4 hours (240 minutes) and more - $50 per 1000 downloads
> Average lifetime bot 8 hours (480 minutes) and more - $100 per 1000 downloads
> Average lifetime of bots 24 hours (1440 minutes) and more - $200 per 1000 downloads"

Inspection of the traffic logs maintained on the UTS revealed the following affiliate IDs: `abc`, `assam`, `asti`, `atata`, `aunt`, `aunt2`, `birdie`, `birdie2`, `birdie3`, `birdie4`, birdie6, birdie7, birdie8, birdie9, dmitriy777, exp7, july4th, kopyha, krab, lacrimo, sa, lynx, mirabella, mirabella_dies, mirabella_exp, mirabella_exp2, mirabella_exp3, mirabella_exp4, mirabella_exp5, mirabella_exp6, mirabella_site, panda, panda3, pp, s52, savage, slavik, spyware, spyware2, spyware3, spyware49, steelman, stockholm, stratum, sware50, sware51, swift, swift2, swift3, swift5, traff, tty2, ttyimsn, twist, ub, udu, and william. With consideration that several of these affiliate strings are similar (identical with apparent versioning characters appended) it is possible there were at least 35 affiliates participating in Waledac's FairMoney program. The two most popular affiliate IDs found in the UTS log data are sware51 and s52, which are found in 82.6% of all hashID-affiliateID pairs. It should be noted that some hash IDs are associated with more than one unique affiliate ID.

4.2.8 Tier Ratios

The ratio of systems in the infected-host tiers can be explored to further define the structure of this botnet. This ratio is not fixed, but rather is the natural result of segregating infections with public and private local IP addresses to separate tiers. Using the UTS population log data, filtered to remove anomalous or spoofed traffic associated with experiments by other universities and bogon [39] IP addresses, one can determine daily unique counts for IP addresses and hashes for each tier.

The ratio of Spammer to Repeaters for 18 days in December 2009 is shown in Figure 13 for IP addresses and hash IDs. For hash IDs, the mean Spammer:Repeater ratio is 18.974, with sample standard deviation of .796. For IP addresses, the mean tier ratio is 22.81 with a sample standard deviation of 1.20. The tier ratios in terms of IP addresses were explored as this is publicly viewable. The reader should note that this data is somewhat distorted as the last day of collection was incomplete, and nodes did not experience as much DHCP churn. As the hash IDs are unique identifiers for each infection, these reveal a true tier ratio. Two polynomial functions were fit to this data and are also featured in Figure 13.

One can surmise that for botnets which follow a similar architectural model to Waledac, where two separate tiers are used for nodes with private and public IP addresses, this ratio might be similar. This ratio would be influenced by several factors, however. Namely, a sufficiently large and random sampling of IPv4 space would be required to determine a true ratio between hosts with public and private network addresses. Relative to occupied IPv4 space, the size of this botnet is quite small, and likely not evenly distributed. As Waledac propagated through email addresses, however, the distribution of its infections would be more random than malware which spreads in local subnets, either through physical media or network scans followed by successful exploits.

4.2.9 Demise

On February 24th 2010, the Waledac botnet was systematically incapacitated in an multilateral operation called "Operation b49" involving Microsoft and several anonymous private-sector information-security companies. This operation initially targeted the network's fast-flux DNS infrastructure. Following a complaint filed by Microsoft alleging illegal activity associated with 277 domains, a court-order from a federal

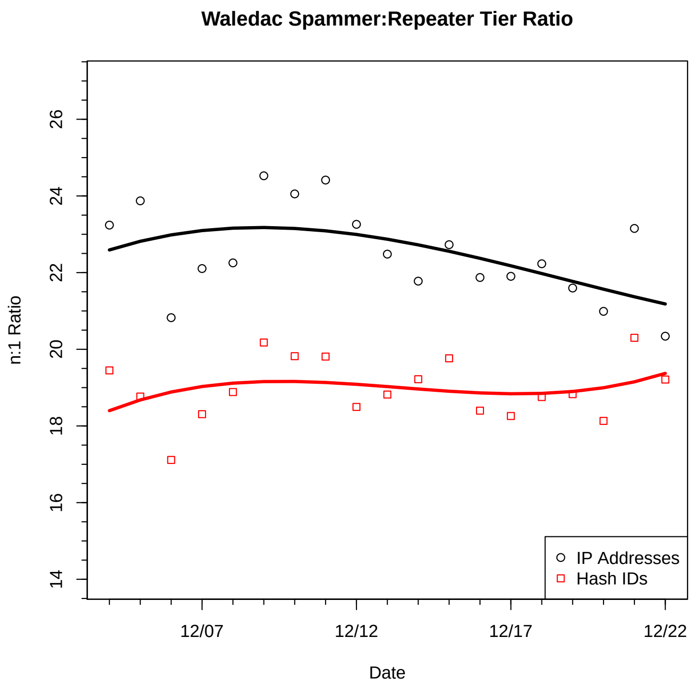


Figure 13: Tier ratios (Spammers:Repeaters) for Waledac hash IDs and IP addresses, as viewed from the UTS logging system.

judge was issued which instructed Verisign to de-register these domains. With the fast-flux infrastructure effectively razed, a peer-to-peer poison attack was initiated, using methods developed in part by Ben Stock. This attack was designed to redirect incoming data requests to sinkhole nodes deployed in Microsoft IP space. Finally, several of the systems deployed by the botmaster were taken offline by their respective Internet Service Providers.

The methodology described by Sinclair et al. in [31] was roughly followed, though the steps were enacted in the incorrect order for guaranteed success. The network poisoning occurred after the incapacitation of the fast-flux network, which allowed a brief window of time for the botnet operators to issue an update which instructed bot nodes to move to an emergency secondary command and control channel. Fortunately, this did not occur. The reader should note that this attack did not remove the malware from the systems residing in the two infected host tiers. Rather, these systems are merely no longer able to receive instructions from the botmaster or exfiltrate sensitive data.

CHAPTER 5: DESIGNING AND EVALUATING ENUMERATION TOOLS

This chapter presents real-world implementations of modern botnet enumeration tools, and explores their coverage and performance characteristics when applied to two notable botnets: Storm and Waledac.

5.1 Enumerating the Storm Botnet

Our analysis of Storms protocol was complemented by an enumeration attempt and the creation of an infiltration-based tool. The tool was teamed with a separate component to test for the presence of firewalls during live enumeration.

5.1.1 Passive Peer Monitor with Live Firewall Probing

The design of the tool is extensively discussed in this section, along with an attempt to model its coverage, and the probability that a different types of nodes will receive messages from either PPM instances or legetimate bots. The coverage for this passive enumeration tool is also compared to a basic network crawler.

5.1.1.1 Design

To discover systems participating in Storm, a Passive P2P Monitor, referred to as *PPM*, was developed, and paired with a tool to test for the presence of firewalls: the *FWC* or Firewall Checker. A PPM node speaks Storm's Overnet protocol and participates in the network routing protocol. It does not send any malicious traffic, however; it only passively listens in to the Storm network and acts as if it is a legitimate bot by routing messages. An additional component of this enumeration tool is the Firewall Checker (FWC), which sends an additional, external request to a node following a request to the PPM. If the node does not respond to the request from the FWC, it is likely the node is behind a firewall. To detect source IP spoofing, the PPM also is designed to send requests to Storm nodes which contact it to verify host legitimacy.

Since an open-source implementation of Overnet protocol is not available, the aMule [1] P2P client was modified to implement a PPM node capable of communicating with the Storm botnet. GUI and file-sharing features of aMule were disabled, which allowed the PPM node to use very little memory and processing power. After the switch to an encrypted network, the PPM node was further changed to be able to talk to the encrypted Storm network. The PPM does respond correctly to all routing requests.

During network monitoring, source IP address spoofing can affect accuracy in enumeration. To remedy this, the tool includes a *handshake* mechanism which, following a request from a node and a reply from the PPM, a request is sent from the PPM to this node. If the node responds, the IP address associated with that node is not being spoofed. The node may also not respond due to DHCP churn, packet loss in transmission, or lack of full protocol emulation if an illegitimate node is being queried.

To further improve accuracy, a mechanism was incorporated into the PPM design to test if a node is behind a firewall. This Firewall Checker (FWC) sends an additional, external request to a node following a request to the PPM. If the node does not respond to the request from the FWC, it is likely the node is behind a firewall. This is due to the fact that nodes behind firewalls will only allow inbound traffic when they initiate the connection. Under a number of circumstances nodes not behind firewalls may also fail to respond to FWC queries. These reasons, listed above, also prevent the PPM from receiving replies to data requests. Figure 14 shows the components of the final PPM and FWC architecture.

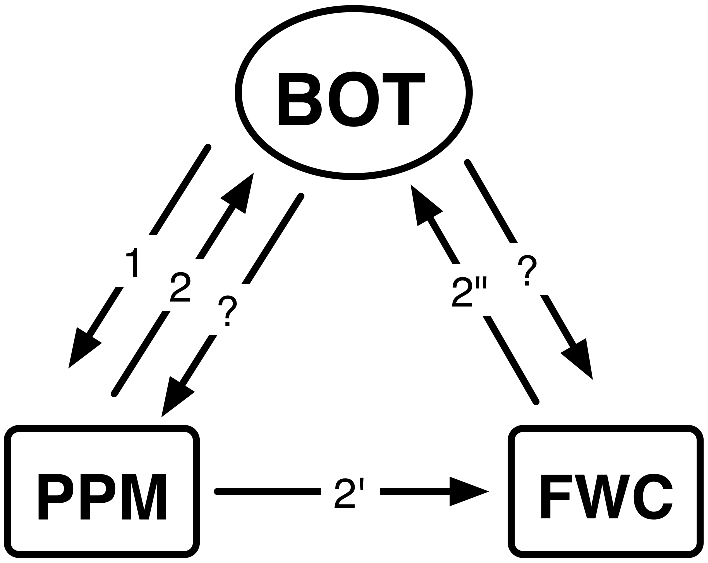


Figure 14: Final Design of the PPM

1. A Storm node in the bot network sends a request to a PPM instance

2. The PPM node replies to the request and sends another request to that Storm node

2'. At the same time, PPM also sends a message to FWC telling it to send a similar request to that Storm node

2". Upon receiving this message, FWC sends a request to the same Storm node (same request that PPM sent to that Storm node).

If the Storm node replies to 2', the IP address is not spoofed. If the Storm node replies to 2", it is not behind a firewall or a NAT device.

For this measurement experiment, occurring from August 25th to September 8th of 2008, 256 PPM nodes run on two computers and deployed evenly across Storm's ID space, each bootstrapping using a different peer in the network to ensure more complete coverage. A single system in the same local network was used for running an instance of the FWC tool, and an additional computer was used to deploy 16 virtualized instances of real Storm executables in Virtual Machines. One final computer was used to deploying a crawler in parallel to the PPM instances, which allows one to compare the network coverage for each method of enumeration. This crawler was designed to send Overnet routing requests to known nodes and then perform additional queries to any discovered nodes. The systems in this network were not situated behind a hardware-based firewall, though illicit traffic from the virtualized execution of actual Storm binaries was blocked.

Analysis of the gathered population data begins with a determination of how many Storm bots are behind firewalls. The CDF of the fraction of responses for every node is shown in Figure 15, where the fraction of responses for an IP addresses is calculated as:

$$\frac{\texttt{Number of Responses}}{\texttt{Number of Responses} + \texttt{Number of Timeouts}}$$

On average, each node in the peer-to-peer network responded to more than 60% of the PPM requests. Roughly 6% of the IP addresses queried never responded to any requests from the PPM. Nearly 46% of the discovered nodes did not respond to the FWC which likely indicates that these nodes are behind firewalls or NAT devices. Failure to respond to the PPM or FWC could result from other phenomenon described

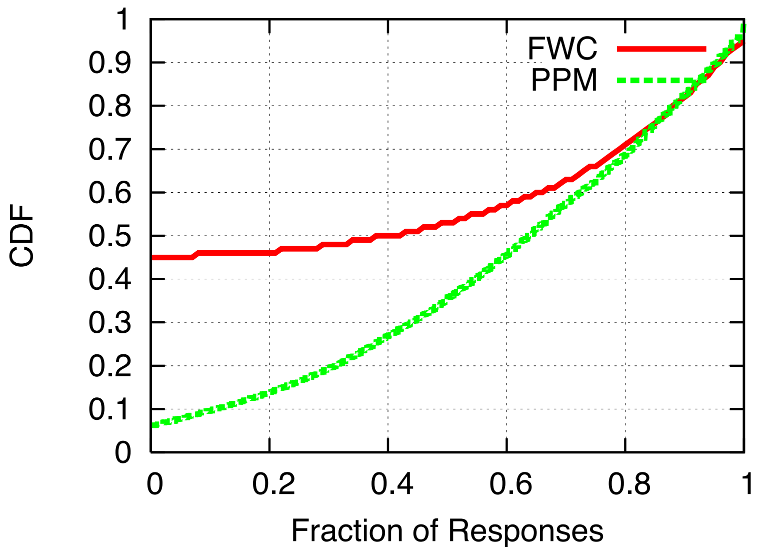


Figure 15: CDF of fraction of responses per IP address for both PPM and FWC

earlier in this section.

The number of IP addresses discovered daily by the PPM and network crawler are shown in Figure 16. Note that the 10 day period where crawler data is not shown is due to the PPM tool being deployed prior to the crawler. The IP addresses described in this figure are daily totals, not daily unique totals, exclusive of previously-seen addresses. The coverage of the PPM tool is consistently higher than the crawler largely due to the ability of this enumeration technique to discover nodes behind firewalls.

Whether the enumeration tools are discovering *different* nodes must also be assessed. Figure 17 shows the percentage of IP addresses found by the crawler and the PPM instances for each day of execution. While a majority of the bot nodes discovered by the crawler were discovered by PPM nodes, a significant number did not send a message to the PPM instances. By closely examining these nodes (set C), we found that these bots had short lifetimes, specifically an average of 19 minutes, while nodes

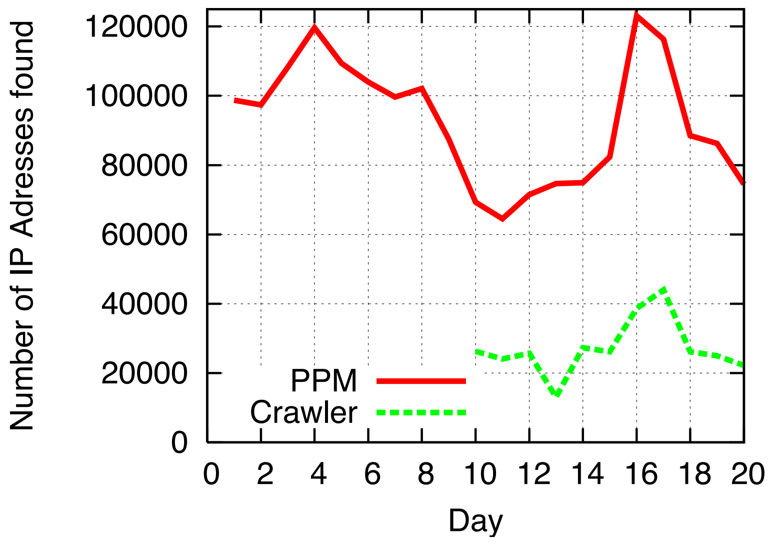


Figure 16: Number of IP addresses found by crawler and PPM per day

found by both tools, (comprising set I), had an average life of 100 minutes.

Figure 18 shows the CDF of the lifetimes of the nodes for both cases C (Crawler $-$ PPM) and I (Crawler & PPM). 80% of nodes in set C have a maximum lifetime of 10 minutes, while 80% of the nodes in set I have a lifetime of 50 minutes. Nodes which have a short lifetime are not found by PPM instances. This is expected as the crawler actively seeks new nodes, while the PPM instances passively wait to receive messages from peers.

5.1.1.2 Modeling Coverage

The yield from the crawler and PPM instances in this experiment indicated that network crawling was not an ideal method to enumerate peers in the Storm network. This section explores in more detail the coverage of the Storm network obtained by the PPM tool. The likelihood that at least one PPM node will receive a message by a Storm node is first determined. This analysis is based on the *bins and balls* problem.

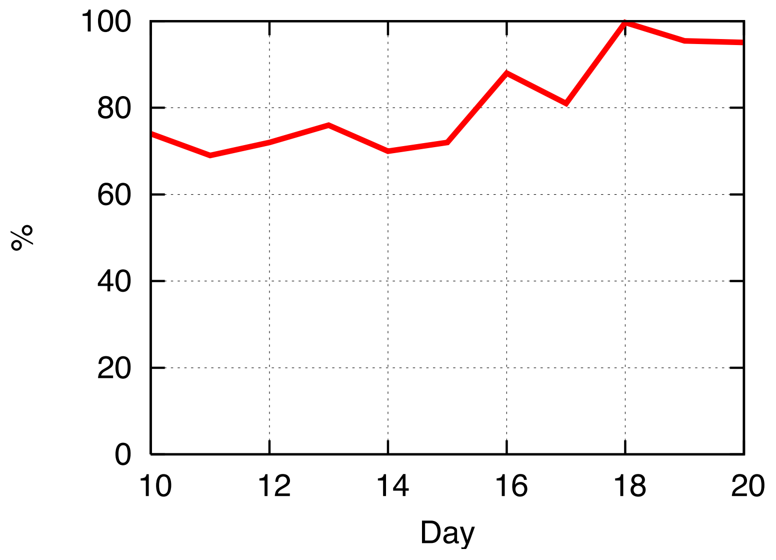


Figure 17: Percentage of IP addresses found by crawler that were also found by PPM per day

In the *bins and balls* problem, one must first assume that they have n bins and 1 ball and must throw the ball into one of the bins. The probability that each bin will get the ball is $\frac{1}{n}$, assuming that each bin is equally likely to receive the ball. The probability of a bin not receiving a ball is thus $1 - \frac{1}{n}$. If k balls are thrown instead of just 1 ball, the probability of a bin not receiving a ball is $(1 - \frac{1}{n})^k$. Finally, the probability of a bin receiving at least 1 ball is $1 - (1 - \frac{1}{n})^k$.

The same model can be used to determine the likelihood of a a PPM node receiving at least 1 message from a bot. Simply changing the variables does not work, however, because in Kademlia-style P2P networks, all the nodes do not have an equal probability of receiving a message. Some nodes have a higher probability of receiving a message than others. For example, since a PPM node is online for a long period of time, it will be in the routing tables of many peers, with the probability it will receive a message is higher than a Storm node that is only online for 15 minutes per

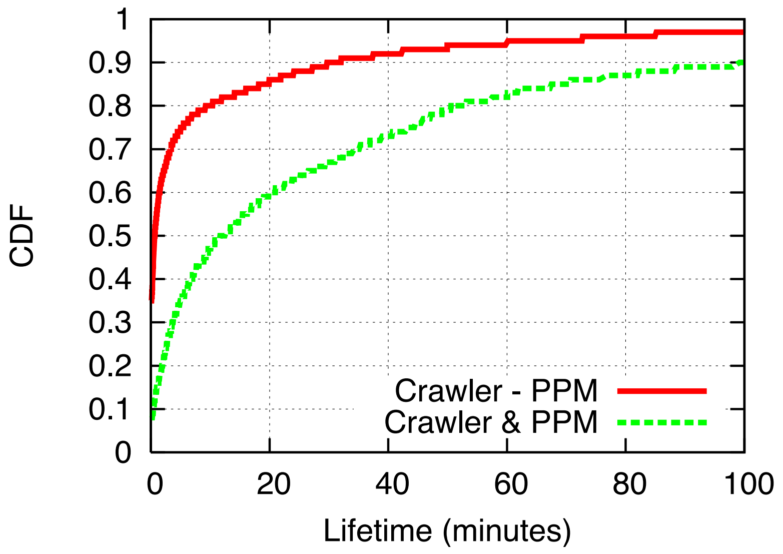


Figure 18: CDF of the lifetimes of nodes that (i) were found by the crawler but *not* by PPM instances and (ii) found by *both* the crawler and the PPM instances.

day – a reasonable time for a user to perhaps check their email.

In the bins and balls problem, some bins are “larger” and have a higher probability of receiving a ball than other bins. Different nodes in the Storm network have a different probability of receiving a message, as shown in Figure 19 and Figure 20. The probability of a PPM instance receiving a message from a bot is calculated as $L = 1 - (1 - p)^k$, where p is the probability of PPM receiving a message from a bot for a particular hash, and k is the number of nodes a bot sends a message related to that hash. Next, the value of p is experimentally determined, showing L is for varying values of k.

5.1.1.3 Probability of the PPM Receiving a Random Message

To obtain the probability of p of the PPM instances receiving a message from a Storm-infected node, network data from the virtualized instances of Storm binaries was utilized. Only 3 message request types were analyzed:

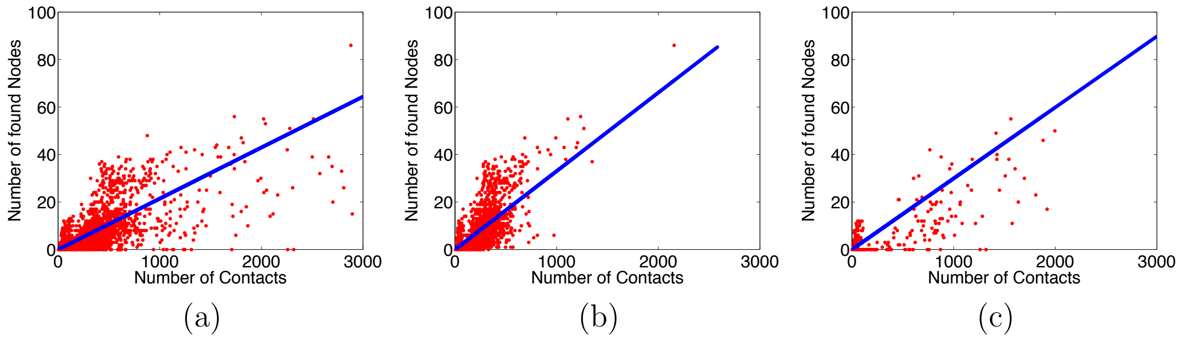


Figure 19: Number of contacts a bot sends a message to and the number of those which are PPM nodes for (a) *Search*, (b) *GetSearchResult*, and (c) *Publish* message types.

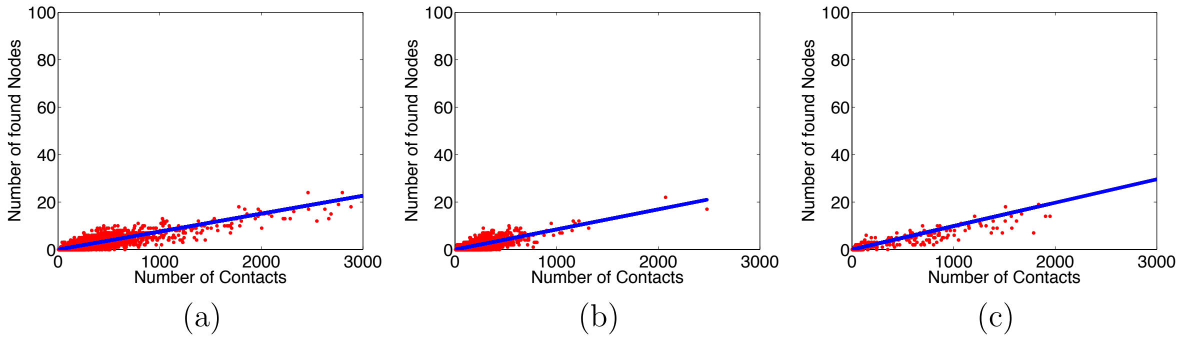
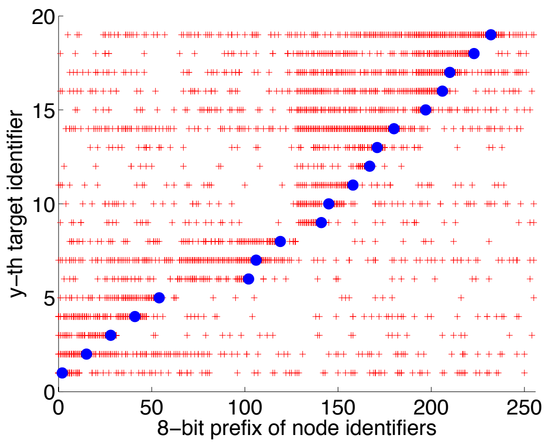
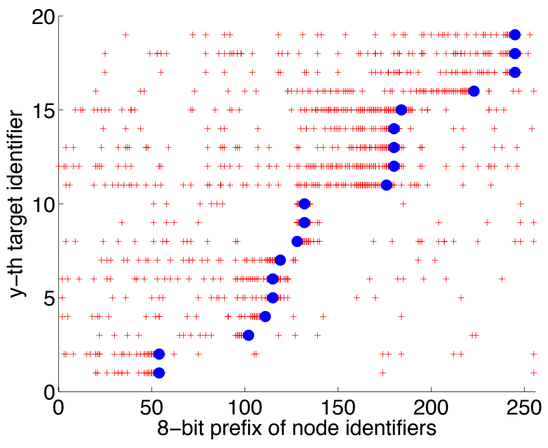


Figure 20: Number of contacts a bot sends a message to and the number of those contacts which are a certain bot for (a) *Search*, (b) *GetSearchResult*, and (c) *Publish* message types.



(a)



(b)

Figure 21: Distribution of Storm node IDs for *search* and *publish*.

- *Search* - A message used in routing to find replica roots
- *GetSearchResult* - A message sent to potential replica roots to obtain results (binding information)
- *Publish* - A message intended to distribute binding information

The other two request messages in the Overnet protocol are *Connect*, used to bootstrap on to the network, and *Publicize* which is a type of "keep-alive" message sent regularly.

Observed *Connect* messages are excluded from analysis as they may introduce bias for nodes that constantly churn in and out of the Storm network as they are produced only by incoming nodes. The *Publicize* requests are also excluded from analysis as these messages are sent only to nodes in routing tables, and will include bias towards nodes that are unusually long-lived, such as PPM instances or other experiments. The other three types of messages are sent more uniformly as shown in Figure 21. Each node in the network, regardless of their node IDs, is equally likely to receive *Search*, *GetSearchResult*, and *Publish* messages. 20 target IDs were monitored for *GetSearchResult* and *Publish*, represented by the 20 blue points on the graph. The red points represent nodes that are contacted for those target IDs. The x-axis represents the first 8 bits of the ID (in decimal) of the nodes contacted. For each target ID, the nodes contacted are uniformly distributed across the ID space. More nodes are contacted closer to the target ID since those nodes are potentially “replica roots.”

Figure 19 shows the number of nodes that each of the 16 virtualized bots sends a request to and the number of our PPM nodes which are among those nodes, for the three message types described above. For example, Figure 19 (a) shows the number of contacted nodes for *Search* requests from each of the virtualized bots. Point $[1000, 22]$ on the graph indicates that a *Search* request with hash H was sent to 1000 nodes and 22 of those 1000 nodes were PPM nodes. Each plot point represents a request for a different hash. A least-squares line of best fit is included, which was determined using the *polyfit* algorithm found in Matlab [18].

For *Search* requests, the probability p (slope of the line of best fit) of the PPM receiving a message from a Storm node is 2.3%; for *GetSearchResult* requests, the probability is 3.3%; and for *Publish* requests, the probability is 3%. These results are shown in Table 2.

Table 2: The probability for a random Storm node and a PPM instance for receiving each of the 3 message types.

	Search	*GetSearchResult*	*Publish*
PPM	2.3%	3.3%	3%
Bot	0.95%	0.9%	1.0%

The number of PPM nodes deployed was also varied to observe the impact on coverage. Figure 22 shows the result of this manipulation for the three message types.

A linear increase in the probability that the PPM instances will receive a message from a bot is observed following an increase in PPM instances. From this graph, one can obtain the probability p of the PPM instances receiving a message with a particular hash from a Storm node to be 3% with 256 PPM nodes. Although it might be tempting to keep increasing the number of PPM nodes to see if the probability keeps increasing linearly, we show that even with $p = 3\%$, the probability L of PPM being sent a message from any bot is close to 100%.

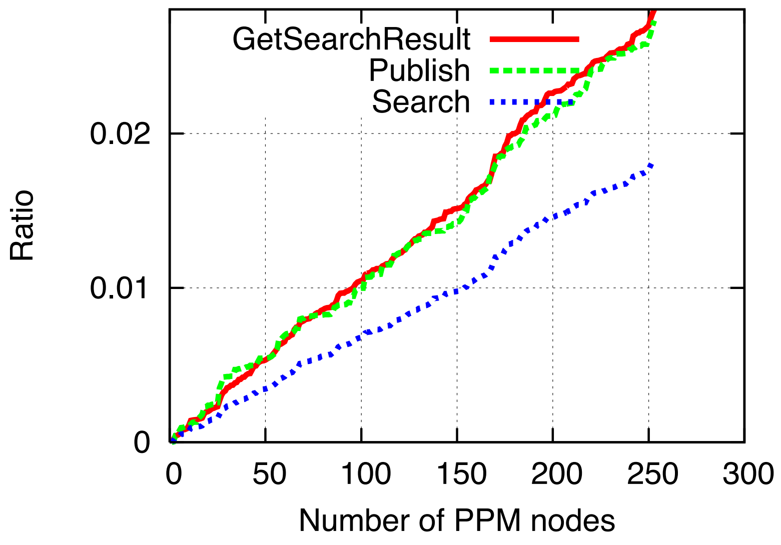


Figure 22: The probability of PPM instances receiving a message from a Storm node for varying numbers of PPM nodes.

Figure 23 shows the plot of $1 - (1 - p)^k$ with varying k, where k is the number of nodes that a bot sends at least one message with a Storm hash. Thus, instances of the PPM tool will receive a message from a bot with high probability (87%) if k is greater than 100 contacted nodes. If $k = 200$, the probability goes up to 98%. Looking at both Figures 19 and 20, each bot sends the same hash (for either of the three message types) to at least 200 nodes, suggesting that the PPM instances can find most of the nodes in the Storm network. It should be emphasized that running 256 concurrent instances of the PPM tool is not resource intensive; each instance uses only a few megabytes of memory and a small amount of bandwidth and CPU usage.

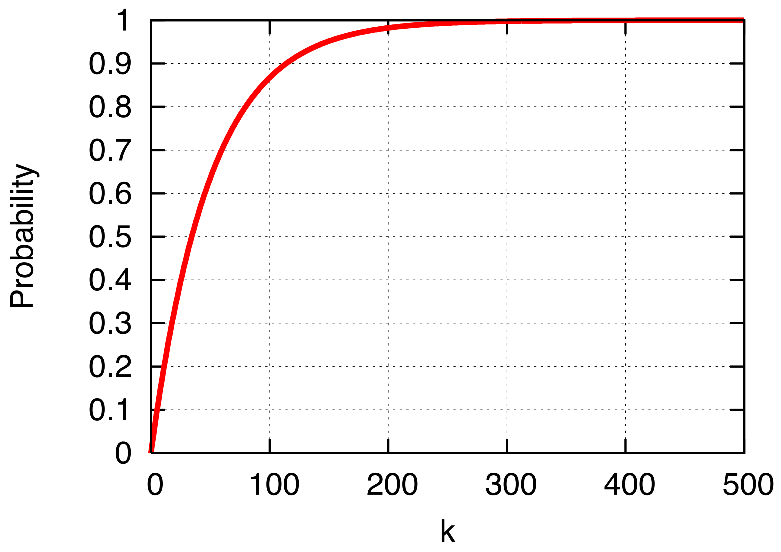


Figure 23: Plot of the probability of 256 PPM nodes receiving a message from all the Storm bots for varying k, where k is the number of nodes contacted by a Storm bot

5.1.1.4 Probability of a Set of Bots Receiving a Random Message

The probability of the PPM receiving a message from a bot is next compared with the probability that a bot will receive a message from another bot for each hash. 256 "long-lived" IP addresses were chosen in the virtualized-bot data set. Figure 20 shows the three graphs, along with the line of best fit. The slopes of each graph is less than the slope for the graphs from Figure 19 for each message type. The probability of 256 PPM nodes seeing a message from a Storm node is higher than the probability of 256 bots seeing a message from the same Storm node. This does **not**, however, indicate that deploying 256 PPM nodes provides a better coverage of the network than deploying 256 bare-metal (or virtualized) bots. Rather, in this experiment the PPM instances had resided in the network for a longer amount of time than the average Storm node and were more likely to be found in peer routing tables. The second line in Table 2 shows the probability for each message type for the probability p of other bots receiving a message from a legitimate Storm bot.

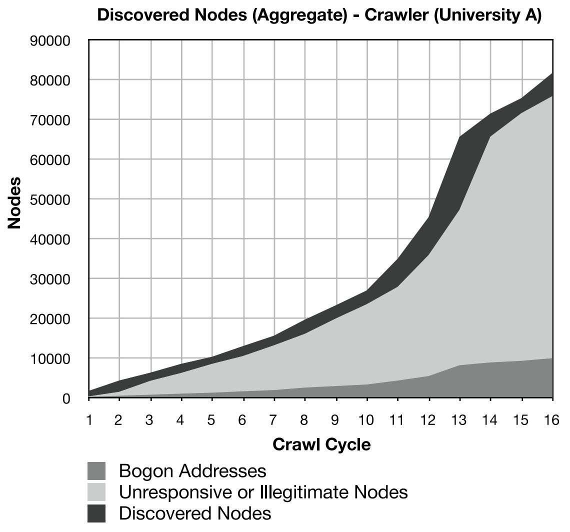
5.2 Enumerating the Waledac Botnet

This section presents three tools created to enumerate Waledac: a crawler, fast-flux enumerator, and active infiltration-based monitoring tool.

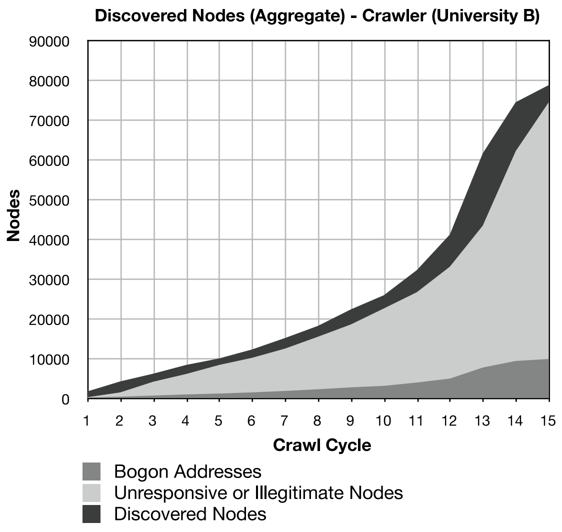
5.2.1 Walleyworld: A Network Crawler

A network crawler was developed that follows the methodology described in Chapter 3 in Chapter 3.1.3. This crawler, dubbed *WalleyWorld*, was designed to discover peers participating in the version of Waledac where Repeater nodes maintained peer data in memory (effectively serving as "routing tables"). The crawler used a fast-flux domain to seed a table of known nodes, ask these nodes for peer lists, decipher responses, and continue to recursively query discovered nodes. Unresponsive hosts and bogon [39] IP addresses were routinely expunged from traversal lists and were rarely queried.

In the experiment, the crawler was deployed for 39 days in March and April of 2009 at two separate universities, which were run concurrently with a fast-flux DNS exploitation tool to compare coverage for each of these enumeration methods applied to this architecture.



(a) Deployment: UNCC.



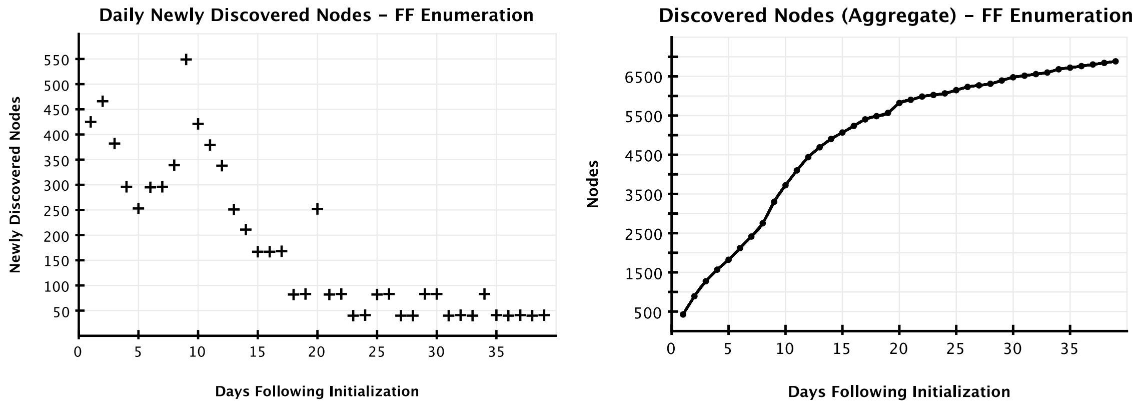
(b) Deployment: Georgia Tech.

Figure 24: Discovered nodes using our *walleyworld* crawler based on methodical peer-table requests. A crawl "cycle" is defined as one complete traversal of our botnet node index-lists, which are created using discovered bot nodes in previous cycles. Cycle completion time grows as traversal lists are populated, but this is limited by the removal of illegitimate and unresponsive addresses during each cycle.

5.2.2 nswalk: A Fast Flux DNS Exploitation Tool

Waledac used fast-flux DNS throughout its existence. Only bot nodes residing in the Repeater layer participated in the botnet's fast-flux DNS scheme. By repeatedly resolving domains associated with the botnet and treating returned addresses as name-servers themselves, one can attempt to enumerate this tier. This querying strategy is referred to as "walking," and offers an additional vantage-point to discover infections participating in Waledac. Unfortunately, only IP addresses are discoverable.

To perform this method of host discovery, a tool originally created by Jose Nazario, `nswalk`, was adapted to first query a normal route for a given domain, then treat the returned IP addresses as name-servers and perform NS queries. Queries are then again performed on each NS record that is returned. With time, this process can produce a large population of nodes participating in the fast-flux DNS infrastructure. A single instance of the *nswalk* tool was deployed alongside the two network crawlers for 39 days in late March and April of 2009. Figure 25 shows the rates at which nodes are discovered with *nswalk*. The data shown in this figure represents a period in which the tool was run without interruption. In the experiment, this form of enumeration discovered nodes rapidly but did not sustain this rate of unique node discovery over time. One can surmise that not all Repeater nodes participate in the fast-flux scheme. A comparison of the coverage between the network crawlers and nswalk is shown in Table 3.



(a) Newly discovered nodes per day. (b) Aggregate.

Figure 25: Enumeration via fast-flux DNS exploitation using the modified *nswalk* tool. A single domain, *whocherish.com*, was used to initialize this enumeration attempt. Newly discovered nodes by day and nodes discovered in the aggregate are shown.

[1]Crawler data only. Nodes are deemed as active when they correctly respond to at least 1 query by the crawler (i.e. the nodes are accessible and legitimate at least once during the deployment of the crawlers). This is not indicative of botnet size at a single point in time.

[2](Intersection / Union) of the data sets.

Table 3: Final IP address enumeration results for both *walleyworld* instances and*nswalk*.

	Total Discovered	***Excluding*** **Bogon IP Addresses**	***Excluding Inactive***[1] ***Nodes***
walleyworld	110500	97336	97329
nswalk	6924	6920	6920
Overlap[2]	5.89%	6.68%	6.68%

5.2.3 Walowdac: An Active Infiltration Tool

An active enumeration tool was developed by Ben Stock from the University of Mannheim to discover infected hosts in Waledac's lowest two tiers. His tool, *Walowdac*, emulated the behavior of a Repeater node, but rather than forwarding data requests to other nodes in network, instances of the tool replied directly with manipulated data. Walowdac nodes responded to inbound requests for peer data with their own network addresses and hash IDs to increase their popularity within the botnet and increase the probability that they would be contacted by infected hosts. Inbound traffic from other nodes in the botnet was logged, allowing the tool to enumerate bots functioning as Repeaters and Spammers.

Instances of Walowdac were deployed at several universities, and an 8-day time overlap of this deployment exists with the UTS log data. Table 4 documents the daily number of unique infections (hash IDs) visible in each data set and the overlap, or percentage of peers found in the intersection of the two sets of data out of their union.

The high daily overlap with the total population (viewable via the UTS logs) demonstrates the success of infiltration-based measurement in discovering infected

Table 4: Comparison of the data obtained from Walowdac and the UTS population logs for a period of 8 days showing unique hash ID counts and set intersection percentages.

	12/15	**12/16**	**12/17**	**12/18**	**12/19**	**12/20**	**12/21**	**12/22**
UTS Logs	33930	27179	37329	35425	35602	33463	32173	46929
Walowdac	23494	24337	27389	28379	28273	27164	27488	35492
Overlap	69.24 %	89.54 %	73.37 %	80.11 %	79.41 %	81.18 %	85.44 %	75.63 %

hosts. For the Waledac architecture, this technique also maintains an advantage over fast-flux exploitation methods and network crawlers as these cannot enumerate nodes outside of the single-tier peer-to-peer segment of the botnet. As analysis of the UTS logs indicates that a significant percentage of the botnet is comprised of infections in the Spammer layer, enumeration using infiltration-based methods is necessary for mitigation. Deployment of this tool, however, did require precise emulation of a legitimate Waledac infection.

5.3 Summary

This study of enumeration methods applied to varying architectures provides evidence that higher yields are produced when one can more completely emulate a botnet's communication protocol; when behavior is understand, it can be exploited. This knowledge comes at the expense of time, however. Such tools are also likely sensitive to protocol changes and would not be characterized as robust. The methods include network crawling and infiltration-based monitoring.

Conversely, enumeration methods that are rapidly deployable and continue to discover hosts in the face of protocol changes tend to lack the ability to enumerate completely. Darknet monitoring and fast-flux DNS exploitation are examples of these methods. Figure 26 suggests a reasonable deployment path for enumeration tools with consideration to time and yield. Methods toward the left of this figure are rapidly deployable but return reduced yield. As one approaches the right, yield is increased. The final “method” of gaining access to a main command and control server is included for completeness.

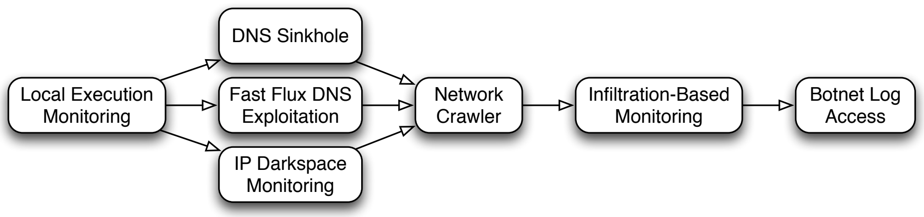

DNS Sinkhole
Local Execution Monitoring
Fast Flux DNS Exploitation
IP Darkspace Monitoring
Network Crawler
Infiltration-Based Monitoring
Botnet Log Access

Figure 26: Reasonable deployment path for enumeration tools with consideration of time needed for implementation and probable yield.

CHAPTER 6: ACCURACY IN BOTNET POPULATION ESTIMATION

Estimating botnet sizes using raw IP address counts can lead to over- and under-counting due to network features such as DHCP, NAT, proxies, and VPNs, as well as commuting users with mobile computers. This chapter explores methods to discover NAT and DHCP boundaries in network populations and quantify the IP address inflation which results. An entropy-based measurement of botnet size is proposed, which can be used to describe a botnet's footprint or its active size.

6.1 An Entropy-Based Inflation Rate

Within a botnet population, let I represent the set of unique IP addresses observed H represent the set of infected hosts. While botnets are most commonly observable by only their IP addresses, in the case when both types of identifiers are available, a naive estimate of the IP inflation rate can be computed as a simple ratio of the two :

$$R_N(I, H) = \frac{|I|}{|H|}$$

For each individual j in a botnet population J (for example infected hosts, or IP addresses), let a_j be the volume of measured activity associated with j. Activity can be measured in different units for different botnets, for example, scanning rates, spamming rates, or counts of C&C *keep-alives*.

The naive inflation ratio lacks the ability to describe how distributed an infection is through this IP space. An entropy-based inflation ratio, however, does possess this capability. The *footprint distribution* p_J is defined as the relative proportion of activity a_j associated with each individual observed throughout the botnet's history, measured in the particular unit of observation:

$$p_J(j) = \frac{a_j}{\sum_{k \in J} a_k}$$

For a population J with footprint distribution p_J, the entropy $S(p_J)$ is defined as:

$$S(p_J) = -\sum_{j \in J} p_J(j) \ln[p_J(j)]$$

Entropy summarizes the footprint distribution according to its degree of uniformity. A footprint distribution that assigns all of its activity to a single individual would have entropy equal to 0. Conversely, a footprint distribution that uniformly allocates activity of $1/N$ to N unique individuals would have maximal entropy, equal to $\ln(N)$.

A measure of the IP inflation rate based on the footprint distributions p_I and p_H among IP addresses and individuals in a botnet is given by the calculation:

$$R_E(p_I, p_H) = \exp\left[S(p_I) - S(p_H)\right].$$

The difference $S(p_I) - S(p_H)$ is also known as the *relative entropy* or Kullback-Leibler divergence [17] of the two distributions. R_E can be calculated for entire botnet histories, or for subsets of activity pertaining to periods of time or locations in space. Calculated across an entire botnet's history or smaller time increments, R_E is an extension of R_N that accounts for differences in both size and allocation of activity. $R_E = R_N$ when both p_I and p_H are uniform distributions. However, $R_E \neq R_N$ when random IP assignment (for example, from DHCP leases) spreads unequal activity from N hosts equally between N IP addresses.

6.2 Building Rate-Preserving Partitions

The value R_E can be calculated for any collection of IP addresses $I_\ell \subset I$ and the hosts H_ℓ that comprise all of the non-zero activity associated with I_ℓ, or vice versa. The IP space of a botnet, for example, can be decomposed by Autonomous System Number (ASN), country code, or CIDR block, to study the average IP inflation rate of each of these sub-populations. The botnet can also be decomposed into individual hosts, to study the average individual IP inflation rate per unique infection.

Each of these collections is a *partition* of its corresponding set (I or H). The union of these overlapping sets is equal to the entire botnet population. Further, the sum of all *activity* across any of these partitions comprises all of the *activity* of a botnet. However, the weighted average of R_E will not sum to the overall IP inflation rate of the botnet, unless the original partitioning sets are carefully chosen. This research must therefore pursue a method for preserving inflation rates in the creation of partitions to avoid creating overlapping sets.

Let a_{hi} be the volume of activity between individual h and IP address i. For botnet populations H and I of individual hosts and IP addresses, define the undirected graph G with vertices $V = \{H \cup I\}$ and edges $E = \{a_{hi} : h \in H, i \in I\}$. Any sub-graph $G_\ell \subset G$ contains a set of vertices $V_\ell = \{(H_\ell \subset H) \cup (I_\ell \subset I)\}$ and a set of edges $E_\ell = \{a_{hi} : h \in H_\ell, i \in I_\ell\}$ of non-zero activity between H_ℓ and I_ℓ that induce footprint distributions p_{H_ℓ} and p_{I_ℓ}. For simplicity, the IP inflation rate of these elements of G_ℓ is denoted as:

$$R_E(G_\ell) = R_E(p_{I_\ell}, p_{H_\ell}).$$

Let a_ℓ be the sum of all activity a_{hi} associated with a sub-graph G_ℓ, and define $a_L = \sum_\ell a_\ell$ as the total activity for a botnet. The weighted IP inflation rate of any sub-graph G_ℓ is defined as $\frac{a_\ell}{a_L} R_E(G_\ell)$. A *rate-preserving* collection of sub-graphs of G is defined as any collection that satisfies the following equality:

$$\ln R_E(G) = \sum_\ell \frac{a_\ell}{a_L} \ln R_E(G_\ell).$$

If a path of non-zero-valued edges exists that join v_1 to v_2, the vertices v_1 and v_2 in G are *connected.* The graph G can be partitioned into *strongly connected components*, which are disjoint sub-graphs where all vertices within each sub-graph are connected, and no vertices between sub-graphs are connected. The set of vertices and edges associated with each strongly connected component is referred to as an *equivalence class.* The collection of sets of vertices corresponding to each of the strongly connected components of G is denoted by $\{V\}_{\text{SCC}}$.

The *sigma algebra* $\sigma(\{A_\ell\}_1^L)$ of a collection of sets is defined as the set of all unions, intersections, and complements of sets in the collection. When $\{A_\ell\}_1^L$ is a partition, $\sigma(\{A_\ell\}_1^L)$ is equal to the union of the empty set $\emptyset$ and the power set of all unions of subsets A_ℓ. Using these definitions, the relationship between strongly connected components and the IP inflation rate is summarized in the following theorem:

Theorem 1 (rate-preserving sub-graphs) *Suppose that the collection of sets $\{H_\ell\}_1^L$ form a partition of the hosts H in G. Let $\{G_\ell\}_1^L$ be the sub-graphs of G obtained by* *taking $I_\ell \subset V_\ell$ as the set $\{i : a_{hi} > 0$ for at least one $h \in H_\ell\}$. Then the following inequality holds:*

$$\ln R_E(G) \leq \sum_\ell \frac{a_\ell}{a_L} \ln R_E(G_\ell),$$

with equality occurring if and only if $\sigma(\{H_\ell\}_1^L) \subset \sigma(\{V\}_{SCC})$.

Theorem 1, with a proof in Appendix A.3, holds with the inequality in the opposite direction for sub-graphs induced on H associated with any partition of I. The theorem states that the only rate-preserving collections of sub-graphs of G that partition one population, and examine all non-zero activity of this partition among the other population, are collections whose vertices consist of unions of strongly connected components. This result implies that any study of IP inflation rate across sub-networks of a botnet should use equivalence classes as the basic "building blocks" of activity.

6.3 Finding Subnets via Localized Component Clustering

The equivalence classes of G cluster individuals and IP addresses into related networks of activity. Large rates of $R_E(G_\ell)$ or large IP entropy $S(p_{I_\ell})$ within an equivalence class can indicate the presence of a region experiencing DHCP churn. Assuming transitivity of relationships between hosts and IP addresses, the IP addresses in each equivalence class represent likely points of communication (formerly held addresses) for any of the associated hosts in the class, weighted by observed activity within the sub-graph. Unfortunately, host ID collisions, erroneous ASN mappings, or hosts using open proxies, VPN connections, or spoofed IP addresses can link clusters of IP addresses into equivalence classes across ISPs, ASNs, or even countries. To mitigate these effects in mapping DHCP regions, a localized version of connected component clustering is employed as follows:

1. For each host $h \in H$, the *modal ASN* M_h of h is defined as the set of IP addresses in the autonomous system through which the largest proportion of h's activity a_h occurred.

2. Remove from G any edge (h, i) such that $i \notin M_h$.

The resulting intra-ASN equivalence classes that are produced are more interpretable as DHCP regions belonging to a single network provider. The set P of pruned edges is denoted as the *inter-ASN network* and the resulting pruned graph is referred to as G_P. It must be noted that restricting equivalence classes to reside only within ASNs does sacrifice some explanatory power for interpretability. The *weight* $W_E(P)$ of the inter-ASN network as the ratio of IP inflation ratesis defined as:

$$W_E(P) = \frac{R_E(G)}{R_E(G_P)}.$$

The intra-ASN equivalence classes are rate-preserving for G_P, but they cannot account for the weight of P. A ratio $W_E(P)$ close to 1 indicates that the pruned edges have little effect on the overall IP inflation rate of the botnet. However, if $W_E(P) = R_E(G)$, this indicates that P contains all inter-connectivity in G.

6.4 Application: Waledac

This section describes the result of the above techniques applied to the Waledac botnet to learn more about its IP inflation rate from the presence of NAT and DHCP regions.

6.4.1 Data Collection

These analysis techniques were applied to data gathered from the command and control server in the Waledac botnet which provides total network coverage. To perform this data collection, specialized `HTTP GET` requests for logs were sent to a node in the second highest layer, the *TSL* tier, which were forwarded to a single node in the *UTS* tier. Logs maintained on this system record a variety of inbound bot node messages. Requests were only forwarded when the *User-Agent* contained the string `LMK`. The `nginx` proxy on the TSL tier controls this access. The retrieval method is shown in Figure 11 in Section 4.2.6. Several log files, `1-first.txt`, `2-notify.txt`, and `4-words.txt` contained timestamps, IP addresses, and unique hash IDs for each node reporting or requesting data. Population data found in these logs documents activity from both of Waledac's infected host tiers.

Logs were retrieved during the period of December 4th through December 22nd, 2009. ASNs and approximate latitude and longitude coordinates were compiled for each IP address using the Maxmind GeoLite ASN [19] and City [20] databases. A total of 44412486 log-ins from infected hosts were observed in the UTS logs, comprising 172283 unique hashes and 667033 unique IP addresses.

The graph associated with the complete, unfiltered set of traffic is referred to as G_F. Any hash that belongs to an equivalence class with only itself and one unique IP address is called a *singleton* (or *singleton pair* in reference to both hash and IP address). Any hash that belongs to an equivalence class with multiple hashes and only a single IP address (as would occur with network address translation) is called a *static sharer*. Any hash seen communicating through multiple IP addresses is referred to as a *mobile hash*.

6.4.2 Data Filtering to Remove Chaff

To discover proportions of bot nodes in dynamically allocated IP space, population data must first be carefully cleaned to remove anomalous log-ins from illegitimate hosts or other spoofed traffic.

Two sub-graphs of G_F were used in the analysis. A filtered graph $G_L \subset G_F$ was created to remove anomalous, illegitimate traffic. Then, localized component clustering was used to create an inter-ASN graph $G_P \subset G_L$, in order to study sub-networks of related activity within ASN.

6.4.2.1 Filtering Aliases

Within the Waledac UTS data, several similarly composed hashes were discovered which were mapped to an unusually diverse set of IP addresses. Several of these hash IDs appear to be "man-made," and are not the result of *natural* (sufficiently random) or *expected* (determined through reverse-engineering) execution and activity.

This activity is possible as the Waledac network was open to monitoring, infiltration, and the manipulation of hash IDs for research or monitoring purposes. An *alias* is defined as a hash that appears to represent multiple infected hosts, multiple alternative hashes, spoofed traffic, or a reconnaissance or data-gathering process initiated by either researchers or botnet administrators. Any study of the IP inflation rate for Waledac should not include aliases, as even a relatively small number of aliases can have a large effect on the botnet's footprint (both R_N and R_E). A suspect hash was excluded from G by removing all log-ins associated with the hash. A set of 45 hashes, totaling 8% of all log-ins, were excluded from G, with the remaining 92% of log-ins comprising G_L.

A set of 18 hashes seen generating traffic from IPv4 bogon addresses [39] were excluded as aliases of spoofed traffic. As these packets should not be found on the public Internet, they are assumed to be illegitimate. A set of 11 hashes communicating through IP addresses belonging to the University of Mannheim were also excluded as aliases of concurrent research efforts. Communication with a researcher at this university confirmed that this traffic was the result of several experiments and was not produced by legitimate Waledac bots.

To discover aliases for other spoofed addresses or multiple infections, each mobile hash h was assigned a *mobility score* to quantify how aggressively it moved through physical IPv4 space. Let $t_1, \cdots, t_K$ be a hash's ordered log-in timestamps, and $c_1, \cdots, c_K$ be its associated latitude and longitude co-ordinates. The mobility score δ measures the average speed (in miles per hour) with which a hash changes locations in its log-ins:

$$\delta = \frac{1}{K-1} \sum_{k=2}^{K} \frac{\Delta(c_k, c_{k-1})}{t_k - t_{k-1}},$$

where $\Delta(\cdot)$ is the Haversine [43] or great-circle distance between the two locations:

$$haversin(\tfrac{d}{R}) = haversin(\varphi_1 - \varphi_2) + cos(\varphi_1)cos(\varphi_2)haversin(\Delta\lambda)$$

Where $haversin(\theta) = \frac{1}{2}(1 - cos(\theta))$

d = great-circle distance between points

R = sphere radius

φ_1 = latitude, point 1

φ_1 = latitude, point 2

$\Delta\lambda$ = longitudinal separation between points

A mobility score can be interpreted as the estimated average speed (in miles-per-hour) a system would be required to physically move throughout its life for it to indeed be a single, unique infection. While certain network features such as proxies and VPNs allow for rapid movement in IP space which can correspond with large perceived geographic distances, an assumptions is made that rapid oscillation between addresses for long periods of time, which would inflate the mobility score greatly, occurs rarely. Also, while mobile computing devices may rove as their owners travel, the mobility scores should reflect the physical limitations of this kind of movement. Outlying mobility scores are evidence of hashes aliasing multiple infected hosts that check in concurrently from wide-spread locations.

This process was implemented in Ruby to create `geoFilter`, which is reproduced in Appendix 7.2.

The results from this tool when applied to the UTS population data are shown in Figure 27. Hash IDs with unusually high mobility scores are identifiers deemed to be non-unique or illegitimate, and are present due to various phenomenon discussed previously.

Figure 28 shows a histogram of mobility scores on the log scale, with curves indi-

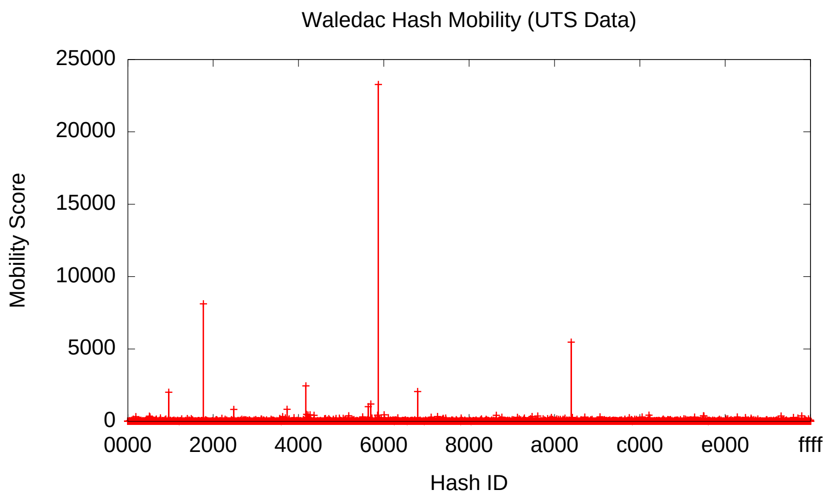


Figure 27: Mobility scores for hash IDs found in Waledac population data.

cating the best mixture of Normal distributions that fit the data [9]. Two groups are apparent; 89% of hashes move on average less than 10 miles per hour, while 11% of hashes appear more mobile. The top 10 scores are marked by solid black lines. Dashed lines mark the upper percentiles of 0.001, 0.0001, and 0.00001. An outlier analysis based on Monte Carlo simulations of this distribution flagged hashes with the top three mobility scores (23272.11MpH, 8114.45MpH, and 5471.55MpH) as anomalies.

An additional 21 hashes were flagged as suspected aliases for the three mobility outliers due to shared substrings in hash IDs (for example,

"990024015<u>e300d441c49064db74d4715de54b3</u>3912" and

"3e0f99002401<u>5e300d441c49064db74d4715de54b3</u>")

One can obtain these hash IDs by appending and prepending a small number of characters to the three mobility score outliers. This overlapping behavior, which is not found with any other of Waledac's hash IDs is shown in Figure 29. These hashes also do not possess the characteristics expected of hashes generated by Waledac's hash algorithm, as described in Section 4.2.3.4.

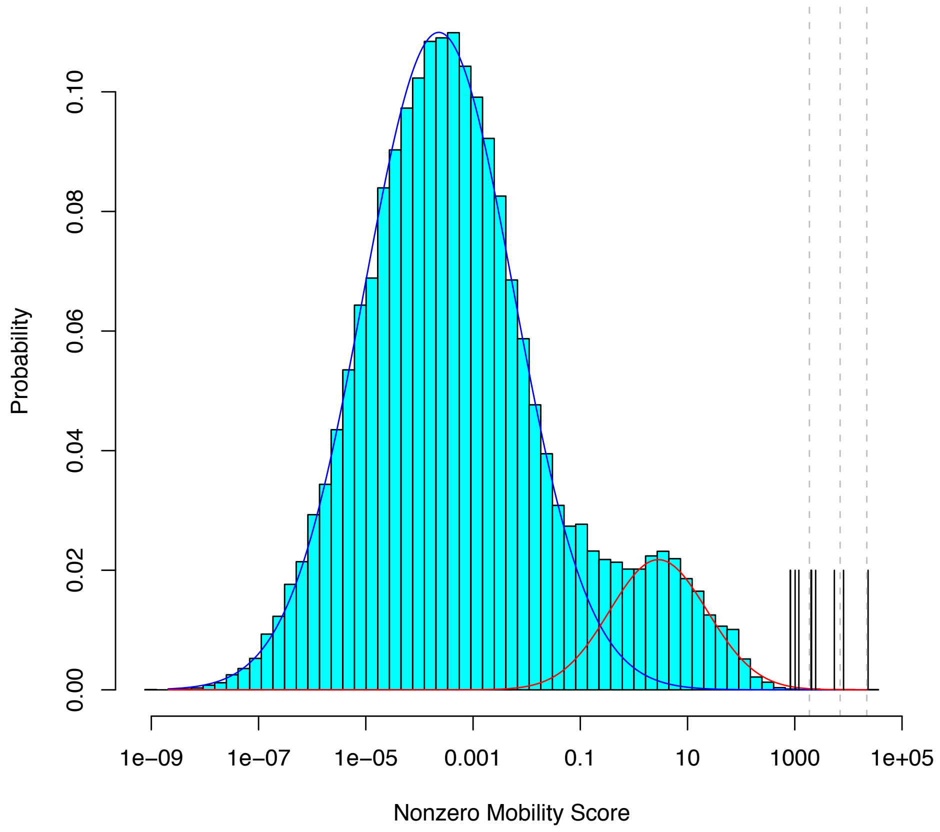


Figure 28: Probability histrogram and bimodal log-normal model for hash mobility scores.

6.4.3 Results

6.4.3.1 Initial Data Summary: G_F, G_L and G_P

Python scripts were developed to produce network equivalence classesand perform the entropy calculations. Table 5 summarizes the IP address counts, hash counts and IP inflation rates for the three graphs of Waledac log-insdescribed previously. The effect of aliases can be seen in both R_N and R_E, but it is more pronounced in R_E, which reduces by a factor of 2 when the aliases are removed. This is the effect of the top mobility score outlier, a product of spoofed network traffic, that comprised

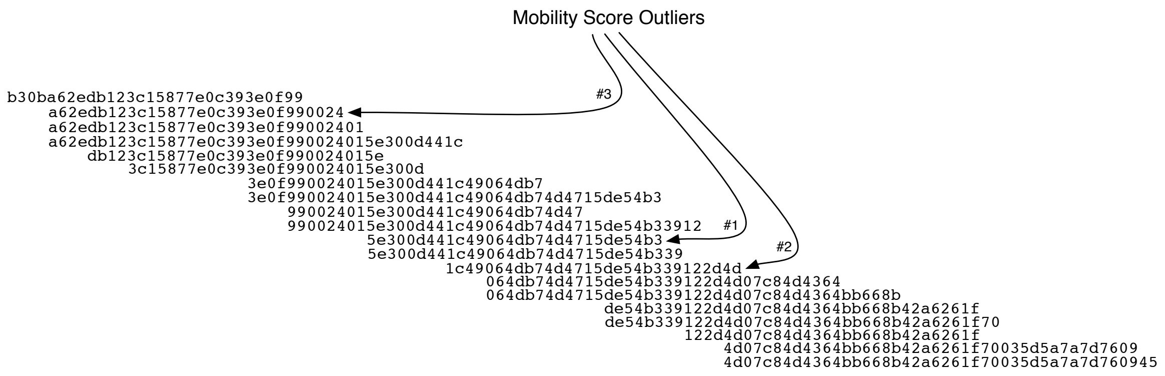

Mobility Score Outliers
#3
#1
#2
b30ba62edb123c15877e0c393e0f99
a62edb123c15877e0c393e0f990024
a62edb123c15877e0c393e0f99002401
a62edb123c15877e0c393e0f990024015e300d441c
db123c15877e0c393e0f990024015e
3c15877e0c393e0f990024015e300d
3e0f990024015e300d441c49064db7
3e0f990024015e300d441c49064db74d4715de54b3
990024015e300d441c49064db74d47
990024015e300d441c49064db74d4715de54b33912
5e300d441c49064db74d4715de54b3
5e300d441c49064db74d4715de54b339
1c49064db74d4715de54b339122d4d
064db74d4715de54b339122d4d07c84d4364
064db74d4715de54b339122d4d07c84d4364bb668b
de54b339122d4d07c84d4364bb668b42a6261f
de54b339122d4d07c84d4364bb668b42a6261f70
122d4d07c84d4364bb668b42a6261f
4d07c84d4364bb668b42a6261f70035d5a7a7d7609
4d07c84d4364bb668b42a6261f70035d5a7a7d760945

Figure 29: Overlapping hashes with the three mobility score outliers. This behavior is unique to this set of hashes and is not found in the rest of the occupied hash ID space.

nearly 6% of activity in G_F, across 89686 IP addresses. Accounting for distribution of activity in G_L reduces the inflation effect of measuring an IP footprint versus a hash footprint, as shown by the reduction in R_E vs. R_N. The weight of the inter-ASN network P is $(2.27/2.00) = 1.135$. This seems to be a reasonable tradeoff of explanatory power versus interpretability of intra-ASN equivalence classes.

Table 5: Naive and entropy-based IP inflation rates calculated for three different sub-graphs of the Waledac botnet.

G	$\lvert I\rvert$	$\lvert H\rvert$	a_G/a_{G_F}	R_N	R_E
G_F	667033	172283	1.00	3.87	4.56
G_L	548997	172238	0.92	3.18	2.27
G_P	475665	172238	0.86	2.76	2.00

A majority of hashes in G_L (63.6%) are associated with only a single IP address. Singleton pairs comprise 55.9% of hashes and 17.5% of all IP addresses in G_L. Singleton pairs and static sharers comprise nearly two thirds of legitimate hashes, but only 18.5% of all observed IP addresses. The top 1% of hashes in G_L are very mobile, however, are associated with 80 or more IP addresses, with a maximum of 428 IP addresses observed for a single hash. The mobile hashes in G_L comprise only 36.3% of all hashes, but communicate through 81.4% of observed IP addresses. The distribution of the number of unique IP addresses for each hash is shown in Figure 30. The sub-graphs in G_P are used to examine the breakdown of IP allocation in greater detail, with the explicit goal of determining the boundaries of DHCP regions which contribute to this result.

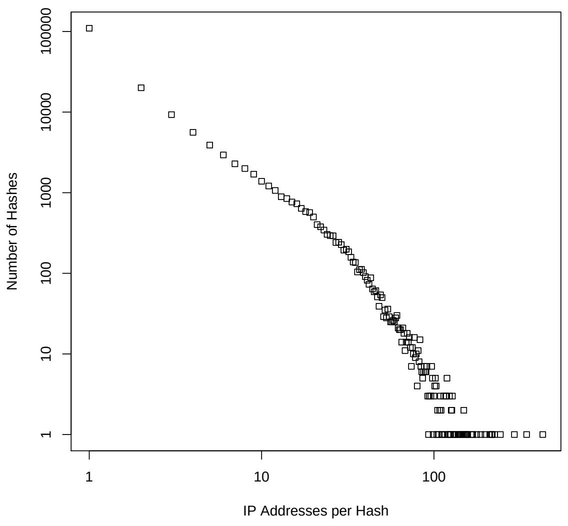


Figure 30: Distribution of the number of unique IP addresses observed per hash in G_L.

Figure 30 shows the distribution of the number of unique IP addresses observed per hash for G_L.

6.4.3.2 Equivalence Class Topology in G_P

Figure 31 is a plot of IP entropy vs. Hash entropy for the 153734 equivalence classes that comprise G_P. The units are shown on the exponentiated scale to display an "Effective Population Size" for each equivalence class, since $\exp[S(p_j)] = N$ when p_j

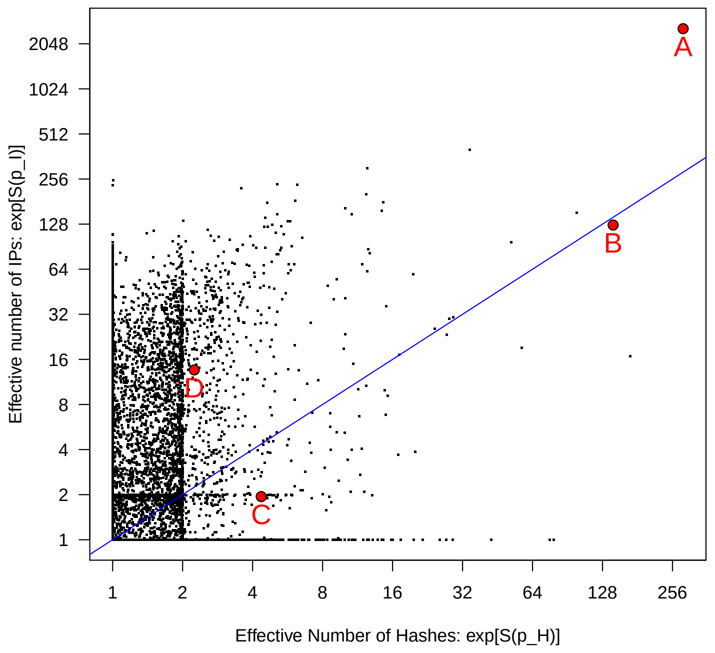


Figure 31: "Effective Size" $\exp[S(p_j)]$ of IP Addresses vs. Hashes for equivalence classes in G_P.

is uniform across N items. The $y = x$ line marks the subnetworks for which $R_E = 1$, with inflation growing in severity toward the top left of the chart, and deflation growing in severity toward the bottom right. NATs appear along the horizontal line $\exp[S(p_I)] = 1$. The largest intra-ASN NAT contained 128 hashes, though an unequal activity distribution among hashes increased the IP inflation rate from 1/128 to approximately 1/79. The majority of equivalence classes appear above the equality line, indicating $R_E > 1$. The largest DHCP effect for a single hash (appearing along the vertical line $\exp[S(p_H)] = 1$) corresponds to the largest mobile hash in G_L, which was assigned its own equivalence class with 428 unique IP addresses and an inflation rate of 233/1.

Four additional equivalence classes(A,B,C, and D) are highlighted on the graph

Table 6: Naive and entropy-based IP inflation rates calculated for four equivalence classes.

G	$\|I\|$	$\|H\|$	a_G	R_N	R_E
A	6789	438	317435	15.50	9.08
B	145	533	119684	0.27	0.89
C	5	5	296	1.00	0.45
D	16	16	1746	1.00	6.06

for further illustrative analysis. The characteristics of these equivalence classes are explored in the next section.

6.4.3.3 Exploring Four Types of Networks

Table 6 summarizes the four equivalence classes within G_P that are highlighted in Figure 31. Activity sizes are shown as the number of log-ins observed for each equivalence class. Figure 32 shows activity profiles of each equivalence class, plotting the ordered weights $p_{J_\ell}(j)$, for both hashes (dotted lines) and IP addresses (solid lines). Plots are shown on log scales to balance the effect of relatively large proportions assigned to the top few individuals versus small proportions spread among large numbers of individuals.

Class A was chosen for inspection as it has the largest number of IP addresses assigned for any equivalence class in G_P. The top IP address and top hash in this class each received approximately 1.5% of activity in A, but p_I is much more widespread in the tails than p_H, contributing to an inflation rate of 9 to 1 even accounting for activity. Class A is a member of a Saudi Arabian ASN that appears to be an ISP with a very large DHCP pool. It is possible that the ASN mappings for this region,

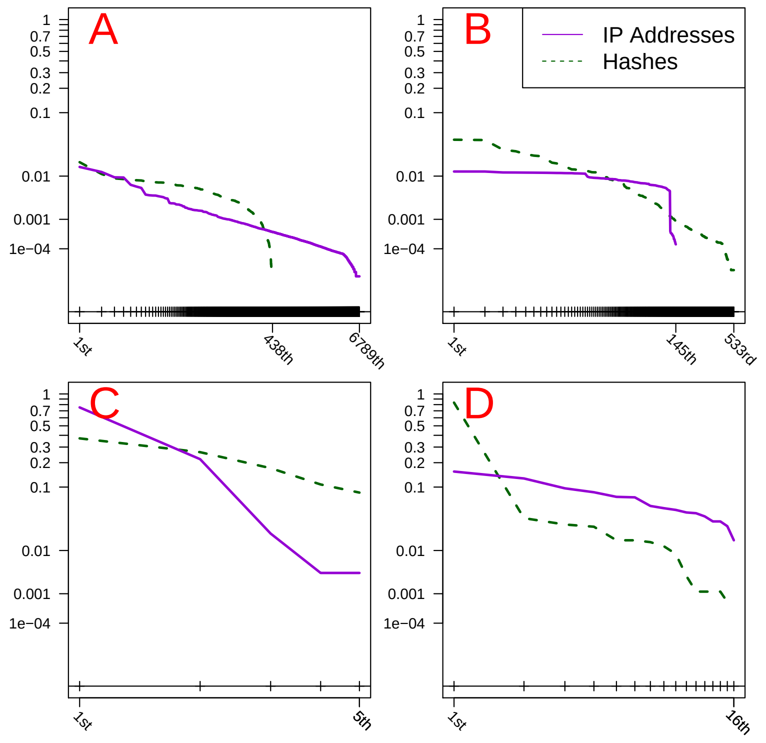


Figure 32: Activity profiles of four equivalence classes.

which relies heavily on satellite ISPs, are erroneous. In total 1861 equivalence classes belong to this ASN, all with overlapping IP ranges, which suggests that they are part of the same DHCP network. The overall IP inflation rate R_E of the union of these 1861 classes is 7 to 1.

Class B has the largest number of hashes assigned for any equivalence class in G_P. Despite having 3.67 times as many hashes as IP addresses, the entropy inflation R_E of B is only slightly under 1. As seen by the activity profile, B's unequal hash activity is spread much more evenly across its available IP addresses, suggesting a DHCP pool with a short lease. Class B is one of only two equivalence classes in its ASN, assigned to a broadband provider in the United States. Non-overlapping IP ranges suggest that the two equivalence classes represent two differently leased sub-networks.

Classes C and D highlight the effect of differences in allocation of activity on the entropy-based inflation rate R_E. For each of these classes, the number of IP addresses is equal to the number of hashes, however the activity profiles differ considerably. Class C shows a network with 5 relatively active infected hosts that appear to check in mostly with a single IP address (75% of activity). Class C is one of 672 overlapping equivalence classes, with a overall inflation rate of 6 to 1 among the union, within a large telecom company based in Vietnam. C could possibly represent a static gateway that occasionally changes location. Class D, rather, appears to spread activity from one very active hash (89% of logins) and fifteen hashes which are more ephemeral evenly among a DHCP pool of 16 addresses. D belongs to an ASN that contains 5 other equivalence classes, 4 of which are singletons, and another that also appears to be a DHCP pool belonging to the same provider as D.

6.4.3.4 Time-Based Inflation

Inflation rates can also be viewed in cumulative daily totals. This offers insight into the amount of inflation that can be expected as a population is enumerated This daily inflation is shown in Figure 33. Inflation rates for Spammer and Repeater plotted separately. It can be theorized that Repeater nodes bear lower inflation rates than infected hosts in the Spammer tier, as these nodes have public IP addresses and may not be subjected to as much network movement as hosts in NAT blocks, as these NAT blocks may rove behind multiple public addresses. From this data, this notion is confirmed. Nodes in the Repeater tier produce less inflation over time.

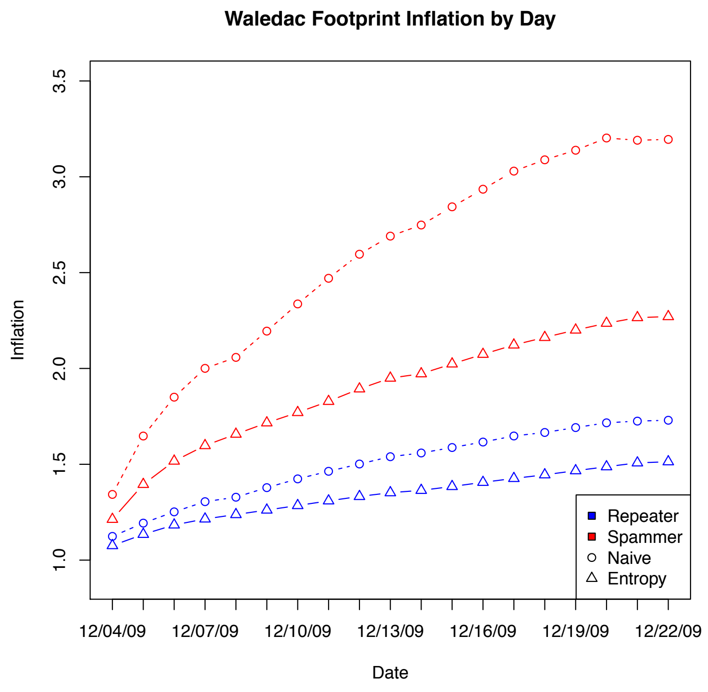


Figure 33: Daily cumulative inflation for the Waledac population. Naive and entropy-based rates are shown.

6.5 Summary and Applicability

Observable relationships between IP addresses and infected hosts provides insight not only into the particular botnet, but also into the networks that compose it. Although the type of infection may change, the characteristics of infected network space dictated by administrators are relatively consistent, including DHCP regions, NAT policies, and throughput rates. Equivalence classes constructed from observable botnets such as Waledac can be used to map out the boundaries of shared IP address pools and to profile network properties such as load-balancing thresholds and DHCP lease times among these networks.

This information can be utilized in the study of less visible botnets, such as Conficker, by using measured network profiles to adjust and refine population models based on behavior. Not only can one transfer information directly between networks that house both kinds of infections, statistical models can be used to extrapolate this information and infer the unobservable properties of a hidden botnet across its footprint. Finally, measuring a footprint distribution as opposed to direct counts can help to determine the potential threat of a botnet's available assets, either infected hosts or IP addresses.

CHAPTER 7: CONCLUSIONS

This chapter reiterates and clearly defines the contributions of this dissertation work and discusses future research to extend these findings. The contributions are documented in Section 7.1, and potential future work is delineated in Section 7.2.

7.1 Contributions

In an effort to organize and document existing and new botnet enumeration techniques, Chapter 3 presented 7 methods to discover infected hosts participating in malicious networks. Tools designed to implement several of these conceptual methods, including *infiltration-based monitoring*, *network crawling*, and *fast-flux DNS exploitation* were developed and deployed to enumerate the Storm and Waledac botnets. Their coverage and performance was characterized in Chapter 5, and a deployment strategy was proposed for these tools and methods to optimize yield given the slow discovery of botnet protocols which allow for more rapid and complete enumeration.

The architectures, communication protocols, and malicious output for the Storm and Waledac botnets were presented in Chapter 4. These botnets were operational at the time of their study, and their topologies, protocols, and other behaviors were unknown. The findings for Waledac's architecture are particularly insightful, as network traces and file-system data from botmaster-deployed systems were inspected to document the back-end infrastructure. The study of this infrastructure revealed previously unknown deployment methodologies, hidden defensive techniques, the botnets economic model, and new ways to leverage infected hosts. These findings impacted enumeration techniques and remediation strategies. Additionally, complete population logs gathered from a top-tier command and control server allowed for the rigorous study of its population characteristics. This research contributes toward the overall understanding of modern botnet architectures.

Finally, a method to measure IP inflation based on relative entropy across IP addresses and unique machine IDs was developed and presented in Chapter 6. Unlike a naive ratio of IP addresses to unique identifiers, entropy is capable of describing a botnet or network footprint distribution according to its uniformity. Rate-preserving partitioning of a botnet's historical footprint was demonstrated, along with how to construct connected sub-networks that highlight DHCP regions and NAT blocks. These methods were applied to Waledac population data, where entropy rates for a total footprint and cumulative daily time-slices were discovered. Four types of networks within Waledac were scrutinized in an attempt to present how inflation rates can vary within a botnet given different network structures. In total, this network characterization constitutes work toward pursuing accuracy in botnet size and threat estimation.

7.2 Future Work

The methods developed to enumeration botnets can be applied to other botnet structures, and their performance more quantitatively explored. Further, while the enumeration techniques presented in this study are applicable to current botnet architectures, evolutionary changes in botnet structures will likely necessitate the creation of new techniques to discover hosts. Changes in communication protocols or schemes, deployed services, and defensive mechanisms have the potential to impede host discovery.

The Storm and Waledac botnets represent two successful architectures capable of generating profit. The botnets are believed to be of the same lineage, and while both are no longer operational, a new botnet which bears a strikingly similar architecture to Storm and Waledac was discovered in early January 2011. This new botnet, along with other future botnets with sophisticated architectures will also need to be inspected and documented to continue the development of remediation techniques.

Finally, accuracy in enumeration must continue to be explored, specifically with regard to the impact of DHCP churn and NAT blocks when counting raw IP addresses. Methods to define the boundaries of these network structures in botnet populations must be refined or developed, particularly when bot infections do not bear unique identifiers. The entropy-based inflation rate can be applied to other botnets to determine whether the inflation characteristics of Waledac hold true for other samples of network space. The Conficker malware family is a likely target for such research.

REFERENCES

[1] aMule. http://www.amule.org.

[2] ProxyChains. http://proxychains.sourceforge.net/.

[3] Webmoney. http://www.wmtransfer.com/.

[4] Abu Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. My Botnet is Bigger than Yours (Maybe, Better than Yours: Why Size Estimates Remain Challenging. In *The 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots 2007)* (April 2007).

[5] Calvet, J., Davis, C., and Bureau, P. M. Malware Authors Don't Learn, and That's Good. In *The Fourth Annual Conference on Malicious and Unwanted Software (Malware '09)* (October 2009).

[6] Cooke, E., Jahanian, F., and McPherson, D. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *The 1st USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI 2005)* (July 2006), pp. 39–44.

[7] D'Acunto, L., Pouwelse, J., and Sips, H. A Measurement of NAT and Firewall Characteristics in Peer to Peer Systems. In *Proceedings of the 15th Advanced School for Computing and Imaging Conference* (2009).

[8] Dagon, D., Gu, G., Lee, C., and Lee, W. A Taxonomy of Botnet Structures. In *The 23rd Computer Security Applications Conference (ACSAC 2007)* (December 2007).

[9] Fraley, C., and Raftery, A. E. Model-Based Clustering, Discriminant Analysis and Density Estimation. *Journal of the American Statistical Association 97* (2002), 611–631.

[10] Grizzard, J., V.Sharma, Nunnery, C., Kang, B., and Dagon, D. Peer-to-Peer Botnets: Overview and Case Study. In *First USENIX Workshop on Hot Topics in Understanding Botnets (HotBots 2007)* (April 2007).

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

[12] Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *Proceedings of the First USENIX Workshop on Large Scale Exploits and Emergent Threats (LEET 2008)* (April 2008).

[13] Kang, B., Chan-Tin, E., Lee, C., Tyra, J., Kang, H. J., Nunnery, C., Wadler, Z., Sinclair, G., Hopper, N., Dagon, D., and Kim, Y. Towards Complete Node Enumeration in a Peer-to-Peer Botnet. In *ACM Symposium on Information, Computer and Communication Security (ASIACCS 2009* (March 2009).

[14] Kanich, C., Kreibich, C., Levchenko, K., Enright, B., Voelker, G., Paxson, V., and Savage, S. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *The 15th Conference on Computer and Communications Security (CCS 2008)* (2008).

[15] Kanich, C., Levchenko, K., Enright, B., Voelker, G., and Savage, S. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *Proceedings of the 1st USENIX Workshop on Large Scale Exploits and Emergent Threats (LEET 2008)* (April 2008).

[16] Kreibich, C., Kanich, C., Levchenko, K., Enright, B., Voelker, G., Paxson, V., and Savage, S. On the Spam Campaign Trail. In *Proceedings of the First USENIX Workshop on Large Scale Exploits and Emergent Threats (LEET 2008)* (April 2008).

[17] Kullback, S., and Leibler, R. On Information And Sufficiency. *The Annals of Mathematical Statistics 22*, 1 (1951), 79–86.

[18] matlab. `http://www.mathworks.com/`.

[19] MaxMind. GeoLite ASN. `http://www.maxmind.com/app/asnum`.

[20] MaxMind. GeoLite City. `http://www.maxmind.com/app/geolitecity`.

[21] Maymounkov, P., and Mazières, D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *The 1st International Workshop on Peer-to-Peer Systems* (2002).

[22] Microsoft. GetTickCount Function. `http://msdn.microsoft.com/en-us/library/ms724408(VS.85).aspx`.

[23] Nazario, J., and Holz, T. As the Net Churns: Fast-Flux Botnet Observations. In *The Third Annual Conference on Malicious and Unwanted Software (Malware 2008)* (October 2008).

[24] Nunnery, C., Sinclair, G., and Kang, B. Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-Tier Botnet Infrastructure. In *The 4th USENIX Conference on Large Scale Exploits and Emergent Threats (LEET 2010)* (April 2010).

[25] Nychis, G., Sekar, V., Andersen, D., Kim, H., and Zhang, H. An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In *Proceedings of the ACM Internet Measurement Conference* (October 2008).

[26] Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *The 6th ACM SIGCOMM on Internet Measurement (IMC 2006)* (October 2006), pp. 41–52.

[27] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and Lear, E. Address Allocation for Private Internets. `http://tools.ietf.org/html/rfc1918`.

[28] Roelofs, G., and Adler, M. zlib. `http://zlib.net/`.

[29] Samosseiko, D. The Partnerka - What Is It, and Why Should You Care? In *Virus Bulletin Conference* (September 2009).

[30] Sinclair, G. Blog Post: Waledac's Communication Protocol. `http://www.nnl-labs.com/cblog/index.php?/archives/7-Waledacs-Communcation-Protocol.html`.

[31] Sinclair, G., Nunnery, C., and Kang, B. The Waledac Protocol: The How and Why. In *The Fourth Annual Conference on Malicious and Unwanted Software (Malware 2009)* (October 2009).

[32] Stewart, J. Storm worm DDoS Attack. `http://www.secureworks.com/research/threats/storm-worm/`, February 2007.

[33] Stewart, J. Protocols and Encryption of The Storm Botnet. `http://www.blackhat.com/presentations/bh-usa-08/Stewart/BH_US_08_Stewart_Protocols_of_the_Storm.pdf`, 2008.

[34] Stock, B., Engelberth, M., Freiling, F., and Holz, T. Walowdac - Analysis of a Peer-to-Peer Botnet. In *The 5th European Conference on Computer Network Defense (EC2ND 2009)* (November 2009).

[35] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmer, R., Kruegel, C., and Vigna, G. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *The 16th ACM Conference on Computer and Communications Security (CCS 2009* (October 2009).

[36] Stover, S., Dittrich, D., Hernandez, J., and Deitrich, S. Analysis of the Storm and Nugache Trojans - P2P is Here. *Login* (December 2007).

[37] Sudosecure. Waledac Tracker. `http://www.sudosecure.net/waledac/bmd5updatecycle.php`.

[38] Sysoev, I. nginx. `http://nginx.net/`.

[39] Team Cymru. The Team Cymru Bogon List. `http://www.team-cymru.org/Services/Bogons/`.

[40] Thonnard, O., Mees, W., and Dacier, M. Addressing the Attack Attribution Problem Using Knowledge Discovery and Multi-criteria Fuzzy Decision-making. In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics* (June 2009), pp. 11–21.

[41] Weaver, R. A Probabilistic Population Study of the Conficker-C Botnet. In *The 11th Passive and Active Management Conference (PAM 2010)* (April 2010).

[42] Weaver, R. Beyond the Top Talkers: Empirical Correlation of Conficker-C Infected IP Space. In *FloCon 2010* (January 2010). `http://www.cert.org/flocon/2010/presentations/Weaver_BeyondTheTopTalkers.pdf`.

[43] Weisstein, E. Haversine - MathWorld. `http://mathworld.wolfram.com/Haversine.html`.

[44] Xie, Y., Yu, F., Achan, K., Gillum, E., Goldszmidt, M., and Wobber, T. How Dynamic are IP Addresses? In *The 7th Conference of the ACM Special Interest Group on Data Communication Conference (SIGCOMM 2007)* (2007).

[45] Yu, T., Lippmann, R., Riordan, J., and Boyer, S. EMBER: A Global Perspective on Extreme Malicious Behavior. In *Proceedings of the ACM VizSEC* (September 2010).

APPENDIX A: SUPPLEMENTARY CODE, DATA AND THEOREMS

This appendix contains supplementary details related to the created tools and discovered protocol details. A proof for a mathematical theorem is also included.

A.1 Reproduced Code

This section includes reproduced code from tools developed during this study.

A.1.1 WalleyWorld

The WalleyWorld crawler was designed to discover nodes participating in the peer-to-peer layer of the Waledac botnet. Only Repeater nodes were enumerated using this tool. The decoding method from this tool is reproduced below.

```
def decode(message)
  message = message.tr('_-', '/+').gsub(/[\r\n\s]/,'')
  message += "=" while message.length \% 4 != 0
   data = Base64.decode64(message)
   offset = 0
   key = [0xEF, 0xC2, 0x25, 0xCD, 0x36, 0xBB, 0x77, 0xE3, 0x69, 0x1B,
       0xE0, 0x96, 0xC5, 0x40, 0xD8, 0x78].pack("C*")
   out = rijndael_decrypt(key, data[offset,1E9])
   BZ2::bunzip2(out)
   end
end
```

A.1.2 geoFilter.rb

The `geoFilter.rb` tool was designed to compute the Mobility Scores for Waledac infections. This metric is described in Section 6.4.2. Three code fragments are reproduced here: the Haversine function required for Great Circle distance calculation between latitude and longitude coordinates, the speed calculation, and the final Mobilty Score computation.

A.1.2.1 Haversine Function

```
 #haversine - find distance along a great circle
  def haversine_distance( lat1, lon1, lat2, lon2 )
    rpd = 0.01745329251994 #rad per degree; PI/180
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    dlon_rad = dlon * rpd
    dlat_rad = dlat * rpd
    lat1_rad = lat1 * rpd
    lon1_rad = lon1 * rpd
    lat2_rad = lat2 * rpd
    lon2_rad = lon2 * rpd
    x = (Math.sin(dlat_rad/2))**2 + Math.cos(lat1_rad) *
         Math.cos(lat2_rad) * (Math.sin(dlon_rad/2))**2
    y = 2 * Math.atan2( Math.sqrt(x), Math.sqrt(1-x))
    dMi = 3956.6 * y #radius of eath in miles * c = delta between points
    return dMi
  end
end
```

A.1.2.2 Speed Calculation

```
#masterset = array of hash clusters
until masterSet[0] == nil
  aSet = masterSet.pop()
  aMovement  = []
  aLoc = []
  aInd = 0
  $aDropped = 0
  aSet[2].each do |aSetEntry|
    aCord = gf.getCord(aSetEntry)
    if (aCord == nil) #unknown IPs
      $aDropped+=1
    else
      aLoc << aCord
      if (aInd!=0)
        #get estimated geographic distance between IPs,
        #divide by time between log entries, mult by 3600
        aDistChange = gf.haversine_distance(aLoc[aInd][0],aLoc[aInd][1],
            aLoc[(aInd-1)][0],aLoc[(aInd-1)][1])
        aTimeDiff = (aSet[1][aInd].to_f) - (aSet[1][aInd-1].to_f)
        if aTimeDiff == 0.0 #inject time diff for entries w/ same time
          aTimeDiff = 0.1
        end
        aMovement << 3600*(aDistChange.to_f/aTimeDiff)
      end
      aInd+=1
   end
end
```

A.1.2.3 Mobility Score Calculation

```
aMoveSum = aMovement.inject(0){|b,i| b+i}
aMovementLength = aMovement.length()
if aMovementLength == 0 #all available IPs were dropped; force a score of 0
  aMovementLength = 1
end
aMobScore = (aMoveSum / aMovementLength)
puts aSet[0] + "|"  + aMobScore.to_s + "|" + aMovement.length().to_s +
      "|" + $aDropped.to_s
```

A.2 Botnet Case Study Data

This appendix contains information related to the case-studies for the Storm and Waledac botnets. File-system contents, script contents, communication protocol specifics, supplementary population statistics, and a mathematical theorem as a reference for the reader.

A.2.1 Waledac Command and Reply Syntax

Tables 7 and 8 explicate the types, purpose, and syntax for Waledac's communication protocol. Note that in Table 7, the Command Names correspond with the types of logs discovered on the botnet's primary command and control server.

Table 7: Valid Waledac Command Request Types

Command Number	Command Name	Purpose	XML Attributes
0 (aliased as 0xFF)	getkey	Request is used to obtain the AES key for use in commands 1-7	<p n=cert>{ASCII form of node's public cert} </p>
1	first	Identifies the infected node's OS version and the "label" of the bot binary	<p n=winver> *(major OS version).(minor OS version).(OS subversion)*</p> <p n=label>*(mirabella_site or lynx)*</p>
2	notify	Request instructions from C&C for upcoming campaigns	<p n="label"> *(mairabella_site or lynx)*</p> <p n="time_sys"> *(current time in ASCII)*</p> <p n="time_init"> *(time node was activated in ASCII)*</p> <p n="time_now"> *(current time in ASCII)*</p> <p n="time_ticks"> *(current tickcount, converted to 64bit ASCII)*</p>
3	taskreq	Request spam campaign configuration	
4	words	Request meaning of variables used in spam templates obtained from *taskreq*	<p n="word_name">*(word)*</p>
5	taskrep	Report campaign details (e.g. spam sending success, which email addresses were spammed, etc.)	<props><p n="b64">true</p></props> <reports><rep id= "*(%d number)*" rcpt= "*(email address in Base64 encoding)*">*(status in base64 encoding)*</rep></reports>
6	httpstats	(Sent by repeaters only.) Internal HTTP access_log from phishing activity	<props><p n="b64">true</p></props> <http_stats><stat ip="*(ip address that made a request)*" time="*(time of request in UNIX decimal format)*"> <![CDATA[*(the URL request in typical W3C log format)*]] ></stat></http_stats>
7	emails	Report of email addresses found on the victims machine	<emails><![CDATA[*(emails, one per line)*]] ></emails>

Table 8: Waledac Bot Replies

Command Number	**Command Name**	**Purpose**	**XML Attributes**
0 (aliased as 0xFF)	getkey	Replies with AES key to use for commands 1-7 in an encrypted form	<p n=key>*(base64 encoded RSA encrypted AES key)*</p>
1	first	Empty acknowledgement	
2	notify	Instructions from C&C for upcoming campaigns and possibly node update URLs	<p n="ptr">*(RDNS of node)* </p> <p n="ip">*(IP of node)* </p> <p n="dns_ip"> *(DNS server to use)* </p> <p n="smtp_ip">*(SMTP server to test)* </p> <p n="sender_threads">*(the number of sender threads to activate at once)* </p> <p n="sender_queue"> *(Unknown purpose)*</p> <p n="short_logs"> *(Unknown purpose)*</p> <p n="http_cache_timeout"> *(Timeout for caching proxy/repeater data)*</p> <p n="commands">< [CDATA[*Command Values*[1]]] ></p> <dns_zones><zone>*(name of the DNS zone)* </zone> </dns_zones> <dns_hosts><host>*(IP address of DNS host)* </host> </dns_hosts> <socks5><allow max_conn= "*(maximum SOCK5 connections allowed)*"> *(IP address to allow)* </allow></socks5> <dos><target><ip>*(target IP address)* </ip><port> *(specifiy port)* </port><rate> *(flood rate)* </rate><rate2> *(Purpose unknown)* </rate></target></dos>
3	taskreq	Spam campaign configuration	<tasks><task id="*(number presenting the task identifier as more than one task can be)*"><body>*(body of email message in Base64 encoding)*</body><a> *(email address to spam)*</a><w>*(words/variables entry)*</w></task></tasks> <words><w name="*(word)*" time= "*(timestamp)*"/></words>
4	words	Meaning of variables used in spam template obtained from *taskreq*	<word name="*(word)*"><![CDATA[*(series of words that define the (word), one per line)*]] </word>
5	taskrep	Empty acknowledgement	
6	httpstats	Empty acknowledgement	
7	emails	Empty acknowledgement	

A.2.2 Message types in the Overnet protocol

- `Connect` (`0x0a`): Used to bootstrap/join the network.
- `ConnectReply` (`0x0b`): A bootstrap peer will return back to the new node a list of other peers so that the new node can start to build its routing table.
- `Publicize` (`0x0c`): A type of "Hello" message to convey that a node is active.
- `PublicizeAck` (`0x0d`): Reply to the "Hello" message.
- `Search` (`0x0e`): To find a certain id or to maintain its routing table (basically a peer will send a Search message looking for itself, then it will know of other peers that are really close to itself in the DHT).
- `SearchNext` (`0x0f`): The reply to a `Search` message which includes IDs and IP addresses of other peers.
- `SearchInfo` (`0x10`): After the node closest to the target ID is found, this request for results is sent.
- `SearchResult` (`0x11`): Reply to the `SearchInfo` containing the results for the search.
- `SearchEnd` (`0x12`): A reply indicating no data was found.
- `Publish` (`0x13`): Publish an IP address-ID binding or metadata relative to a particular possessed file.
- `PublishAck` (`0x14`): Acknowledgment that the `Publish` message was received.

A.3 Proof for Rate Preserving Theorem

This section provides a proof of Theorem 1 described in Chapter 6.

For $h, i \in G_\ell$, let $\lfloor a_h \rfloor_\ell = \sum_{i \in I_\ell} a_{hi}$, and $\lfloor a_i \rfloor_\ell = \sum_{h \in H_\ell} a_{hi}$. The relative entropy of any sub-graph, including G itself, can be re-written as:

$$\frac{1}{a_\ell}\left(\sum_{h \in H_\ell} \lfloor a_h \rfloor_\ell \ln \lfloor a_h \rfloor_\ell - \sum_{i \in I_\ell} \lfloor a_i \rfloor_\ell \ln \lfloor a_i \rfloor_\ell\right). \tag{1}$$

Let $\{H_\ell\}_1^L$ be the set of host vertices associated with each strongly connected component in $\{V\}_{\text{SCC}}$. Because hosts are directly connected only to IP addresses, the union of H_ℓ and the set $I_\ell = \{i : a_{hi} > 0 \text{ for some } h \in H_\ell\}$ is equivalent to set $V_{\text{SCC}\ell}$. If $\{H_\ell\}_1^L$ is any union of sets in $\{V\}_{\text{SCC}}$ then for all individuals j, $\lfloor a_j \rfloor_\ell = a_j$ for some component ℓ and 0 elsewhere, which completes the equality when substituted in equation 7.2. If $\sigma(\{H_\ell\}_1^L) \not\subset \sigma(\{V\}_{\text{SCC}})$, then for some i and sub-graphs G_{ℓ_1}, G_{ℓ_2}, $a_i = \lfloor a_i \rfloor_{\ell_1} + \lfloor a_i \rfloor_{\ell_2}$, both non-zero, and the sum of relative entropy across subsets equals that of a graph with IP addresses $i_1 \in I_{\ell_1}$ with activity $\lfloor a_i \rfloor_{\ell_1}$, and $i_2 \in I_{\ell_2}$ with activity $\lfloor a_i \rfloor_{\ell_2}$ and thus a higher inflation rate than G. The proof for subsets of H induced by partitions of I follows the same argument.