Measuring distributed malware.

A number of challenges arise in using crawling to measure the size, topology, and dynamism of distributed botnets. These challenges include traffic due to unrelated applications, address aliasing, and other active participants on the network. Based upon experience of developing a crawler for the Storm botnet, one can describe each of the issues encountered in practice. The Storm worm is a botnet which appeared in the early months of 2007. Its prolific growth, the use of decentralized command and control communications based on the overnet P2P protocol and fast-flux servers for secondary-stage binary distribution, as well as the capability to aggressively defend itself, make Storm a notable species in the malware ecosystem. Despite considerable interest, Storm's defensive capabilities and its distributed nature have complicated the accurate estimation of its size and understanding of its network behavior. Unlike traditional DHTs, the distribution of peer IDs is not uniform. Underlying virtually all measurement endeavors is the premise that the signal being measured can be separated from any noise produced by the environment or the measurement system itself. If this assumption is not true, or if care is not taken to account for noise, then the resulting measurements may be badly skewed or meaningless. This issue has long been appreciated in the Internet measurement community, since even quiescent networks receive continual streams of bizarre traffic colloquially referred to as Internet background radiation [1]. However, it is less widely appreciated that variants of this problem impact our ability to meaningfully study online distributed malware such as botnets. In particular, the practice of crawling as a technique for measuring the size, topology and dynamism of distributed botnets is fraught with the potential for error. At a minimum researchers must contend with temporal dynamics (e.g., address space reuse), unrelated applications (i.e., many bots piggyback on existing communications protocols), and address aliasing (e.g., via NAT). Potentially worse, however, is the impact of other active participants. Unlike traditional malware whose environment is largely self-contained within a single machine, distributed malware is effectively a single artifact — providing services to the entire Internet. Thus, no single group can safely presume to be measuring a botnet in isolation. In fact the activities of other researchers, miscreant competitors or well-meaning vigilantes will automatically be part of any raw measurement. One may try to highlight these problems. Experimenters developed a crawler for the Storm botnet. Storm uses an existing well-documented peer-to-peer protocol to implement a directory service and thus is straightforward to crawl. Unfortunately, the intuitive way to implement such a crawler — recursively querying peers for their neighbors until a transitive closure is complete — can

produce a size estimate many times larger than the true size of the network. Indeed, depending on how one measures — over different time scales, using IP addresses or peer identifiers, using inbound requests or active probes — different characterizations can be reached. This may explain in part the considerable diversity of claims in the security press concerning the purported "true size" of the Storm network. Even discounting the effects of flooders, naive approaches for counting the number of active Storm nodes result in estimates an order of magnitude larger than the actual number. Over the course of a typical day, simply counting unique overnet IDs would estimate over 900,000 bots and counting unique IP addresses would estimate over 300,000, whereas the actual number is likely less than 40,000. The presence of other active participants in the system in the certain moments of time may lead to significant Storm-related perturbations of the network traffic. This problem is likely to be fundamental and will continue to present challenges for measuring distributed malware in the future. The combination of its effectiveness and unique architecture has focused increasing attention on the Storm botnet. Initial work has detailed the operation of the Storm malware as it transforms a host into a bot, as well as the basic bot communication patterns [2, 3]. Recent work searching the identifier space of the DHT underlying Storm reveals significant non-uniformities from poisoning, and suggests one heuristic for pruning poisoned peers [4]. One may complement this work with a more comprehensive pruning strategy that detects and distinguishes among peers that are unresponsive, advertise incorrect addresses, or attempt to poison the network. The size of a botnet is the most popular metric measured and reported, yet perhaps the one with the most variance. This issue has been explored in detail in [5], highlighting the challenges of measuring botnet size and explaining why size estimates can vary substantially. The goal is to identify and distinguish active bots in the system in the face of unrelated application traffic, address space reuse, address aliasing, and other active participants such as poisoners, polluters, and researchers. In that sense, one may extend the set of techniques in [5] and [4] for refining size estimates specifically for the Storm botnet, particularly in the face of poisoning and NAT. In doing so, though, [10] not only obtains a more accurate estimate of botnet size, but more importantly removes noise from other analyses such as the dynamics, operation, actions, and behavior of Storm. Address space reuse via DHCP and address aliasing via NATs [6-8] add uncertainty to bot disambiguation when using IP addresses [5]. Application identifiers can disambiguate in both instances [7], but NATs present yet another problem. Ensuring that a peer is live and functioning (beyond just appearing in a routing table) requires active communication with the peer. However, a crawler cannot directly contact peers behind NATs. Instead, analogous to [8], [10] relies upon

communications initiated by peers behind NATs to observe and disambiguate them. Finally, since poisoning attacks are being waged against Storm, whether effective or not as an attack against Storm [9] any measurement efforts must take poisoning into account [4]. [10] describes effective heuristics for differentiating between valid and masquerading Storm nodes and a strong predicate based upon reverse engineering the Storm OID generator. The Storm botnet is organized around a Kademlia based [11] distributed hash table (DHT) implemented using the overnet protocol. Each node is identified by a 128-bit overnet identifier (OID). At startup, a node chooses its OID pseudo-randomly and proceeds to find its neighbors (in the space of OIDs). The search is boot- strapped from a list of peers included with the binary itself. The node then advertises itself to its neighbors in the overnet, and continues advertising itself throughout its lifetime to maintain a presence in the network as its set of neighbors changes. To insert a key-value pair into the DHT, a node publishes the key, which comes from the same 128-bit OID space as node identifiers, to a node with an OID close to the key. To retrieve the value associated with a key, a node searches for a node close to the key; if a node has the associated value, it advertises the value to the searching node, which then requests it. Because every Storm node participates in overnet, enumerating its participants is a logical means of identifying bots and determining the botnet's size and composition. However, not all participants can be trusted implicitly; Storm has both been the target of attempts to disrupt the network via poisoning, as well as the victim of buggy implementations which report participants incorrectly. Each of these properties make identifying the true population of the botnet non-trivial. Based upon the protocol and correct functionality of individual bots, one [10] has developed a number of heuristics for removing invalid nodes from our participant list. Doing so also allows [10] to much more accurately estimate the size of the network. While reverse-engineering the algorithm Storm uses to generate OIDs, [10] discovered that it can generate only a very small fraction of the possible OIDs. Although an apparent flaw in Storm's OID generator, it is very helpful for a crawler because it provides a convenient means for identifying Storm nodes. Multiple nodes will join the network with the same OID at the same time, and therefore OIDs cannot be used to estimate the size of the botnet. Furthermore, deciding which node receives a message addressed to an OID with collisions becomes nondeterministic. It depends on which peers receive published messages, and how those peers deal with aliasing in their routing table implementation. To mitigate the problem, a crawler should search for an OID at several different points in the network. The initial purpose of Storm drain was to identify infected nodes at our institution to maintain a clean network. The crawler bootstraps onto the network using a list of peers either from a

previous crawler instance or a list of peers extracted from a Storm binary. The crawler then contacts every peer on its internal list and requests a list of additional peers to increase its knowledge of the connected peers. After communicating with all known peers, the crawler performs a pruning subroutine to determine which peers to remove from its contact list either because they are no longer responding, are incorrectly advertised addresses, or are possibly attempting to poison the network. After pruning, Storm drain repeats the process continuously, and periodically dumps its internal state. Storm drain continuously sends messages to peers to monitor their responsiveness, contacting each peer roughly every 30 seconds. How peers respond to those probes determines how they transition among states in the machine. Storm drain learns of new nodes as it crawls the routing tables of peers on the network, and when it receives unsolicited messages from other peers. When a new peer replies to a probe, Storm drain moves it to the live state. If a new peer does not respond to any probes, Storm drain places it in the removed state. If a peer responds to a sufficient number and kind of probes, the peer moves from live to active. If an active peer falls below that threshold it moves back to the live state. In [10] have been used different kinds of messages. It required multiple responses to help differentiate between actual Storm nodes and masquerading active responders. Storm drain moves a live peer that has not responded after a timeout expired to the dead state (there are many reasons why a node may not respond). A dead peer that responds before being removed moves back to the live state. Storm drain currently uses a timeout of 15 minutes based upon our experience. Storm drain probes dead peers at a lower rate. If it does not respond after a timeout, it moves to the removed state. An active peer that appears to be abusing overnet (flooding, poisoning, broken implementation, etc.), moves to the removed state immediately and bypasses any other state. Any short sequence of probes to a peer generates ICMP error. Responses moves peer to the removed state, again bypassing any other state. If a removed peer that was previously dead starts responding again, it moves back into the live state. Storm drain clears all statistics and counters for the peer and treats it as if it were a new peer. The set of nodes in each of these states captures the activity of nodes in the Storm botnet. The set of new nodes are those nodes which have been advertised to Storm drain but have yet to respond to a probe. The set of live nodes are those nodes which are responding to Storm drain probes, but have not responded sufficiently well to a range of messages to identify them as Storm participants. The set of active nodes are those nodes which properly respond to a range of Storm messages and appear to be active participants in the Storm botnet. The sets of dead and removed nodes track expired nodes which can become live again (dead) or be removed from further tracking (removed). An immediate consequence of Storm

drain node tracking is that we can estimate the number of active participants of the Storm botnet. Note that Storm drain actually tracks two independent variants of the Storm network simultaneously. They are the remnants of an older version of the Storm implementation and the botnet formed by the current Storm implementation. One unique challenge to tracking the Storm botnet is that Storm and other applications use the same overlay network protocol, creating a protocol aliasing problem. Storm shared its overlay network with users of overnet-based file-sharing programs such as MLDonkey. As a result, a crawler needs to differentiate between nodes participating in the Storm protocol and other applications. Simply identifying participants in the network as Storm would include the file-sharing nodes, thereby overestimating the botnet population. During the evolution of Storm drain, we explored two heuristics for differentiating Storm nodes from nodes using other applications. One technique is to classify nodes by their usage of the protocol and by the content hashes they publish and search for. Overnet published messages can contain metadata tags attached to key value pairs. File sharing applications can publish content hashes with metadata, but all but a few versions of Storm do not. Consequently, a crawler can mark the nodes publishing or searching for these hashes as non-Storm nodes. The heuristic is transitive: when a node is marked as non-Storm, the crawler can mark other hashes that the node searches for and publishes as non-Storm hashes. Any node that searches for or publishes those hashes is non-Storm as well. With this heuristic, the more time spent crawl the network, the more comprehensive the set of known hashes becomes. Conversely, we can also positively identify Storm nodes by their use of hashes specific to Storm. When a newly infected Storm node starts, for example, it searches for well-defined hashes to rendezvous with the rest of the Storm botnet. [10] reverse-engineered the algorithms used to encode and decode Storm key-value pair content hashes. Storm drain can therefore identify hashes as known Storm content with very high probability, and mark any node searching for or publishing those hashes as Storm nodes. Once we reverse-engineered the Storm OID PRNG, these heuristics were no longer necessary. An overnet message containing one of the 32K Storm OIDs immediately identifies that node as a Storm node. For evaluating [10] methodology, however, [10] can use the set of Storm OIDs as an oracle to estimate the accuracy of the above heuristics. For instance, we examined a snapshot of 24 hours of nodes observed on the overnet network that are subject to the protocol aliasing problem between Storm and file-sharing applications. Using the known Storm OID set as an oracle, 35% of those nodes are infected with Storm. The metadata heuristic greatly overestimates the set of Storm nodes as 94% of all nodes. The known Storm hash heuristic, however, severely underestimates the set, only identifying one node as a Storm node. Neither

heuristic effectively resolves the protocol aliasing problem, motivating the other heuristics we developed below. Storm began to encrypt its messages using a simple transformation with a static key. With this change, Storm bifurcated into encrypted and unencrypted networks: newly infected nodes communicated with each other using encrypted messages, and previously infected nodes were stranded. This older version of Storm remains active today, however. At any one point in time, nearly 5,000 hosts remain infected with the older form of Storm communicating using unencrypted traffic. Storm drain actively tracks nodes in both variants of the Storm network. Fortunately, the use of encrypted communication conveniently solves the protocol aliasing problem: all nodes using encryption use the Storm protocol. With the reverse-engineered key, we could readily separate Storm nodes from other nodes in the overnet network. However, although nodes on the encrypted network communicate using the Storm protocol, not all of those nodes are actually hosts infected with the Storm bot. Another unique aspect of the crawling a botnet like Storm is the challenge of differentiating between actual Storm nodes and other nodes that masquerade as Storm nodes. In [10] this effect is called adversarial aliasing. Over time Storm has received significant attention from other parties. Actions of these parties significantly influenced the act of measuring Storm. Just as [10] crawls and monitors Storm, so do other groups. Storm has also been the target of attempts to poison or pollute the network to prevent it from functioning efficiently, or at all. As one example, earlier in Storm's life cycle, updates to the binary were disseminated by publishing URLs to a set of hardcoded keys. By advertising non-existent nodes close to those keys, it was theoretically possible to prevent nodes from finding the published URLs and thus prevent them from downloading updates [12]. As another example, the connectionless nature of UDP allows messages with spoofed source addresses to reach the application layer and force peers to consider this IPs as possible nodes in the network. Finally, when a node trusts unsolicited published messages, an attacker can overwrite the value stored at the key and pollute the network. Network poisoning, for example, remains an active phenomenon. Just as Storm drain has to take into account the presence of other active participants interacting with the Storm network. Storm drain itself has an effect on the network, too. Storm drain selects a random OID on every query and response that it sends, and these OIDs will enter the routing tables of other nodes participating in the Storm overnet that Storm drain interacts with directly or indirectly. As such, Storm drain will appear to be millions of bots with random OIDs over time. Although the impact might appear substantial, other participants tracking nodes in the network can easily identify Storm drain as an invalid node. Storm drain does not spoof its source IP, and manual inspection of the source of such hashes would identify Storm drain as an incorrectly functioning

node. In fact, another instance of Storm drain on the current network would be quickly removed for flooding OIDs and would not have an impact on our estimate of the network size. The active node population of Storm is constantly changing over time: hosts arrive and leave continuously. Learning about new hosts is straightforward. Storm drain actively crawl the routing tables of peers in the network, and continuously receives unsolicited messages advertising hosts. However, when tracking the active population of bots, it is crucial to also track when nodes become inactive. Peers can become inactive, or dead, for a wide variety of reasons: they can be NATted nodes behind silent drop firewalls, previously valid peers which have been cleaned or shut down, currently active peers, which have been restarted and have chosen a different port for communication, or invalid addresses sent by a buggy or malicious poisoning node. In a similar vein, nodes which had previously responded to Storm drain probes and fell silent could be due to rebooted or cleaned machines. It is also common that a NAT device in front of the peer has lost the mapping for communication between the infected machine and Storm drain. The Storm overnet implementation includes no method of removing bad hosts from its internal peer list. As a result, Storm drain has to determine when peers expire and much of the development effort put into Storm drain has been to improve the accuracy of pruning dead nodes from the list of active nodes. Storm drain repeatedly sends messages to its list of known nodes and tracks which nodes respond. The data structure for connected peers includes a counter of unanswered messages for every peer. Peers which have never replied are removed after a handful of unanswered messages, whereas peers, which had previously responded and are considered as connected are given more leeway in terms of the missed replies. If messages sent to a peer result in more than a few ICMP errors of any sort, Storm drain considers the host unlikely to be able to communicate and discards it. While the removal of false positives like poisoned data and stale mappings is an important facet of gaining an accurate view of the network population, ensuring that the crawler does not remove valid peers is equally important. The problem is that valid peers may not be able to always receive and, hence, respond to probes from Storm drain. Lacking responses to probes, Storm drain may then decide that those peers are dead when in fact they are still active participants in the botnet. While [10] does not have any evidence that valid peers will refuse to respond to valid queries, NATs and firewalls can refuse to forward our traffic to nodes which are behind them. This dropping can either happen because the node has never contacted us before, or the mapping between the external port and the internal node has expired and communication is no longer possible. When a node, even one which is not responding, is added to the Storm drain state machine, we keep track of how many times this node has been

advertised to us, and whether it has been advertised by an active node. If a node is behind a NAT and has not been informed of the presence of our crawler by one of its peers, [10] will be informed of its existence by active nodes but will not be able to communicate with it directly. Conversely, if a node is behind a NAT which times out UDP ports rapidly, it will be continually reported to us as active, but our crawler's mapping in the NAT table might be lost and, with it, our ability to communicate with this node. Although second-hand knowledge of NATted nodes is helpful, if we were able to convince these nodes to initiate contact (for the former case) or continually contact Storm drain (in the latter case), we can form a much more informed estimate of which nodes are behind such NATs. For the latter case of rapidly remapping NAT devices, the problem is that nodes do not query Storm drain often enough to maintain the connection within the device's NAT table. One key deficiency of earlier versions of Storm drain in this case is that, while it sent queries and listened for responses, it did not send responses of its own to nodes which query it. Having Storm drain respond to nodes which query it refreshes NAT and firewall timeouts, thereby enabling Storm drain to probe and track those nodes more reliably. For the former case, we devised another heuristic to elicit new connections from nodes with which [10] have never had any direct contact. This heuristic takes advantage of semantics of Storm node behavior. Over time, [10] found that certain regions in the ID space (with locality defined by the XOR metric) would become very popular for a short period of time. To make it more likely that NATted nodes contact Storm drain, whenever Storm drain noticed these popular hashes it chose for itself an OID in that region while crawling the network. These two techniques had a noticeable improvement on Storm drain's ability to track NATted nodes. Even with these heuristics, there could still be errors in Storm drain's estimates of active nodes. To estimate the potential magnitude of this error, we can use counts of the number of nodes that stopped responding and never responded as upper bounds on the error — note, though, there are many reasons why nodes stop responding or never respond. The number of nodes that never respond is much smaller. Even if all of these nodes were actual Storm nodes that Storm drain could not communicate with, the error estimate of the number of active nodes would be less than 1%. What seemed an easy task, given Storm's use of a well-documented protocol overnet, was in fact significantly complicated by a range of factors including innocent applications using the same protocol, the impact of address translation, and the challenges presented by active network spoofing and poisoning by outside parties such as other research groups. The entire endeavor begs the question, how can one be sure in such an environment. It seems likely that the same kinds of problems we have encountered will continue to arise with new botnets, although they will undoubtedly have their

own intricacies and complexities. Unfortunately, it seems overly optimistic to hope that the techniques [10] has developed will prove universally applicable and thus our community should have considerable skepticism about the veracity of botnet measurement results going forward. Considering this [10, 12] offer a number of suggestions, ranging from the grounded and pragmatic to the wildly naive and optimistic. First, when evaluating botnets using existing protocols for their organization or command and control, researchers should be required to explain how they differentiate between bots and aliasing from other protocol users. Simply claiming that the botnet does not overlap with other applications is not sufficient since this can happen both intentionally from the bot master using the existing infrastructure to jump start their own or inadvertently via aliasing from NAT or simply a bot host that also happens to have the application on it. Second, researchers should be expected to document the key assumptions underlying their measurements. Several of these assumptions (e.g., bot ID persistence, bot ID uniqueness, IP address persistence and IP address uniqueness) should be considered fundamentally suspect in the same way that assumptions of Poisson packet arrival times trigger eye rolls in the networking community. Only when such assumptions are independently validated should results based on them be permissible. Finally, care must be taken not to inadvertently measure the activities of other researchers. Addressing this problem is perplexing because there is nothing fundamental allows one to make this determination; a sufficiently sophisticated researcher can design software that is indistinguishable from a bot at the network layer or simply infect a large number of honeypots with the bot itself and manipulate their behavior. Conversely, a bot herder wishing to conceal their activities might make subsets of their bots behave anomalously and thereby be mistaken for other researchers (and hence not be measured). In the end we have no perfect proposal. Life was decidedly easier when one could study malware in isolation. However, since network-persistent malware like today's botnets seems likely to stay, it behooves us to be aware of these problems and do our best to manage them.

References

1. R. Pang et al. Characteristics of Inernet Background Radiation. In Proceedings of the ACM Internet Measurement Conference (IMC 2004), October 2004.

2. J. Grizzard et al. Peer-to-Peer Botnets: Overview and Case Study. In Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets, April 2007.

3. P. Porras et al. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, Computer Science Laboratory, SRI International, October 2007.

4. S. Sarat, A. Terzis. Measuring the Storm Worm Network. Technical Report HiNRG 01-10-2007, Johns Hopkins University, 2007.

5. M. Rajab et al. My Botnet is Bigger than Yours (Maybe, Better than Yours). In Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets, April 2007.

6. S. M. Bellovin. A Technique for Counting NATted Hosts. In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, pages 267-272, November 2002.

7. R. Bhagwan et al. Understanding Availability. In Proceedings of the International Workshop on Peer To Peer Systems (IPTPS), pages 256-267, Berkeley, CA, February 2003.

8. M. Casado, M. J. Freedman. Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification. In Proceedings of the 4th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'07), April 2007.

9. J. Grizzard et al. Peer-to-Peer Botnets: Overview and Case Study. In Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets, April 2007.

10. Kanich C. et al. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. Proceeding LEET'08 Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats Article No. 10 USENIX Association Berkeley, CA, USA. 2008.

11. P.Maymounkov, D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In Proceedings of the 1st International Workshop on Peer-to-peer Systems, 2002.

12. Старобогатов Р. Scenarios propagation information through a large network of nodes. **http://econf.rae.ru/article/6973** (дата обращения: 23.08.2012).