Towards Complete Node Enumeration in a Peer-to-Peer Botnet

Brent ByungHoon Kang[1], Eric Chan-Tin[2], Christopher P. Lee[3], James Tyra[2],
Hun Jeong Kang[2], Chris Nunnery[1], Zachariah Wadler[1], Greg Sinclair[1],
Nicholas Hopper[2], David Dagon[3], and Yongdae Kim[2]

[1]{bbkang, cenunner, zawadler, gssincla}@uncc.edu
College of Computing and Informatics, University of North Carolina at Charlotte

[2]{dchantin, tyra, hkang, hopper, kyd}@cs.umn.edu
Computer Science Department, University of Minnesota

[3]{chrislee, dagon}@cc.gatech.edu
Georgia Tech Information Security Center, Georgia Institute of Technology

ABSTRACT

Modern advanced botnets may employ a decentralized peer-to-peer overlay network to bootstrap and maintain their command and control channels, making them more resilient to traditional mitigation efforts such as server incapacitation. As an alternative strategy, the malware defense community has been trying to identify the bot-infected hosts and enumerate the IP addresses of the participating nodes so that the list can be used by system administrators to identify local infections, block spam emails sent from bots, and configure firewalls to protect local users. Enumerating the infected hosts, however, has presented challenges. One cannot identify infected hosts behind firewalls or NAT devices by employing crawlers, a commonly used enumeration technique where recursive get-peerlist lookup requests are sent newly discovered IP addresses of infected hosts. As many bot-infected machines in homes or offices are behind firewall or NAT devices, these crawler-based enumeration methods would miss a large portions of botnet infections. In this paper, we present the Passive P2P Monitor (PPM), which can enumerate the infected hosts regardless whether or not they are behind a firewall or NAT. As an empirical study, we examined the Storm botnet and enumerated its infected hosts using the PPM. We also improve our PPM design by incorporating a FireWall Checker (FWC) to identify nodes behind a firewall. Our experiment with the peer-to-peer Storm botnet shows that more than 40% of bots that contact the PPM are behind firewall or NAT devices, implying that crawler-based enumeration techniques would miss out a significant portion of the botnet population. Finally, we show that the PPM's coverage is based on a probability-based coverage model that we derived from the empirical observation of the Storm botnet.

ASIACCS'09, March 10-12, 2009, Sydney, NSW, Australia.
Copyright 2009 ACM 978-1-60558-394-5/09/03 ...$5.00.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Infected Hosts Enumeration, P2P, Botnet

Keywords

Infected Hosts Enumeration, P2P, Botnet, Storm

1. INTRODUCTION

Botnets using a decentralized, peer-to-peer (P2P) communication architecture have become increasingly common in the arms race between botmasters and security practitioners. Since such P2P botnets do not rely on a centralized command and control (C&C) channel, they are more resistant to traditional mitigation strategies such as domain revocation, DNS redirection, and host-based blocking which focus on a single point of failure. This necessitates a search for new remediation tactics applicable to decentralized P2P botnets.

As there is no general technique for disabling a P2P network – many P2P protocols are designed specifically to resist large-scale denial of service attacks – a natural alternative is to focus on *identifying* infected hosts by measuring the botnet. Enumerating victims in a P2P botnet can protect some local networks, for example, via spam blocking, firewall blocking, and identification and treatment of local victims. In fact, several commercial entities offer listings of the IPs of P2P bots to assist local network administrators.

Researching P2P botnets, like their centralized counterparts, first requires the researcher to understand how the bot modifies the host Operating System. This first, initial impression is achieved by running the bot in a "sandbox" environment, which could be something as simple as a baremetal (i.e., non-virtualized environment) machine running a single guest operating system; then after the initial infection has occurred, analysis can then be performed on determining exactly what files were added, modified, or deleted, in addition to which system calls were made.

Since this initial impression of the P2P bot provides a reasonable understanding of only how it interacts with the host OS, a better understanding of the network behavior by the bot, especially P2P bots, is of much greater concern. Detailed protocol analysis can be ascertained by studying captured network traffic on a network proxy and this protocol analysis provides many clues as to how the bot communicates with its peers and with the bot master, as well.

Once the method for how a bot communicates with its peers is known, it is possible to build a crawler to examine the botnet. A crawler works by initiating some sort of get-peerlists network transactions by either sending look-up requests or get-peerlist protocol employed in the specific peer to peer network that the botnets are built upon. The crawler can then recursively send the same requests to the newly discovered IP addresses of the infected hosts. However, the crawler method in general would not be able to enumerate bots behind firewall or NAT. These bots cannot be accessed from the connection initiated outside unless the firewall or NAT specifically allows such connection. Since the crawler needs to initiate the contact to the nodes, it cannot enumerate the infected hosts behind firewall or behind the NAT configuration that essentially acts as firewall. This limitation to crawling can be significant, as many bot-infected machines can be behind a NAT or firewalls in both homes and offices. Thus, crawler-based enumeration alone would miss a large portion of botnet infections.

Therefore, we sought to find an enumeration method that can identify the infected hosts regardless whether or not they are behind a NAT or firewall devices. Here, we present the Passive P2P Monitor (PPM) and the FireWall Checker (FWC).

Passive P2P Monitor (PPM) is a collection of a "routing only" nodes that act as peer nodes in the P2P network, but are under the control of a single user (i.e., defender). Like a crawler, in order to build the emulated node in PPM, we first need to acquire the understanding of P2P protocol details used by the botnet as a C&C channel or as a bootstrapping mechanism to join the C&C channel. With the P2P protocol knowledge, we can then build the "routing only" nodes on the P2P network.

Since most P2P protocols use regular nodes as overlay "routers" for lookups or as intermediary hops for finding destination nodes, a natural approach to monitor the activities on the P2P network is to join the network as a large number of "routing only" nodes. These nodes can run on a single machine, which can then record the traffic they see and use it to look for bot's location and observe its P2P network interactions.

We further sought to answer a research question: how many bots are behind firewall or NAT. PPM can enumerate both firewalled and unfirewalled nodes, however it cannot distinguish which node is behind firewall or not. To find out which node is behind firewall, we designed and developed the FWC (FireWall Checker), which can be used in conjunction with PPM or any kind of passive monitor that receives the network transaction initiated by the infected host.

FWC utilizes the fact that the modern stateful firewall remembers the connection details of an internally initiated communication such that replies coming from outside (remote) can be allowed back to the initiator. This creates a state table containing the transport protocol, initiator IP address, initiator port number, remote IP address, and remote port number. If the firewall acts as a NAT, it will also remember the port allocated by the NAT for that connection.

When a remote bot-infected machine sends a message to our enumerator (e.g., PPM), we can then send back two query packets to that node: one from the sensor and one from another IP address (i.e., FWC) that we control and monitor. If a reply is sent to both hosts, then the infected host is fully open and not behind any sort of stateful firewall. If a reply is only sent to the enumerator, then the node is behind a firewall/NAT. If no reply is received, then the node may be offline, or the IP could be spoofed in the original message.

As an empirical study, we used the PPM and FWC to enumerate the infected hosts on the Storm botnet, measuring the portions of nodes behind firewall or NAT. Our result shows that almost 40% of bots that contact PPM are behind NAT or firewall, implying that a botnet enumeration based on crawling mechanism alone would miss out significant portion of the botnet population. When we analyzed the difference between the enumerations results from the PPM and a crawler, we also found an interesting fact that the P2P bots that are found by the crawler but not found by the PPM have a fairly short lifetime (e.g., close to 0 minute on average), while the bots found by the PPM but not by the crawler had a lifetime of 19 minutes on average. This implies that the PPM would not be able to enumerate a bot that does not send a search message, while the crawler can quickly enumerate a connected group of localized peers who are connected through constant publicize messages that Storm bots send out. However, such short-lived bot is in fact not a member of the botnet yet, as without a search for a hash it cannot join the C&C channel. Thus, the crawler does not find bots that the PPM is able to.

After we analyzed the difference in enumeration data sets between a crawler and the PPM, we wanted to see if the PPM's enumeration capability indeed covers the entire P2P bot network. To show a complete coverage of the PPM, we first derived a PPM coverage model based on probability theory, and then empirically confirm that with high probability the PPM will enumerate all the bots who sent out search message at least one time to a k number of peer nodes. The k for the Storm bot we experimented with is 200.

2. PRELIMINARIES

2.1 Storm Botnet

The Storm botnet, which originated in January of 2007 [10], is one of a few known P2P-based botnets, and has attracted a great deal of attention for its architecture, variety of transmission methods, and size. Despite its unique architecture, the Storm botnet is capable of engaging in malicious behavior typical of other botnets. Since its discovery, it has been used for distributiing spam emails, participating in "click" frauds, and launching distributed denial of service attacks against a variety of targets, most commonly spam blacklisting services as well as anti-malware researchers [25].

Storm has employed two distinct peer-to-peer networks for use as its Command and Control (C&C) channel. The first network, used from January 2007 until October 2007, co-opted the Overnet. Starting from October 2007, new Storm bots joined an "encrypted" network that follows the same set of protocols as the earlier network but encrypts packets at the application level using the Vigenère cipher with a 320-bit key. Since the packets are encrypted, this network no longer interacts with the original Overnet network.

Overnet Protocol: The Overnet protocol [19] like other widely-used peer-to-peer protocols, such as BitTorrent [16], utilizes a Distributed Hash Table (DHT) interface for finding files. The DHT interface allows a node to publish [key, value] bindings, where *keys* are constrained to be elements of *ID Space* – 128 bits in size, while *values* are arbitrary strings. Lookups 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, including name, length, and the IP addresses of peers that have the file.

Each peer in the Overnet is assigned an ID from the *ID Space*. Peers are then responsible for maintaining the binding values for keys that are "close" to their ID. Each peer maintains a local "routing table" of the IP addresses and IDs of other peers. It is similar to Kademlia [18] where each peer's routing table contains $k = 20$ nodes whose IDs match the first i bits of the peer's ID, for each $i \in \{0, 1, 2, \ldots, \log(n/k)\}$. A peer searches for a given key κ 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 k nodes closest to κ, which are then added to the list of known peers, until it finds a replica root. The distance between two IDs is the 128-bit integer represented by their exclusive or ($\oplus$). In Overnet, like other DHTs, the nodes closest to the key, or "replica roots" can be found with logarithmic efficiency. Nodes can join the network by using a "bootstrapping list" – this can be either a file consisting of hard-coded IP addresses and ports or a link to a website. Important message types for Overnet protocol are introduced in Appendix A.

Storm's Overnet Protocol: Storm bots connected to the Overnet network used the same ID space, message types, and semantics as the other clients, and earlier version of the Storm was connected with Overnet, which has been used for old eDonkey clients. However, after Storm started encrypting traffic using the Vigenère cipher, it has been disconnected from Overnet, forming a separate network. (We describe how we cracked the xor-key in Appendix B.) Storm makes use of its P2P network(s) to publish and search for botnet-related information. Over the course of time, the encoding of this information has undergone several revisions. For example, in one revision, it was stored encrypted with RSA [9], which the searching bots decrypted using a key hard-coded in the binary along with other information received in the search reply [25].

Recent work by Stewart [24] revealed more detailed information about the implementation of the botnet, including a hash generation algorithm [24]. As mentioned above, every node in the Storm network is associated with a 128-bit ID. The hash generation algorithm produces 32 hashes from the current date and 32 hashes from the (current date - 1900 years), which are (relatively) uniformly distributed over the 128-bit ID space. Nodes in the network use those hashes to publish and search for information. For example, a node will publish to one of the 32 hashes a specially-crafted hash containing its IP address and TCP port. Any node can search for that hash and find that it has to connect to that IP address and TCP port to download information (for example, the latest malware update). Due to timezone differences, each Storm node also searches for the previous day and next day hashes.

2.2 Enumerating Storm Bots

The peer-to-peer nature of the Storm botnet requires each node to communicate with other nodes in order to search or publish information, using multi-hop P2P routing protocol. There are different ways in trying to enumerate the number of P2P bots in the Storm network.

2.2.1 Bare-Metal Machines

A simple way to monitor a botnet is to run a bot by infecting a machine (called a Bare-metal) and observe its traffic. Usually, it is fairly easy to infect a real machine to become part of the Storm network. The downside is that a physical machine is needed, which costs both time and money. Other resource constraints include obtaining an IP address and configuring the internal network so as not to take part in the malicious activities, by for example having the external router block all SMTP traffic.

Virtual Machines (VM) emulate a real Operating System on a physical machine. Several VMs can be run on one physical machine. Thus, the number of infected machines can be increased. Moreover, the binary can be analyzed – the state of the virtualized hardware, including the CPU registers and memory, can be inspected. However, some malwares have been reported to detect these VM environments [8]. With either bare-metal machines or virtual machines, it is very hard to enumerate the Storm network: (i) only a few of the Storm nodes are controlled, and (ii) only a partial view of the network is obtained.

2.2.2 Crawler

One of the simplest but most well-known approach to enumerating any network is to crawl that network. The crawler starts from some known starting point (for example controlled bare-metal machines). It then asks the first node for more nodes. This can be done in the Storm network by sending an Overnet routing request. Since Storm is based on the Overnet protocol, nodes in the network need to route requests so that the searched ID (hash) can be found. The crawler will then ask the newly found nodes for more nodes, gradually building its list of known Storm nodes.

A crawler can be easily implemented and is lightweight and could probably crawl the whole network within minutes. However, the major problem with crawling is that nodes behind firewalls or NATs cannot be contacted. Modern stateful firewalls remember the connection details of an internally initiated communication such that replies coming from outside (remote) can be allowed back to the initiator. This creates a state table containing the transport protocol, initiator IP address, initiator port number, remote IP address, and remote port number. If the firewall acts as a NAT, it will also remember the port allocated by the NAT for that connection.

Moreover, churn, dynamic IP addresses, packet loss, and stale nodes returned by Storm bots can lead the crawler to finding only a subset of all the nodes in the network. If a node is contacted by the crawler and does not respond, the crawler cannot assume that this node is part of the network.

2.2.3 Sybil Attack

Since we know the P2P protocol spoken by Storm nodes, any process that speaks the P2P protocol can be joined to the network. They only listen in to the network and reply to routing messages and participate in the P2P network. This node could be implemented in a very light-weight manner, which allows many nodes to be run on a single physical machine, thus acting as a Sybil [5] and passively monitoring the P2P network. Over time, these nodes will become "popular" (so called, Eclipse attack [21]) since they are always online and contribute to the P2P network. Thus, they will attract more messages and more Storm nodes will know of these nodes (even bootstrapping from them). The main downside of this approach is its passive nature. It is also unable to determine if a message received from an IP address is spoofed or if it is a real IP address.

2.3 Related Work

Understanding the behavior of botnets themselves, in both centralized and decentralized forms, is critical to botnet node enumeration efforts. Rajab et al. presented in [20] a study of nearly 200 unique IRC botnets in an effort to methodically dissect botnet behavior. Assessed is the prevalence of IRC botnet activity, subspecies variety and quantity, and changes in a botnet over time. In this attempt, they were quite successful. The taxonomy of botnet structures offered by Dagon et al. in [4] further characterizes botnet behavior and architectures, with the intent of classifying botnets in dimensions which can correspond to mitigation strategies. While our research is concerned exclusively with a peer-to-peer botnet and incorporates different analysis techniques, e.g., in enumeration, our efforts are complementary for overall botnet study.

Grizzard et al. outlined the history of bots, both malicious and non-malicious, and the emergence of peer-to-peer based botnets in [10]. The key challenge in detecting the bot controller in a peer-to-peer network is due to the dynamic and distributed design of the architecture. A case study of a particular peer-to-peer bot, Peacomm, another name for the Storm bot, is also included to provide the reader with an understanding of how peer-to-peer botnet strategies may be implemented in the real world.

The potential threat posed by bots using peer-to-peer protocols for their command and control (C&C) was discussed in [3]. Cooke et al. identified some of the foundational analysis techniques for handling botnets including incapacitation of the botnet itself, monitoring the C&C channels, and tracking the propagation and attack mechanisms. This work highlights the underlying difficulties in monitoring the channel(s) that may lead back to the bot controller [3].

Critical to enumeration and mitigation research in peer-to-peer botnets is binary analysis, including the system and network activity of malware. Stewart provided the first in-depth analysis of a Storm binary in [25], which includes detailed information about its hard-coded peer lists and the hash generation algorithm Storm employs. This research also provides information as to what the bot, at the time of writing, was used for: distributed denial of service attacks and email spamming. Stover et al., Florino, and Cibotario also provided comprehensive binary analyses [26, 9] of the Storm malware. Stover et al. also explored the Nugache botnet which also employs a peer-to-peer architecture.

Historically, the Storm malware authors have continuously changed the operations of the botnet in attempts to thwart intrusion and manipulation by outside entities. Following several evolutionary changes to Storm's architecture and behavior, Stewart presented a briefing on the Storm protocols and intricacies of its encryption usage [24]. The architectural hierarchy of the botnet within Storm's Overnet-based network was discussed, which gave insight into the botnet's use of tiers and Nginx proxies to create five different levels of functionality. Storm nodes lowest in the hierarchy, subnodes, are relegated to spamming and DDoS attack tasks. Supernodes, next in the hierarchy, are used as reverse HTTP proxies and DNS fast-flux name servers. A tier of Nginx servers known as subcontrollers are employed to obscure a single master Nginx proxy, which is in turn used to conceal the node on the top-tier of the hierarchy, a master Apache C&C server. This insightful presentation principally covered the current architecture of Storm rather than the complete enumeration of its nodes, which is our focus in our research.

Efforts to detect Storm traffic in networks have also been performed. BotHunter [12] is a host-based IDS Storm detection framework. BotHunter correlates conversation dialogues between internal nodes and external IPs with consistent known Storm behavior in an effort to detect local assets infected with Storm. BotHunter is effective for local system administrators and users in finding these bots within their monitored infrastructure. BotMiner, by Gu et al. [11] also is designed to identify botnet activity in IRC, HTTP-based, and peer-to-peer botnets in a locally monitored network. BotSniffer [13], by Gu et al., is also a botnet traffic detection framework, but only for HTTP-based and IRC botnets. These detection frameworks are designed to locally identify botnet activity, rather than enumerate external nodes, as in our work.

Beyond local network and system behavioral analysis of Storm malware and localized IDS-based tool design, Storm node enumeration, which this paper elucidates, is an additional avenue of research pursued by several individuals. Enright presented an exploration of the Storm botnet using a Storm network crawler in [6]. This work was later expanded, where Kanich et. al discussed the Storm botnet in an exploration of accuracy assessments in botnet measurement with their implementation of a Storm network crawler known as Stormdrain [15, 7]. Particular focus is placed on parsing nebulous traffic from a variety of sources from actual, legitimate Storm traffic, based on the ability of encountered nodes to correctly speak the Storm botnet protocol.

An enumeration attempt was also performed by Holz et al. in [14], where a delineation of the network and system behavior of Storm binaries was offered, as well as tracking methodologies for measurement, and mitigation strategies for these types of peer-to-peer botnets. The authors' work involved estimating the number of compromised nodes within Storm with a crawler that repeatedly performed route-requests as well as delineating two methods to disrupt Storm command and control functions. These disruption methods employed the use of the Sybil attack as well as a pollution attack to mitigate the botnet. The ability of our PPM and FWC tools to enumerate and differentiate nodes that are behind a firewall or a NAT-network allows us to enumerate the network more completely.

3. ARCHITECTURE

Crawling has been a major tool for enumerating nodes in a P2P network [23, 27] and the Storm botnet [15, 14]. However, it has a fundamental limitation: Nodes behind a firewall or a NAT cannot be reached. To resolve this issue, we focus on designing a monitoring system, called Passive P2P Monitor (PPM), similar to a Sybil [5] attacker. A node in PPM speaks Storm's Overnet protocol and participate in the network routing protocol. However, it does not send any malicious traffic – it only listens in to the Storm network and acts as if it is a legitimate bot by routing messages.

As mentioned in Section 2.1, the Storm botnet uses a modified Overnet protocol, which is a variation of Kademlia protocol [18]. Since open-source implementation of Overnet protocol is not available, we modified the aMule [2] P2P client to implement a PPM node, which communicates with Storm Botnet. GUI and file-sharing features of aMule were disabled, which allows 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.

Expected Problems and Fixes: PPM is passive by nature as opposed to a crawler, which will actively seek new nodes. The problem with the passive nature of PPM is that it cannot identify a new node, if Storm botnet does not send messages to it. It will just reply to requests from other nodes and regularly send self-find-requests to maintain its routing table. The PPM may have only a small view of the whole network – it will know only of those nodes in its routing table and those nodes that contacted it. To remedy this problem, many PPM nodes can be run in parallel (therefore, becoming a Sybil node), each bootstrapping off of a different node in the Storm network. This ensures a more global view of the botnet. Moreover, PPM is run for a long time so that it will be more widely known (similar to Eclipse attack [21]) – (i) stay in other nodes' routing tables longer, and (ii) in responding to routing requests, nodes are more likely to return contacts that have been long-lived, hence more likely to return PPM as one of the contacts. Thus PPM acts as an extension to the Sybil attack. Enumeration of the size of the Storm network could be feasible and PPM could detect nodes behind a firewall/NAT box, potentially providing a better estimate than a crawler, as we will see in Section 4.

Another problem with PPM is its lack of source address spoofing detection. It does not check if a request received is from a real IP address or from a spoofed IP address. Spoofing can be used to "poison" the network or by other researchers monitoring the Storm network. PPM is hence modified to include a *handshake* mechanism. After receiving a request from a node in the network, the PPM will reply to that node as before. In addition to that reply message, the PPM will send a request to that node, expecting a response. If the node responds, then the IP address is real, not spoofed. However, if the node does not respond, it does not necessarily mean the IP address is spoofed: (i) The node could have gone offline or changed its IP address due to DHCP, or (ii) the request packet from the PPM was lost in transmission, (iii) the IP address is in fact spoofed, or (iv) the bot (or the node who sends messages to the PPM) has a special protocol, which prevents it from replying to arbitrary messages. For example, a crawler does not have to reply to either the PPM or FWC.

Distinguishing Firewalled Nodes: The final modification to the PPM design is to add a firewall checker (FWC), which is used to determine if a Storm node is firewalled. As mentioned in Section 2.2, firewalls and NATs will allow packets through only if the machine behind the firewall initiated (sent a message) the connection to the machine outside of the firewall. If a node is firewalled, it will reply to the PPM request since it originally sent a request to PPM (firewalls do not speak specific application-level protocols and will let any packets through) but not to the FWC request. Again, this is not fool-proof for the same reasons as the PPM, that is, if a node did not reply to FWC, it does not necessarily mean that the node is firewalled. Since both the PPM and FWC are run for an extended time, if a node never replies to FWC, then it is highly likely that the node belongs to case (i), (iii) or (iv) described above. Note that for all these cases, it will not reply to crawler as well. In other words, FWC tries to emulate the behavior of the crawler, which sends messages to arbitrary node. The PPM and the FWC have to be run on separate IP addresses but on the same LAN so that the PPM request and the FWC request will be sent at almost the same time. This prevents the Storm node from receiving the PPM request, replying to it, going offline, and then receiving the FWC request, which does not get replied since the Storm node is now offline.

Implementation Details: Both PPM and FWC each keep track of all request packets sent. After not receiving a response for 15 seconds, that packet will be deleted and marked as timed-out. The number of responses received and requests timed-out for each IP address is collected. Since FWC will be sending requests to nodes in the Storm network, unfirewalled nodes will learn of FWC and might pass this information along to other nodes. Thus, FWC might also become popular. This needs to be avoided because if FWC becomes popular, then it will receive messages from firewalled nodes (from our observation of the Storm network, a node sends a request to every node in its routing table every 10 seconds). A firewalled node might respond to a FWC request if the node recently sent FWC a message – the firewall will let the FWC request through). This leads to an undercount of firewalled nodes. FWC is hence modified to drop all request messages not coming from PPM IP addresses. Also, PPM messages to FWC are in a special format, which is unlikely to be replicated by outside nodes – the message type and format is not the same as the Storm network.

Figure 1 shows the components of the final PPM architecture:

1. A Storm node in the bot network sends a request to one of our PPM

2. PPM 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

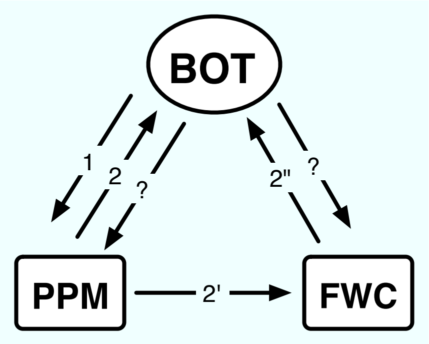


Figure 1: Final Design of the PPM

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.

4. ANALYTICAL AND EXPERIMENTAL RESULTS

4.1 Experimental Settings

Using 5 computers not behind a firewall, we deployed infected nodes, PPM nodes, the FWC, and a P2P network crawler. On two machines, we deployed 256 PPM nodes, which lies on the same LAN as a third machine running the FWC. The results of this paper are based on data collected from 20 days (Aug 25 to Sep 18 2008) of the PPMs and FWC, combined with 10 days (Sep 8 to Sep 18 2008) crawler, which ran on the fourth machine. We also deployed 16 virtual machines (referred to as VMBOTs) on the fifth computer and infected each of them with Storm. As mentioned in Section 2.2, running a bot on a Virtual Machine is similar to running it on a bare-metal machine with the added benefit that multiple bots can be run on one physical machine instead of multiple physical machines. We made sure that the behavior of the bot running on a VM and running on a bare-metal machine are similar, so as to make sure that the bot did not detect the VM and started behaving abnormally. The reason to run a bot within a VM is to obtain some ground truth about the normal behavior of the botnet if its behavior changed due to an update. We want to stress that both PPM, FWC, and crawler speak only UDP and do not participate in any illegal activities of the Storm botnet. Moreover, we set up our VM bot such that all spam and other illicit traffic was blocked.

4.2 PPM vs Crawler

We compare the coverage of the crawler and the PPM and analyze how well the crawler can accurately enumerate all the Storm nodes in the network. First, we try to determine how many Storm nodes are firewalled, since as explained in Section 3, crawling cannot find firewalled nodes. Figure 2 shows the CDF of the fraction of responses of every node. The fraction of responses for each IP address is calculated as

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

PPM: On average, each node in the network will respond to more than 60% of the PPM requests. The time-outs can be attributed to

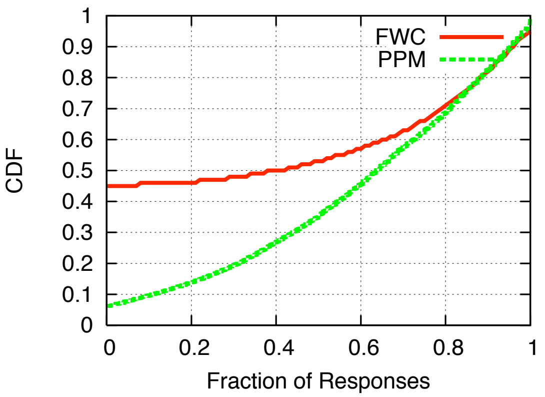


Figure 2: CDF of fraction of responses per IP address for both PPM and FWC

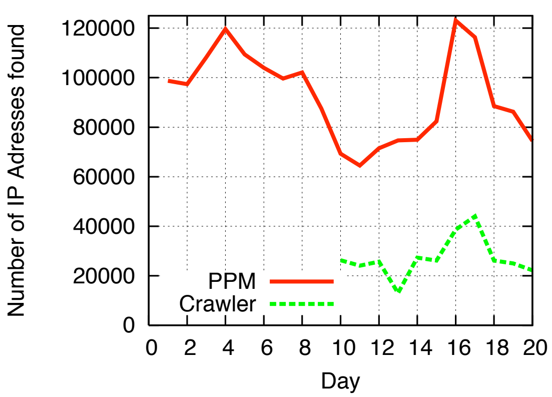


Figure 3: Number of IP addresses found by crawler and PPM per day

packet loss and churn. A response to PPM means that the node is part of the Storm network. We do not distinguish if some of the IP addresses fall within the same subnet, for example, controlled by an ISP or BGP router. About 6% of the IP addresses never respond to any requests sent from the PPM. We believe the number of requests that were not responded to is due to a combination of the following factors:

- Spoofed IP addresses: There is no ground truth as to whether an IP address is spoofed, but some of those nodes could be spoofed – either by researchers or rival botnet operators.
- Packet loss: All messages are over UDP, which uses no packet retransmission or packet acknowledgment.
- Overloaded bot: Each of our PPM nodes receives about 100 messages per second (not counting the extra messages generated due to the handshake mechanism). A bot will receive the same order of magnitude of messages and if it is running on an older machine, it could be overloaded with messages and not be able to respond to all requests.
- Application state: Each bot could be keeping track of some state such as IP addresses a message has been sent to in the past time period. Since half of the PPM nodes uses the same IP address, a response might be sent to one PPM node but not the other.
- Other researchers: For example, another researcher's crawler will most likely not respond to PPM (or FWC) requests. We believe that there are a fairly large number of crawlers on the Storm network.

We do not try to differentiate if a failure to reply to a packet means that the IP address is spoofed or for any of the reasons mentioned above, since one of the goals of PPM is to find the nodes in the network.

FWC: About 46% of the nodes never respond to FWC – this does not mean that 46% of the nodes are firewalled. Nodes' failure to reply to packets can be due to any of the reasons, or a combination of reasons, mentioned above. This result implies that more than 40% of the nodes in the Storm cannot be discovered by a crawler.

Coverage: We next compare the coverage of PPM and the crawler for each day. Coverage indicates the number of Storm nodes found. Figure 3 shows the number of IP addresses discovered by both the crawler and the PPM for each day that PPM has been running. The lack of data for the crawler for the first 10 days is due to starting PPM before the crawler. As can be seen from the figure, even if the crawler was started earlier, it would not have improved the crawler's coverage of the Storm network since it contacts a constant amount of Storm nodes every day. Note that the figure indicates the total number of IP addresses found per day. It does not differentiate if the same IP addresses were also found the previous day.

PPM's coverage is consistently higher than the crawler's. This is due to a number of reasons – the major one being that the crawler is not able to find and enumerate most of the Storm nodes due to firewalled bots.

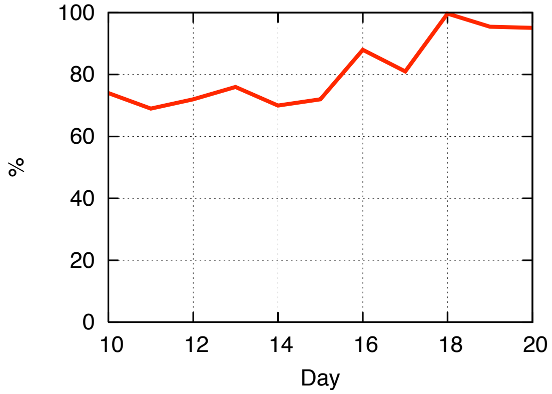


Figure 4: Percentage of IP addresses found by crawler that were also found by PPM per day

Although PPM can find more nodes than the crawler, it could be the case that they are finding different nodes. Figure 4 shows the percentage of IP addresses that the crawler found which the PPM also found, for each day of running both the crawler and the PPM. This shows that most of the nodes found by the crawler were also found by the PPM. However a lot of nodes (up to 30% on some days) are seen by the crawler but never sent a message to our PPM nodes. The percentage of nodes found by the crawler that were also found by PPM over the whole time period (8 days) is 87%. Since this would imply that PPM did not receive a message from a large portion (13%) of the network, we looked at the crawler logs to determine when each of those nodes were found and how long they responded to the crawler's requests. We found that those IP addresses that crawler found but PPM did not find (call those nodes C) had a *lifetime* of 19 minutes. Those IP addresses that were found by both the crawler and the PPM (call those nodes I) had a *lifetime* of 100 minutes in the crawler logs. A lifetime of 100 minutes means that the node keeps responding to the crawler for 100 minutes, then stopped responding to the crawler. This can be due to (i) the Storm worm being removed from that node, or (ii) the node went offline.

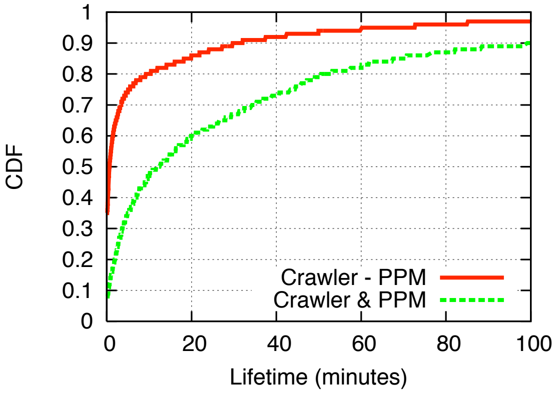


Figure 5: CDF of the lifetimes of nodes that (i) were found by the crawler but not by PPM and (ii) found by both the crawler and the PPM

Figure 5 shows the CDF of the lifetimes of the nodes for both cases C (Crawler $-$ PPM) and I (Crawler & PPM). Note that the graph does not go all the way to a CDF of 1 because there are a few nodes that are long-lived and the graph will be unreadable. 80% of C nodes have a maximum lifetime of 10 minutes, while 80% of the I nodes' lifetime are 50 minutes. The nodes which have a short lifetime are not found by PPM. This is expected as the crawler actively seeks new nodes, whereas the PPM passively wait to receive messages from other nodes, and thus wait to become popular and be in many nodes' routing tables. Although the PPM misses those nodes, they do not come back online afterwards. Thus the crawler is not finding nodes that the PPM can find.

4.3 Coverage of PPM

The previous section showed that crawling is not a good method for enumerating all the P2P bots in the Storm network. This section examines in more details the coverage of the Storm network obtained by PPM. We also analyze how good the enumeration provided by PPM is and how likely is it true.

Modelling Coverage of PPM: First of all, we determine the likelihood that at least one of our PPM nodes will be sent a message by a Storm bot. Our analysis is based on the *bins and balls* problem. We will describe the problem and then see how it relates to PPM receiving a message.

In the *bins and balls* problem, let's first assume that you have n bins and 1 ball and you have to 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 $1 - (1 - \frac{1}{n})^k$.

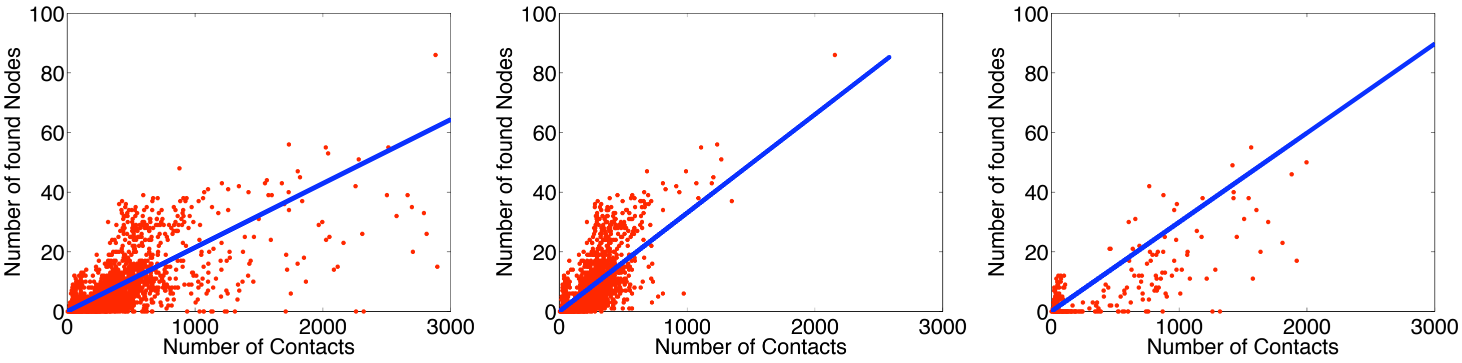
The same model can be used to determine the likelihood of PPM receiving at least 1 message from a bot. However, simply changing the variables do not work because in Kademlia-style P2P networks, all the nodes do not have an equal probability of receiving a message. Some nodes have a higher probability of receiving a message than others. For example, since our PPM is online for a long period of time, it will be on many nodes' routing table and will be popular and the probability of PPM receiving a message is higher than a Storm bot that only comes online for 15 minutes per day – enough time for the user to check his/her email. The model used can be thought of as a *biased bins and balls* problem. Going back to the basic *bins and balls* problem, some bins are "larger" and have a higher probability of receiving a ball than other bins. As we will see later, different nodes in the Storm network have a different probability of receiving a message (see Figures 6 and 7). The probability of PPM 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 with that hash to (see Section 2.1 for the Storm hash generation algorithm). Next, we experimentally determine the value of p and show what L is for varying values of k.

Probability of PPM receiving a random message: To obtain the probability p of PPM receiving a message from a Storm bot, we analyzed the VMBOT data. We looked at only the following 3 message request types (an outline of the other Overnet message types is given in Appendix A) –

- *Search* is a message used in routing to find the replica roots
- *GetSearchResult* is a message sent to possible replica roots to get the actual result (binding information)
- *Publish* is a message meant to publish binding information

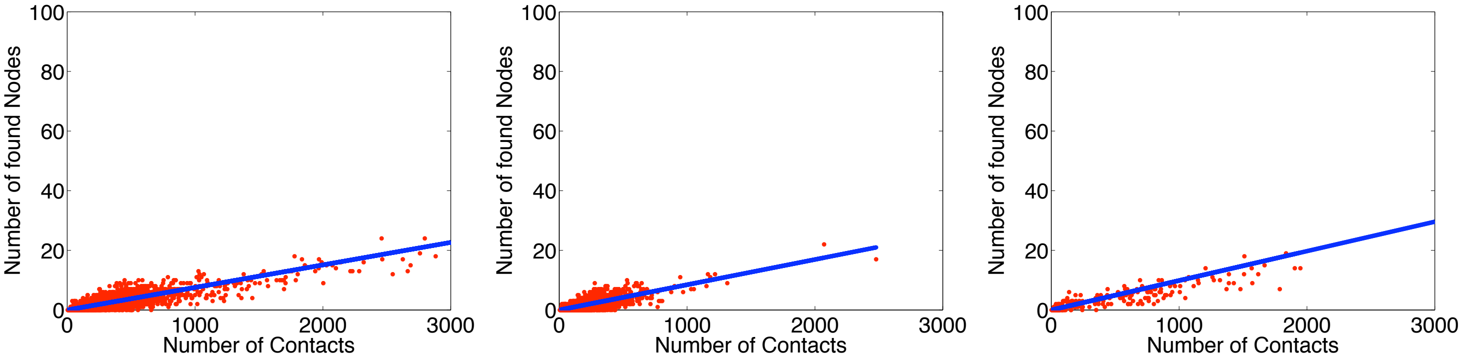
The other two request message types are *Connect* which is to bootstrap to the Storm network, and *Publicize* which is a "keep-alive" message sent every 10 seconds. The *Connect* messages are used only for new nodes to bootstrap to the network and are not considered since they apply only to new nodes or old nodes that want to rejoin the network, and might include a bias for nodes that constantly churn out and in the network. The *Publicize* requests are not considered since those are sent only to nodes in the routing table. This will include a bias towards nodes that are long-lived (such as PPM). The three message types *Search*, *GetSearchResult*, and *Publish* are sent more uniformly as shown in Figure 8. Each node is equally likely, regardless of nodeID, to receive a *Search*, *GetSearchResult*, and *Publish* request. We looked at 20 target IDs for *GetSearchResult* and *Publish*, represented by the 20 blue points on the graph. The red points indicate the nodes that are contacted for that target ID. 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. As expected, more nodes are contacted closer to the target ID since those nodes are potentially the "replica roots".

Figure 6 shows the number of nodes that each of the 16 VMBOTs 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 6 (a) shows the number of contacted nodes for **Search** requests from each of the VMBOTs. Point [1000, 22] on the graph means that a *Search* request with hash H was sent to 1000 nodes and 22 of those 1000 nodes were our PPM nodes. Each point represents a request for a different hash. The line of best fit is also shown and is drawn using the *polyfit* algorithm from matlab [17]. The algorithm works by solving the least square problem to draw the line of best fit. For *Search* requests, the probability p (slope of the line of best fit) of PPM receiving a message from a Storm bot is 2.3%; for *GetSearchResult* requests, the probability is 3.3%; and for *Publish* requests, the probability is 3%. The first line of Table 1 shows the result for PPM.



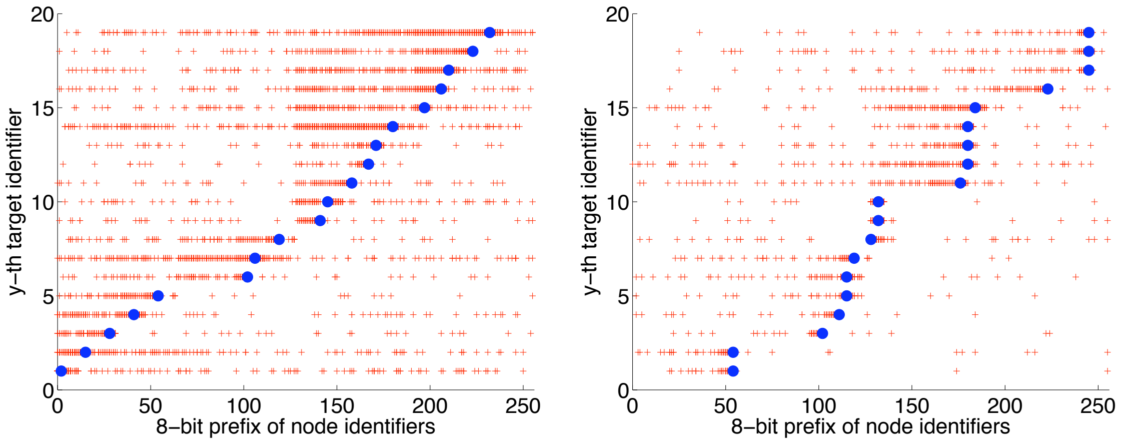
(a) (b) (c)

Figure 6: Number of contacts a bot sends a message to and the number of those contacts which are our PPM nodes for (a) *Search*, (b) *GetSearchResult*, and (c) *Publish* message types



(a) (b) (c)

Figure 7: 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 8: Distribution of Storm node IDs for search and publish

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

Table 1: Probability of receiving a message for each of the 3 message types for PPM and a random Storm node

We also varied the number of PPM nodes to determine the effect on the coverage. Figure 9 shows the result for the three message types. It shows that increasing the number of PPM nodes results in a linear increase in the probability of PPM seeing a message from a bot. From the graph, we can obtain the probability p of PPM receiving a message with some hash from a bot 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%.

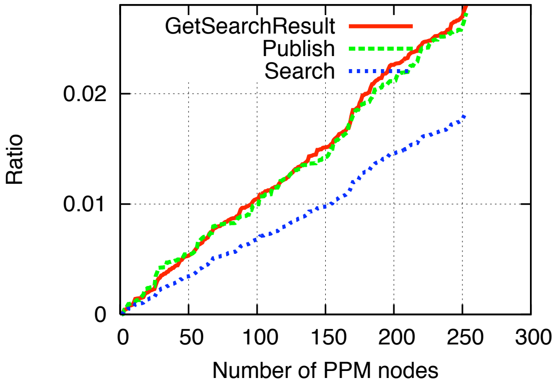


Figure 9: Probability of PPM receiving a message from a Storm node for varying number of PPM nodes

Figure 10 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 daily hash to. Thus, our PPM will receive a message from one bot with very high likelihood (87%) if k is greater than 100 contacted nodes. If $k = 200$, the probability goes up to 98%. Looking at both Figures 6 and 7, each bot sends the same hash (for either of the three message types) to at least 200 nodes, suggesting that PPM can find most of the nodes in the Storm network. We also want to emphasize that running 256 PPM nodes is not hard – each PPM nodes uses a few MB of memory and very little bandwidth and CPU usage.

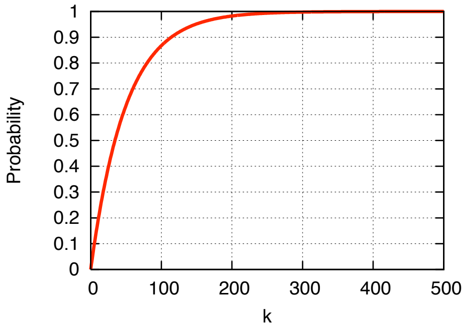


Figure 10: Plot of the probability of 256 PPM nodes receiving a message from all the Storm bots for varying k, where k is the number of nodes contacted by a Storm bot

Probability of a set of bots receiving a random message: We next compare the probability of PPM receiving a message from a bot with the probability that a bot will receive a message from another bot for each hash. Since we do not control 256 real bots, we randomly picked 256 "long-lived" IP addresses in the VMBOT data set, that is, IP addresses that appear every day. Figure 7 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 6 for each message type. The probability of 256 PPM nodes seeing a message from a Storm bot is higher than the probability of 256 bots seeing a message from the same Storm bot. This does **not** indicate that deploying 256 PPM nodes provides a better coverage of the network than deploying 256 bare-metal (or VM) bots. This is because all the PPM nodes have been online for weeks, which is longer than the average Storm bot. PPM is thus more likely to be in other nodes' routing table and more likely to be returned in responses when nodes try to learn of other nodes in the network. The second line from Table 1 shows the probability for each message type for the probability p of other bots receiving a message from a Storm bot.

4.4 Dynamic IP Block Aliasing

In the previous section, we have shown that we can enumerate the entire Storm network with high probability. However, care has to be taken not to overcount the number of Storm nodes in the network. Some of the different IP addresses could actually be related and represent the same Storm bot, due to dynamic IP block aliasing and DHCP changes. Thus, counting the total number of IP addresses as the total number of bots in Storm is a gross overestimate.

SORBS [22] (Spam and Open-Relay Blocking System) is a system which contains a blacklist of spamming IP addresses. It also contains a list of dynamic IP addresses. The IP addresses inside the SORBS dynamic user and host list are usually manually added and could be missing a lot of dynamic IP addresses. Moreover, some of the IP addresses could not be dynamic anymore. We believe that although the SORBS data is probably an underestimate of the total number of dynamic IP addresses, it still reliably can tell us whether an IP address is dynamic or not.

Figure 11 shows the total number of IP addresses found by PPM and the number of IP addresses from that total which are dynamic according to SORBS. About 20% of the IP addresses are dynamic. However, it is very hard to tell which IP addresses are from the same bot. For example, two dynamic IP addresses in a
16 subnet might be the same node, but could be from two different smaller ISPs which are customers of the same bigger ISP.

Figure12 shows the number of IP addresses found by PPM daily. The number of dynamic IP addresses is also shown. $25 - 30\%$ of the total number of IP addresses found per day are dynamic. This shows that just counting the number of IP addresses is an overestimate in trying to enumerate any network and the dynamic IP addresses consist of a good percentage of the total number and should be considered.

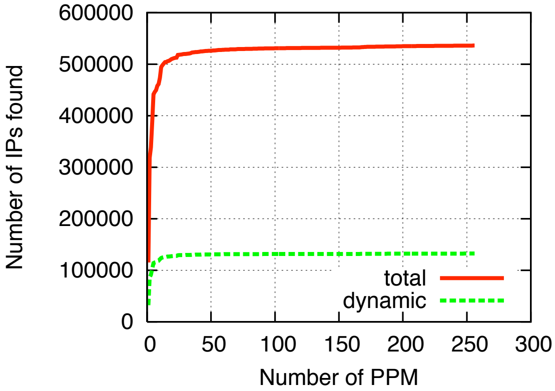


Figure 11: Number of IP addresses found by different number of PPM nodes for 7 days

Although SORBS gives only a rough estimate of the total number of dynamic IP addresses, we explored other methods of determining whether an IP address is dynamic. UDMAP [28] is such a method. It looks at application-level server logs and patterns of IP addresses. The IP addresses are then classified as either dynamic or

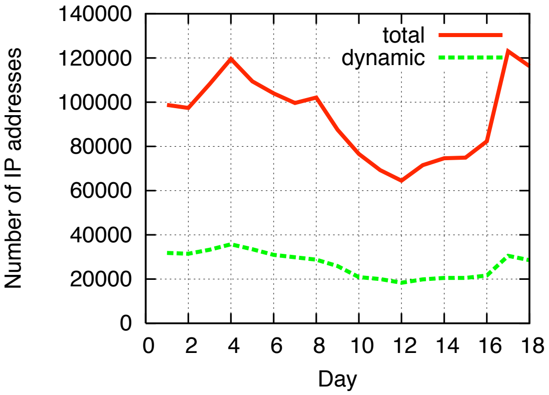


Figure 12: Number of IP addresses found daily by PPM and the number which are dynamic according to SORBS

not and can change over time. It found over 100 million dynamic IP addresses in the world. However, UDMAP is not publicly available at the moment.

4.5 Recent Development

More recently, PPM was left running for more than 90 days (Aug 24 to Nov 30 2008). The number of IP addresses found by PPM is shown in Figure 13. The sharp drop at Day 21 (September 20, 2008) is related to the shutdown of the Intercage ISP [1] which was believed to host the subcontrollers of the Storm network. Since that time, the number of IP addresses found by PPM has been continuously decreasing. We believe that Storm is not dead yet but can easily bounce back to its better days.

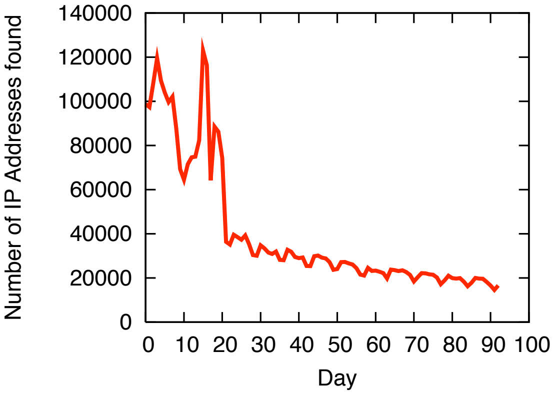


Figure 13: The number of IP addresses found by PPM per day

5. CONCLUSION

In this paper, we presented PPM as an enumeration method for the Storm peer-to-peer botnet and analyzed the differences in enumeration results from the PPM and a crawler. As expected, the PPM instances were able to enumerate significantly more nodes than the crawler as it cannot enumerate nodes behind firewalls. Further, some of the nodes which the PPM could not find (but the crawler does) have a shorter life-time. We also used the biased bins and balls problem to model and analyze the PPM coverage. The result indicates that when a bot sends a sufficient number (approximately 200) of P2P messages, the PPM can detect it with high probability. We also verified that most of the nodes detected by the PPM are either researchers or actual bots.

6. ACKNOWLEDGEMENT

We thank Rick Wesson at Support Intelligence for providing network testbeds for our experiments. This research is supported in part by the NSF under grant CNS-0716025, DUE-0723808, DUE-0830624, DUE-0516085, iDefense (VeriSign), KISA, ROSAEC, and GAANN Fellowship.

7. REFERENCES

[1] Atrivo/intercage's disconnection briefly disrupts spam levels. http://blogs.zdnet.com/security/?p=2006.

[2] aMule network. http://www.amule.org.

[3] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pages 39–44, July 2006.

[4] D. Dagon, G. Gu, C. Lee, and W. Lee. A taxonomy of botnet structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference*. ACSAC, December 2007.

[5] J. R. Douceur. The sybil attack. In *Proc. of the International workshop on Peer-To-Peer Systems (IPTPS) 02*, March 2002.

[6] B. Enright. Exposing storm. In *ToorCon*, 2007.

[7] B. Enright, G. Voelker, S. Savage, C. Kanich, and K. Levchenko. Storm: When researchers collide. *;Login, Usenix*, 33(4), August 2008.

[8] Attacks on virtual machine emulators, http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf.

[9] E. Florino and M. Cibotariu. Peerbot: Catch me if you can. In *Symantec Security Response: Ireland, Virus Bulletin*, March 2007.

[10] J. Grizzard, V.Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Usenix First Workshop on Hot Topics in Understanding Botnets*, April 2007.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic from protocol and command and control channels in network traffic. In *Proceedings of the 17th annual USENIX Security Symposium*. USENIX Association, July 2008.

[12] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of The 16th USENIX Security Symposium*. USENIX Association, August 2007.

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

[14] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the First USENIX Workshop on Large Scale Exploits and Emergent Threats*. USENIX Association, April 2008.

[15] C. Kanich, K. Levchenko, B. Enright, G. Voelker, and S. Savage. The Heisenbot uncertainty problem: Challenges in separating bots from chaff. In *Proceedings of the First USENIX Workshop on Large Scale Exploits and Emergent Threats*. USENIX Association, April 2008.

[16] Mainline. http://www.bittorrent.com.

[17] matlab. http://www.mathworks.com/.

[18] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *1st International Workshop on Peer-to-Peer Systems*, pages 53–62, 2002.

[19] The Overnet Protocol, https://opensvn.csie.org/mlnet/trunk/docs/overnet.txt.

[20] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the USENIX Internet Measurement Conference*. USENIX Association, October 2006.

[21] A. Singh, T.-W. J. Ngan, , P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *IEEE International Conference on Computer Communications (Infocom)*, 2006.

[22] SORBS. http://www.us.sorbs.net/faq/dul.shtml.

[23] M. Steiner, T. En-Najjary, and E. W. Biersack. A global view of kad. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, New York, NY, USA, 2007. ACM.

[24] J. Stewart. 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.

[25] J. Stewart. Storm worm ddos attack. http://www.secureworks.com/research/threats/view.html?threat=storm-worm, February 2007.

[26] S. Stover, D. Dittrich, J. Hernandez, and S. Deitrich. Analysis of the storm and nugache trojans - P2P is here. *;Login, Usenix*, 32(6), December 2007.

[27] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed DHT. In *IEEE International Conference on Computer Communications (Infocom) 06*, 2006.

[28] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are ip addresses? In *Special Interest Group on Data Communications (SIGCOMM)*, 2007.

APPENDIX

A. OVERNET MESSAGE TYPES

The Overnet message types are outlined below

- *Connect*: To bootstrap/join the network.
- *ConnectReply*: 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*: Hello message to say that you are still alive.
- *PublicizeAck*: Reply to the Hello message.
- *Search*: 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*: The reply to a Search message – includes ID, IP, and port of other peers.
- *GetSearchResult*: The node closest to the target ID is found, thus get the results of that search from that ID.
- *SearchResult*: Reply to the SearchInfo.

- *SearchEnd*: Nothing has been found.
- *Publish*: Publish a [ID, IP] binding or for example in a file-sharing application, publish a metadata saying that you have a particular file.
- *PublishAck*: Response to the Publish message.

B. CRACKING XOR KEY

To crack the XOR encryption, we set forth the following initial hypotheses:

1. There exists a single 40 byte XOR key k used for encryption on an overlay network
2. The key k XORs the entire UDP payload u delivered onto the overlay network
3. The key k does not modify u such that $u \,! = ((u \oplus k) \oplus k)$.

All three of these properties were verified during our decryption efforts. First, we collected known plaintext fields, such as eDonkey protocol identifiers (e.g., 0xe3), eDonkey Message Types (e.g., 0x11), IP addresses, and ports. Comparing these known plaintexts and corresponding ciphertexts, we could easily find 20 bytes of the total k keyspace. The rest of the key bytes were determined by solving linear equations obtained from related ciphertexts (e.g., “Search” and “Search Reply”). Finally, using this encryption key, we have been able to run our PPM and Crawler on the encrypted Storm overlay network.