Discovery techniques for P2P botnets

David Dittrich
Applied Physics Laboratory
University of Washington
dittrich@u.washington.edu

Sven Dietrich
Computer Science Department
Stevens Institute of Technology
spock@cs.stevens.edu

Stevens CS Technical Report 2008-4, September 2008

Last revision: 21 April 2009*

Abstract

Over the last few years, researchers and network operators have examined networks of DDoS agents, more recently called botnets due to their connection to Internet Relay Chat (IRC). In the continued quest to take down these networks of bots, two important questions arise: how many bots are there, and how to find every last bot? When one reads about a ten thousand, hundred thousand, one million node botnet, how much credibility does it have? Is botnet A really bigger than botnet B? The difficulty in accurately and stealthily assessing the size of the botnet often lies in the structure of the botnet itself, such as IRC, HTTP, P2P-based, or a hybrid thereof. We present a general overview of discovery techniques for networks of malware, and provide a glimpse at a two-year study of a P2P botnet.

1 Introduction

As the threat of DDoS agent networks or botnets increases, so does the concern of the capabilities, whether computational, bandwidth exhaustion, keystroke recording, or file sharing. There have been multiple accounts of large botnets, with sizes ranging from the thousands to even over a million. To what extent are these figures believable? Rajab et al. [25] have mentioned the problem of botnet size and quality already. Most botnet discovery, i.e. enumeration, techniques discussed publicly today are of the passive kind, such as traffic analysis. We would like to measure the size of a botnet, with an emphasis on the accuracy and stealthiness of the approach. In the case of P2P botnets, the structure of the botnet makes passive discovery more difficult.

This report is structured as follows: we present a general overview of discovery techniques for networks of malware, discussing the cases of DDoS agent networks and botnets in their various forms, such as IRC, HTTP, and P2P, and then provide a glimpse at a two-year study of Nugache, a pure P2P botnet, and the techniques used for long-term enumeration.

1.1 Related work

We could only find two other studies of P2P botnet enumeration by crawling, where Holz et al. [13] as well as Enright [6] crawled the Storm botnet, exhausting all possible connections. Earlier work by Stover et al. [27] analyzed the host and network aspects of both Storm and Nugache.

*Document history: Original submission June 2008, Technical report September 2008, revised 15 and 21 April 2009

2 General discovery techniques

2.1 Classic DDoS agent/handler setup

Early tools in the late 1990s were heavily centralized. A server (or handler) would coordinate the commands of the attacker and relay them to the individual DDoS agents. The key to finding all DDoS agents was the discovery of one or more handlers. Even if the list of agents was kept encrypted, it was possible to coax the information out of the handler by means of replay of the same command the attacker would send to the handler to get a status of the network [5, 18, 4]. Even if such a replay was not possible, simple observation of the network traffic close to the handler or in the Internet core revealed simultaneous connections, which resulted from a command or status request sent by the attacker.

2.2 Switch to IRC

With the switch to IRC, the attackers achieved two things: vanishing in the high volume of IRC chat traffic, and taking advantage of existing infrastructure with its advanced functionality, such as channel encryption. However, membership in a channel, being a bot, is equivalent to being part of the botnet. So simply hanging out in a channel and counting the bots would lead to the discovery of bots, and a rough calculation of their number. More advanced techniques such as channel hopping (moving an entire set, or perhaps a subset, to a new channel) or customized servers and clients contributed to the stealthy nature of the business and made discovery of bots more challenging, but still possible [26] As of 2007, discussions about "fast flux" networks [14] emerged and described hiding techniques for attackers that make it harder for network defenders to track down compromised hosts. The so-called "single-flux" and "double-flux" distinction in [14] is just variations of low-TTL DNS to IP mappings of A records with or without accompanying NS records and recursion to more participating DNS servers. The results are the same, in that a given query for the IP address associated with an A record will return apparently constantly-changing results for repeated queries from the same client Using DNS monitoring techniques that are well understood, "fast-flux" does not present an insurmountable obstacle to enumeration.

2.3 HTTP-based bots

Some bots use HTTP as a transport for the command and control [19, 21]. This is not to be confused with IRC bots that use port 80/tcp instead of the standard IRC port 6667/tcp, which is another technique to hide the IRC-based C&C traffic by mixing it in with normal HTTP traffic (e.g., GET, POST, PUSH requests, and the resulting HTTP responses to these requests.) These bots have used two primary techniques:

1. An HTTP GET request is used to retrieve a file from a specific URL. The resulting response – the contents of the file requested – contain a list of several (typically around 3) alternate locations where the bot can go to retrieve the actual address(es) and/or channels of IRC servers that act as the central C&C channel. HTTP is only used as a relay mechanism. HTTP is not used here for C&C, but rather for rallying the bots to the C&C location. The Storm trojan used a similar technique, only substituting a complex encoding of the location of the central C&C server using a P2P service rather than the multi-layered HTTP GET mechanism just described.
2. The HTTP GET request itself, often from a PHP script that back-ends the C&C server infrastructure, returns an encoded response that *is* the command, just as if it were obtained by the bot monitoring an IRC channel. [19] In this case, HTTP replaces IRC, and the back-end PHP script serves the same purpose as the IRC server formerly served, that of transmitting the commands from miscreant to bots.

In those cases, the C&C communication can easily be intercepted and analyzed. As with early DDoS tools, some bot would generally phone home and register/poll the C&C server. However, the botnet-related HTTP requests do vanish in the sheer volume of true HTTP requests in web server logs. With known signatures, one could try to extract the C&C traffic from a conveniently located observation point, e.g. close to the core. An obvious hindrance to the passive discovery of content of the C&C channel is encryption. However, one could still observe all packet flows to a known C&C server (or multiple servers) despite encryption of the content, and count the connections (discounting any dynamic IP addressing, see the discussion in section 3.5.1).

2.4 RSS feeds, or other "pull" techniques

Researchers have proposed that the next threatening move in botnet C&C may be RSS feeds, blog postings, or other *Web 2.0* mechanisms. [8] While these methods may provide obfuscation of the commands, any *Web 2.0* method of C&C by nature requires a central C&C server. All agents must know where to find the commands, and must pull them at some frequency in order to determine when new commands exist. While RSS feeds may provide a mechanism for attackers to push a new command block to the RSS servers, it still remains that all agents must subscribe to the feed – which is a pull operation that must be repeated on some schedule – to know when a new command may be available. This results in significant latency and asynchrony, but more importantly retains the fundamental problems of a limited number of C&C servers, traffic patterns that don't scale well and can easily be detected.

Given the limitations just described, these types of C&C would not be useful for attacks that require direct and tight control, such as extortion attempts, or DoS attacks coordinated manually with physical events, or that change the nature of the attack in response to defensive countermeasures. In these cases the risk of C&C server detection and takedown is the same as with central IRC C&C and requires the attacker also use some other method for concealing their own connections via proxies or stepping-stones. [20]

On the other hand, for attacks that do not require direct and immediate command and control, such as the retrieval of keystroke log files, exfiltration of documents or other private data, or attacks that can be planned and timed weeks or months in advance, these types of *pull* techniques could blend in with normal RSS or blog reading activity that typically occurs. This assumes that subscription to RSS feeds and/or blog post reading is a normal occurrence on the compromised host, which may not always be the case. If it is not, even a once-a-month RSS feed check will draw attention as the only activity of its type on those networks that monitor connection flows.

2.5 Pre-P2P approaches for counting

There are several distinct phases that any distributed malware operator must go through, repeatedly, in order to build and maintain a sufficiently large botnet for nefarious purposes. In [18], the authors describe three basic phases: *recruitment*, *control*, and *attack*.

- In the *recruitment* phase, the new host is compromised by any means, such as by taking advantage of an exploit or by social engineering. Once compromised, the host will then phone home, either by a classical DDoS network, an IRC network, or an HTTP C&C server. The phoning home aspect assumes a fixed address, leading to a weakness in the system: it can be tracked. If the fixed address is an IP address, then it can easily be observed, throttled, or blocked. If the attackers choose to be a bit more flexible, e.g. playing with Dynamic DNS or "fast-flux", they can use their own domain names, own IRC servers, and take advantage of DNS manipulation attacks (e.g., modifying locally stored host files mapping DNS names to IP addresses on the compromised computer.) More advanced techniques, such as fast flux [11, 24], similar to Dynamic DNS (TTL-based

or server-based), make identifying the C&C server relatively harder to track. However, the same fundamental problem exists that all requests channel back to a fixed set (possibly just one) C&C server or servers. The bots can be counted during this phase if the malware is being downloaded from a central cache.

- In the *control* phase, the attacker uses direct or indirect mechanisms to pass commands along to the bots in order to control them. Part of keeping control of the bots involves moving them around to prevent them from being detected as a group and taken out of service. With respect to IRC-based botnets, this is known as *herding* the bots. If the C&C method being used involves a single, central control channel, then herding must be done in such a manner that new recruits can be managed along with the previously held hosts. Again, that could mean pointing to a domain name or IP address, or perhaps even using multiple IRC servers/channels to hold multiple groups of bots (as opposed to keeping them all in one large channel, where it may be possible to mitigate the entire group all at once.) This migration during herding is noted by [26]. You can count the bots if they are parked in an IRC channel, for example.
- When the attacker is ready to perform some malicious act, they begin the *attack* phase. This can involve many actions, from recruiting additional bots to replenish the botnet(s), retrieving keystroke log files, performing a DDoS attack, engaging in click-fraud, etc. As botnets degrade over time, additional recruitment operations will constantly be necessary. Using the botnet to perform the recruitment directly increases the chances of detection and mitigation, and also may affect enumeration as the size of the botnet may be changing while enumeration is being performed. If all bots are participating in a flooding or a click-fraud attack, then they can be counted

2.6 P2P networks

In the P2P setting, centralized structures don't exist. The structure, albeit varied, allows for some clustering, but there still is no central authority other than the botmaster or botherder. In most cases, P2P bots talk to a subset of the network, the set of immediate peers.

Discovery of the botnet is now limited to enumeration via a bot-by-bot approach, unless the bots participate in a global botnet activity. Each bot has a limited view of the botnet by design (such as a top limit of peer addresses retained) or by limitation of the network (firewalling or NATing).

As a general technique, one must visit each bot, request a list of peers known to that peer, and build up a prioritized queue to walk through the entire set, removing duplicates. However, there is the issue of hidden bots, such as those behind NAT, for which their presence is only known to certain bots. These hidden bots will only initiate connections (and act as a polling bot), but not receive them, which causes an obstacle to an accurate count. The hidden bots may have knowledge of bots that could act as a bridge to an otherwise completely disconnected botnet.

In some cases, encryption is used in this information exchange. A passive observation of P2P traffic will therefore not easily yield information about the complete network, because peers only maintain knowledge of their immediate neighbors. Only the connection data, e.g. link analysis or who talks to whom, will then provide partial network knowledge.

So what to do? One can then:

1. Fire up a bot on a real host and participate in the botnet. This seems to be the most accurate approach, as it represents a realistic scenario. For enumeration purposes, even after extraction of P2P connection data or content, that provides a limited view of the P2P network.

2. Fire up a bot in a virtual machine (or multiples thereof). The same concerns as in the previous case exist. An additional concern here would be virtual machine detection by the malware, leading to malfunction or modified function of the bot.
3. Create a client that mimics the protocol used by the botnet. That assumes that the bot has been completely reverse engineered. However, this may prevent adaptation by the enumerator to new malware code being downloaded in order to stay connected to the botnet.

If sufficient knowledge exists, then the last option appears to be the most appropriate one in conjunction with one or both of the remaining ones.

2.7 Accuracy and stealthiness

Some concerns arise from discovery techniques:

1. *Accuracy.* How does one count the number of bots correctly? How much of a miscount exists between the bots discovered today and the ones discovered the day or the week before? Some causes for miscount are the dynamic nature of IP address assignment by Internet Service Provides, also known as DHCP. Some botnet clients allow for unique identifiers (UIDs) that identify a host across DHCP address changes. If such UIDs are exchanged, directly as part of the communication protocol or indirectly via enhanced communication, then one can observe them and present a more accurate count.The remaining inaccuracy relates to machines simply being offline or hidden behind firewalls or NAT devices.
2. *Stealthiness.* A major concern is the stealthiness of the approach. Firing up hundreds or thousands of virtual machines (VMs) containing a bot may taint the accuracy of the count, if not for the discoverer firing up those VMs, but perhaps for some other observer. The goal is not to distort the count of bots. A reasonable figure seems to be at around 1%, since the accuracy of the count would not be affected by such a skew with botnets sized around 10000 bots[1]. The volume of traffic should also not increase significantly either.

3 Application to Nugache

We decided to apply the aforementioned techniques to Nugache [27, 9], a P2P botnet. The approach varied over time, but it was meant to remain as accurate as possible. We used a conservative approach with regard to stealthiness, specifically geared towards avoiding pollution of the botnet count.

3.1 Background on Nugache

Stover et al. [27] already dissected Nugache, so we will limit ourselves to a quick summary of its communication means. Nugache is a P2P bot that uses random high-numbered ports for its communication over TCP. Moreover, each connection from peer to peer is encrypted using Rijndael 256, with keys exchanged via an RSA-style public key mechanism.

3.2 Experiment setup

Early observations of Nugache in May 2006 showed very few new network connections in any given day. Within a few weeks, it was obvious that network traffic analysis alone would be a very ineffective means of enumerating the complete Nugache network.

[1]This assumes that there is not a large number of uncoordinated activity amplifying the distortion

After initial reverse engineering, the key exchange and encryption mechanisms were understood. Further analysis derived a nearly complete understanding of the internal command and control features, and a partial understanding of the human-readable command set. It was determined that a custom client could be created to allow connecting to active Nugache nodes and issuing a limited set of commands to the node.

This custom client, implemented in Ruby, was run within a Windows XP Pro virtual machine guest, but it also runs under Linux and MacOS X. Once this custom client was tested, several experiments were performed to investigate mechanisms for accurately determining the size of the complete P2P network.

The enumeration tool ("*PeerA*") connects to specified peer(s) ("*PeerB*"), establishing an encrypted session, and querying PeerB for (a) its currently connected client list[2], (b) the list of peers known to PeerB (i.e., those stored in its Windows Registry), and (c) the release version of the Nugache executable that PeerB was currently running.

It should be noted here that our tool, for the purposes of these enumeration experiments, was designed explicitly to *not* insert its own IP address in the Registry of peers it communicates with, nor to do anything that would be noticeable to the botnet operator.[3]

Algorithm 1 Algorithm for Nugache Enumeration

```
while True do
  queue ← read(defaultNodeList)
  while not empty(queue) do
    peer ← popHead(queue)
    if connect(peer) then
      connectList ⇐ getConnected()
      knownList ⇐ getKnown()
      version ⇐ getVersion()
      disconnect(peer)
      for all p in connectList do
        pushHead(queue, p)
      end for
      for all p in knownList do
        pushTail(queue, p)
      end for
      report(peer, connectList, knownList, version)
    end if
  end while
end while
```

3.3 Enumeration experiment #1

The first experiment was to determine the relationship between the peer list stored in the `HKEY_CURRENT_USER\Software\GNU\Data` registry sub-tree of a peer we controlled and the peer list retrieved from a set of remote peers, and whether the results are random or fixed. It was

[2]Note that this returns a list of only those peers who initiated inbound connections *to the bot*, not including any outbound connections initiated *by the bot*

[3]Of course the high-number of connections over time *is* noticeable, using the methodology we used for these experiments. More stealthy ways of accomplishing the same task are possible.

also a development test of the enumeration program itself. Because this experiment was a *beta test* of the enumeration tool, we don't trust the results very much, however the experimental runs did allow us to gain confidence in the procedures we were developing.

The following steps were then performed:

1. The `HKEY_CURRENT_USER\Software\GNU\Data` registry sub-tree on an active peer was dumped to a text file.
2. An ICMP packet with a fixed byte pattern (0x00FF) was sent to a non-existent IP address in order to capture it with the honeywall for timing purposes.
3. PeerA was then used to contact PeerB and request from it a list of peers.
4. A second ICMP packet with fixed signature (0xFF00) was sent.

The list returned to PeerA from PeerB was then compared with the previous contents of the `HKEY_CURRENT_USER\Software\GNU\Data` registry subtree. There was a difference between the list retrieved and the registry, which is mostly likely due to latency in the registry being updated.

This experiment was repeated 10 times. The results showed a significant variance in the list of known and active peers. We observed a minimum of 6,278 connectable (i.e., servent) peers, and a maximum of 9,465 connectable peers.

3.4 Enumeration experiment #2

The second experiment was to estimate the time required to dump the peer lists from a fixed set of peers seen recently in network traffic (implying they would likely be up and running at the time.) The list of peers obtained from the registry dump in experiment #1 was used.

PeerA was set up in a similar manner as in experiment #1 but this time IP address was set to match that of one of the two controlled peers. This would decrease the possibility that the connections would be seen as anything other than normal peer activity. The actual peer being used to monitor traffic was disconnected from the network, but not shut down, so as to not affect its uptime.

The following steps were then performed:

1. An ICMP packet with a fixed byte pattern (0x00FF) was sent to a non-existent IP address in order to capture it with the honeywall for timing purposes.
2. PeerA was then used to iterate through the list of peers known to PeerB, requesting from each a list of peers they know of.
3. A second ICMP packet with fixed signature (0xFF00) was sent.

The resulting sets were compared to determine any overlap.

This experiment was then extended to a continuous run, arbitrarily chosen to go from 6/23/2007 through 7/10/2007. During this experiment, 90 samples were obtained. Each sample run took a little over 1 hour to complete. Figure 1 graphs the results of this enumeration run. It shows a clear diurnal pattern of peer availability and footprint, which matches observations made in 2002 by the Internet End-to-end Performance Monitoring group at Stanford University, [1] in February 2006 by researchers at Cornell and Google in relation to the Skype P2P VoIP network. [10], and by Dagon et al. [3].

In Figure 1, the top line represents the unique population of peers known to be available at one time as *servents*. These are the hosts kept in the Windows Registry of each bot. This population comprises the known *footprint* of the botnet at a given point in time. The middle line is the list of peers that are reported to be actively connected as clients to all the bots we were able to successfully connect to and query, which is the population shown in the bottom line.

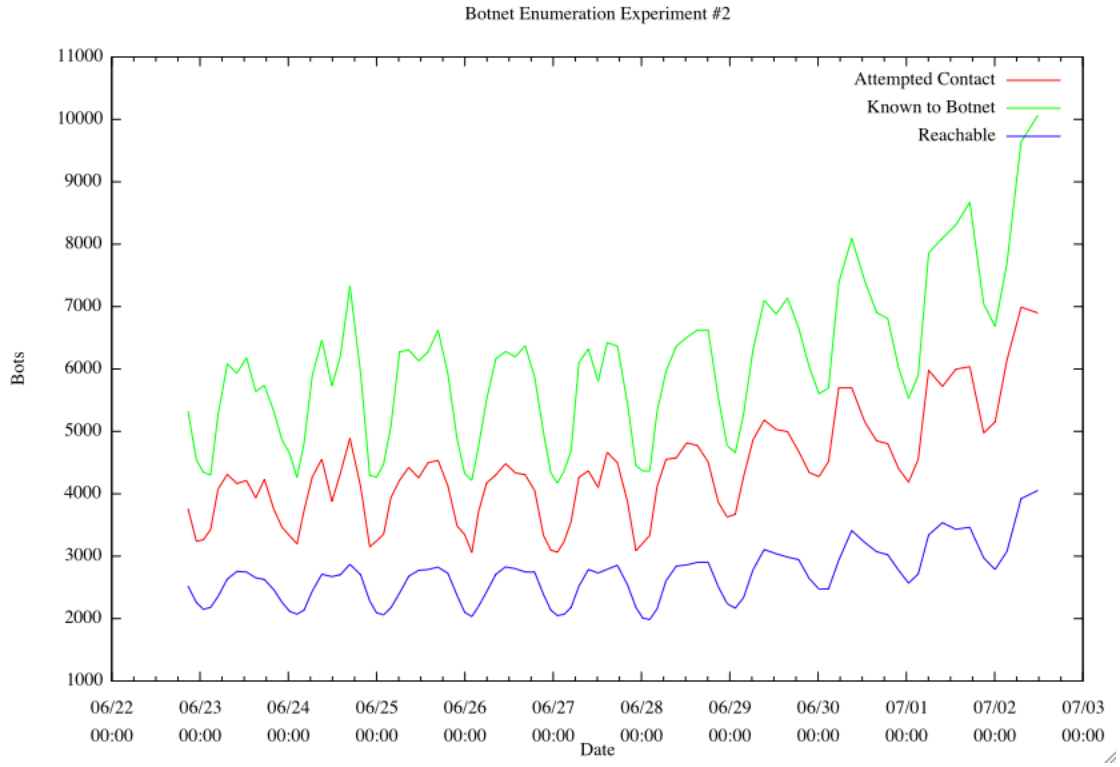


Figure 1: Results of Second Enumeration Experiment

There was no observable activity (that we can identify) related to propagation of the botnet during this time, however there is a clear increase in all populations seen during this time period.

3.5 Enumeration experiment #3

Starting in August 2007, a constant series of enumeration runs was begun, using as a seed list the 160 peers that were found to always be up and available during every run of the second enumeration experiment. We stuck with this list to ensure consistent seeding across all runs. Another enumeration node was added in January 2008 using the same code. This provides a different vantage point for network structure analysis, and allowed us to validate the results of the initial experiments. We also used this opportunity to verify that we were getting consistent results from both vantage points, and that we could get similar results by doing a recursive run using just one active peer, rather than the 160-peer seed list of the ongoing experiment. The results were consistent, give or take a peer or two coming online or going offline.

The algorithm for this enumeration experiment is shown in Algorithm 1. In order to optimize the speed of enumeration, we chose to use a priority queueing algorithm that favors hosts that are known to be up and active. Whenever a connection is successfully made to a peer, the set of reported actively connected peers are placed at the head of the queue, and the list of known peers that would be used on reboot to reconnect to the network (i.e., hosts that may or may not be currently active) are pushed onto the tail of the queue. As of late June 2008, it took somewhere between 30 to 45 minutes to complete each run, yielding about 30 to 45 snapshots of the P2P botnet per day.

Figure 2 graphs the results of this run, with a geolocation snapshot of all bots seen in Figure

7. Clearly, the botnet is degrading over time. Its apparent half-life is on the order of 4 months. We have not witnessed any propagation of commands through the network since the run began, and no activity related to propagation has been witnessed independent of the Nugache network itself (e.g., the web server associated with the download dropper had previously been removed from service, and no further blog postings can be linked with Nugache propagation.) There is no reason we can discern from our data to explain this apparent abandonment of the Nugache network. We will not engage in any speculation as to a cause.

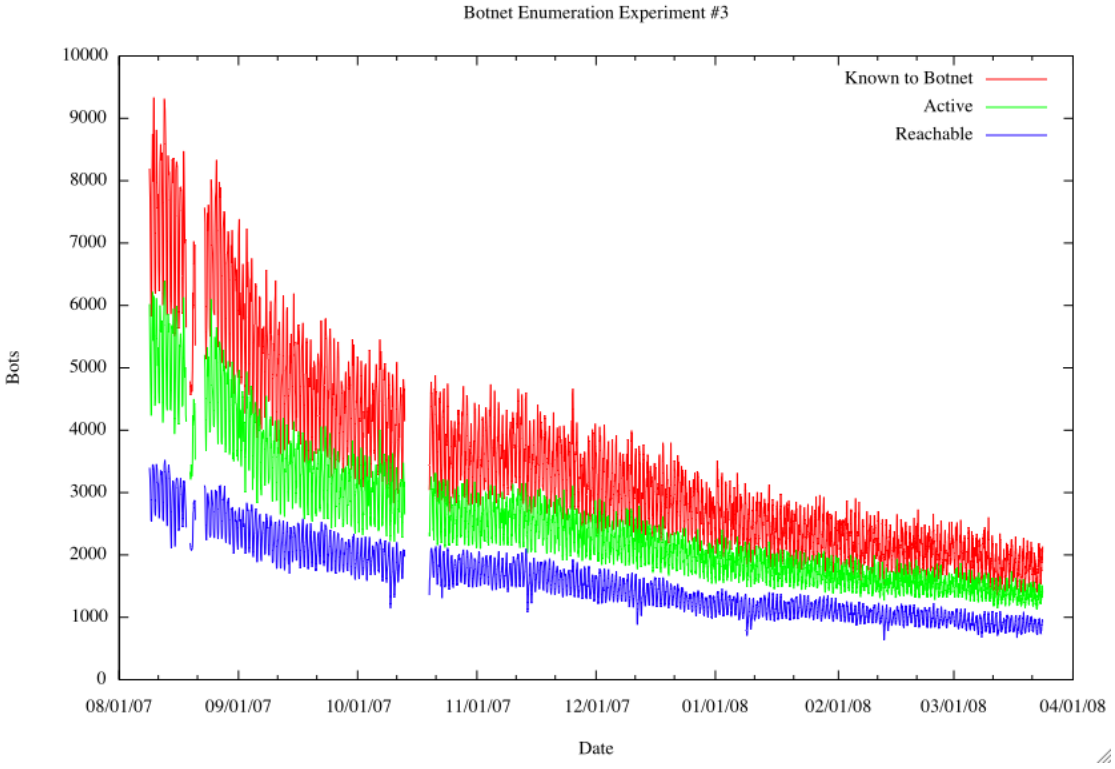


Figure 2: Results of Third Enumeration Experiment

3.5.1 Further analysis of Experiment 3 data

We have not fully analyzed the data that has been collected to date, however some initial inquiries into the nature of the topology and population characteristics have been made.

For example, the mean number of clients actively connected to any given peer was found to be 6, while the mode was found to be 5. The minimum number of clients reported was 1, and the maximum reported was 15. A breakdown of reported clients (the in-degree for any node in the P2P network) is shown in Figure 4(a).

Analysis of reported versions shows that while the vast majority of bots active today are running the latest release version (21) from January 2007, there are still some spurious peers that show up now and then running older versions, in some cases going all the way back to version 11 from June 2006 (about two years old). The breakdown of reported versions is shown in Figure 4(b). It is not clear why a handful of peers were not able to successfully upgrade themselves, as typically occurs when peers connect to each other. This may be due to a bug, or an environmental problem with the infected host (e.g., insufficient write permissions to one

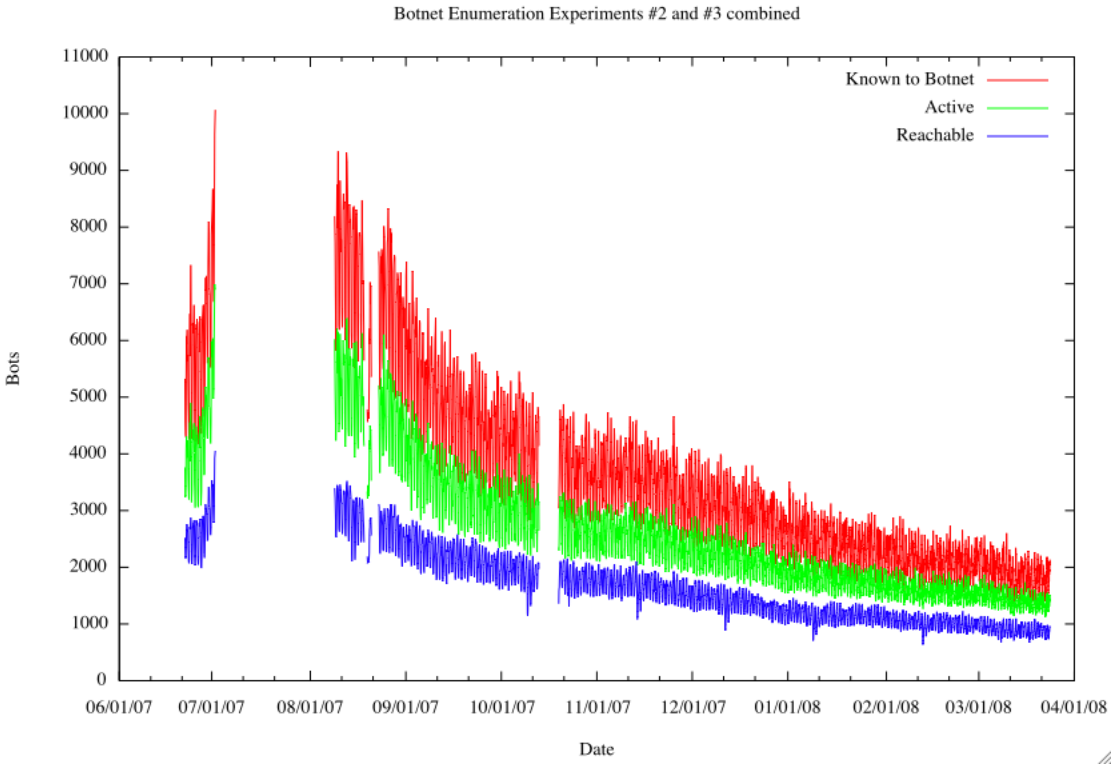


Figure 3: Results of Second and Third Enumeration Experiments

or more directories, a filled partition, anti-virus or other defensive software installed *after the initial infection* that is not able to identify and cleanup the existing infection, etc.)

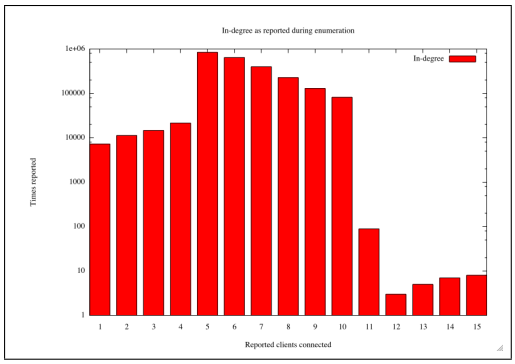
From this result, it appears that there is a hard-limit of 10 inbound peers, but for some as-yet unknown reason there can occasionally be seen up to 15 inbound peer connections. Observations of network connections within one of the two peers controlled by our team, which was running a version 21 sample, show similar patterns of occasional in-degrees of greater than 10. In one case, we witnessed a total of 18 connections, 15 of which were initiated inbound, however the majority of the time there were 9 or 10 connections. Such discovered botnet structures are shown in Figures 5(a) and 5(b).

This also indicates that there are no *super-nodes* in the Nugache network that have extremely high in-degrees of connectivity as is found in other P2P networks, such as Gnutella [17] (see Figure 6 for an illustration from [17] of the structure of the Gnutella network) or Skype [10]. While it is true that some Nugache nodes have high up-times, we have observed no evidence through reverse engineering or direct observation that there is any hierarchy of servents, only a differentiation between nodes that can accept connections (i.e., the servents) and peers that can only act as clients.

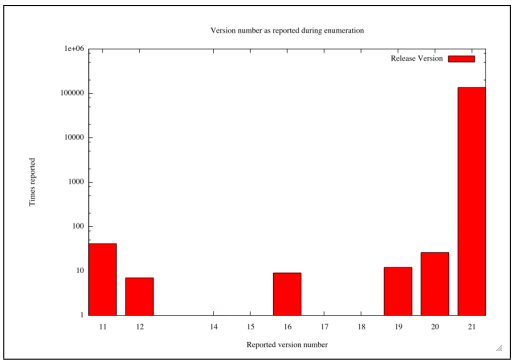
As far as the list of known potential servent peers kept in the Windows Registry, we noted some other anomalies. Peers that were running Nugache release version 21 almost never reported more than 100 known peers,[4] while some peers still running older versions of Nugache (e.g., version 11) would report as many as 1233 known peers. [5] This may have led some researchers

[4]There were some reports of 101 known peers, which we are not able to explain at this point.

[5]The appearance during the experiment of these hosts with a much larger than normal list of previously seen peers

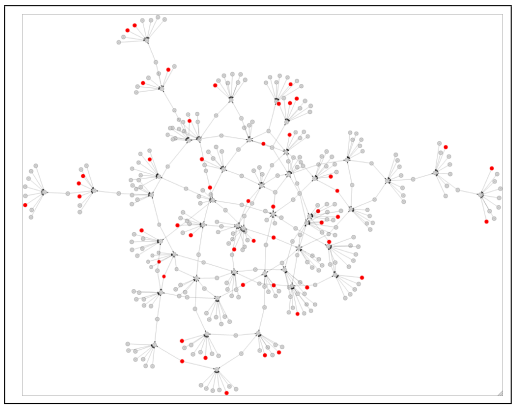


(a) Reported in-degree of nodes

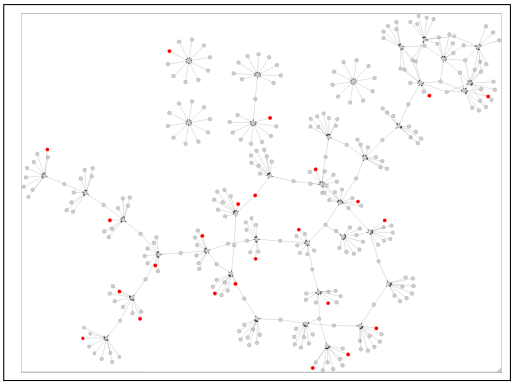


(b) Reported version numbers

Figure 4: Reported In-degree and Release Versions during 3rd enumeration experiment



(a) Nodes with degree 9+



(b) Nodes with degree 10

Figure 5: Discovered botnet structures during the Third Enumeration Experiment

who only performed a cursory analysis of their data to an incorrect conclusion that Nugache uses a scale-free architecture and has, "[an] inner ring ... (created from the hard-coded peers), where we would expect to observe a very high link degree." [2]

What is more, the methodology used by some researchers to "hide" behind frequently changing IP addresses, or use of multiple concurrent instances of peers, can have a polluting effect on the botnet's concept of "active" peers. [2, 26] Other researchers have gone so far as to experiment with purposefully polluting the peering indexes in P2P botnets [12]. Actually doing this should require careful consideration. The issues of count inflation and other impediments to accurate size estimation are described in [25], and deserve much more critical attention and research.

We note that other researchers were attempting to observe the Nugache network during the same time period as one of our enumeration experiments, using a method that involves creating hundreds and hundreds of virtual bots with changing IP addresses for these false peers. [2] There is a clear increase that can be seen in the number of connects per half-hour between 6/30 and 7/2 in Figure 4(b) of [2], which correlates with the increase in the slope of the population curves in Figure 1. The methodology used in [2] may have had a polluting effect on the Nugache peers, so the increase we observe may not be the result of actions by the operators of the Nugache

accounts for the upward spikes of approximately 1000 peers seen in the top line in Figures 2 and 3.

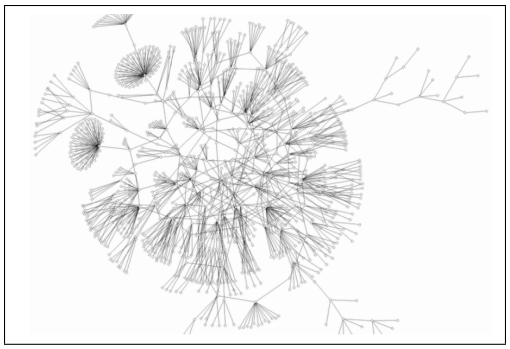


Figure 6: Example from [17] of Gnutella's network structure

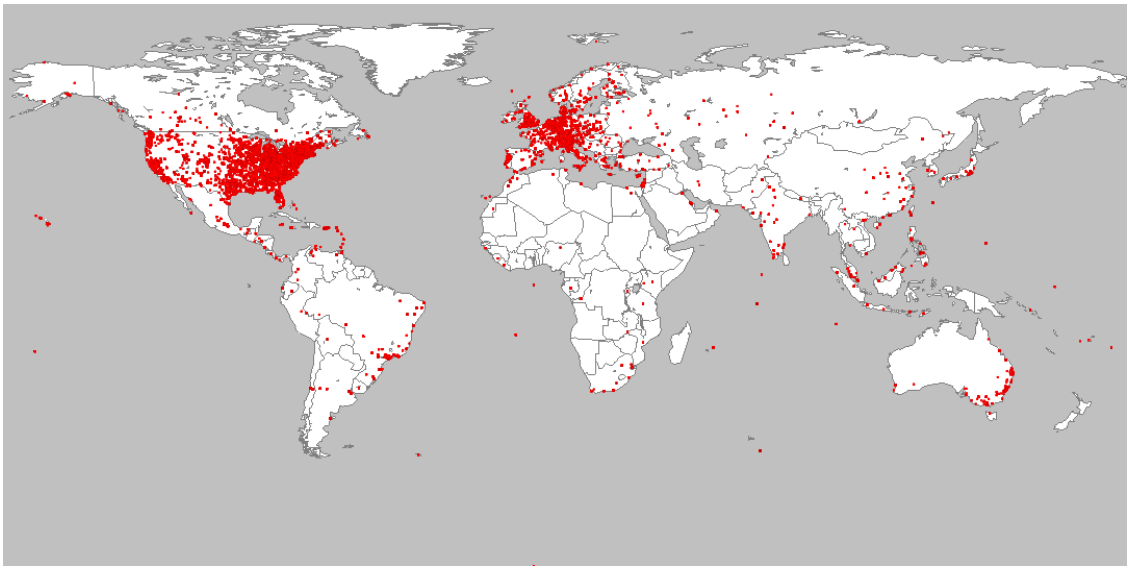


Figure 7: All bots by geolocation from the Third Enumeration Experiment

network.

These techniques may already account for wide discrepancies in the estimated size of various botnets seen in the media. [16, 28, 15] With so many groups taking uncoordinated actions, with noticeable effects, it is only a matter of time before problems occur.

For example, one possible problem would be the effect of a researcher inflating the perceived size of a botnet that is the subject of a criminal investigation. If such a case resulted in a successful prosecution, and a damage estimate were to be derived based on the inflated count of "infected" hosts, multiplied by some estimated cost-of-cleanup accepted by the courts, the resulting damages would be similarly inflated. This is not out of the question, as several cases in the past few years have included evidence obtained by law enforcement agents as to the number of bots under the control of the suspect(s). [29, 22, 23, 7] It is likely that some of these suspects, even if they admit to the numbers stated, may not know *precisely* how many hosts they truly did compromise and control.

One final interesting observation, which we have not seen noted in any other research to date, are the downward spikes in the bottom line (the reachable and responsive peers) of the

graphs in Figures 2 and 3. There is a larger downward spike occurring regularly on the second Wednesday of the month, and a slightly smaller downward spike on the second Thursday of the month. This behavior is regularly seen in each month, except September, 2007, and March, 2008. We believe that this is due to the effect of Windows Update patching and rebooting, following Microsoft's *Patch Tuesday* release schedule.

The reason the dip is seen on Wednesday, rather than Tuesday, is due to several factors. The patches are released in Redmond, Washington, on Tuesday mornings usually at 10:00AM local time. Redmond is in the United States Pacific time zone, or UTC +7 hours (or +8 hours, depending on Daylight Savings time.) There are millions of patches to be distributed. When automatic update is turned on, the default time to check for updates is 3:00AM local time, which would mean the first automatic check for patches would occur the following morning (i.e., Wednesday). It takes a while for hosts set to do automatic updating to begin to pick up the patches, or for users who manually retrieve patches to do so. Not all system clocks are precisely adjusted, so there is also some variation in when patches are retrieved due to clock drift.

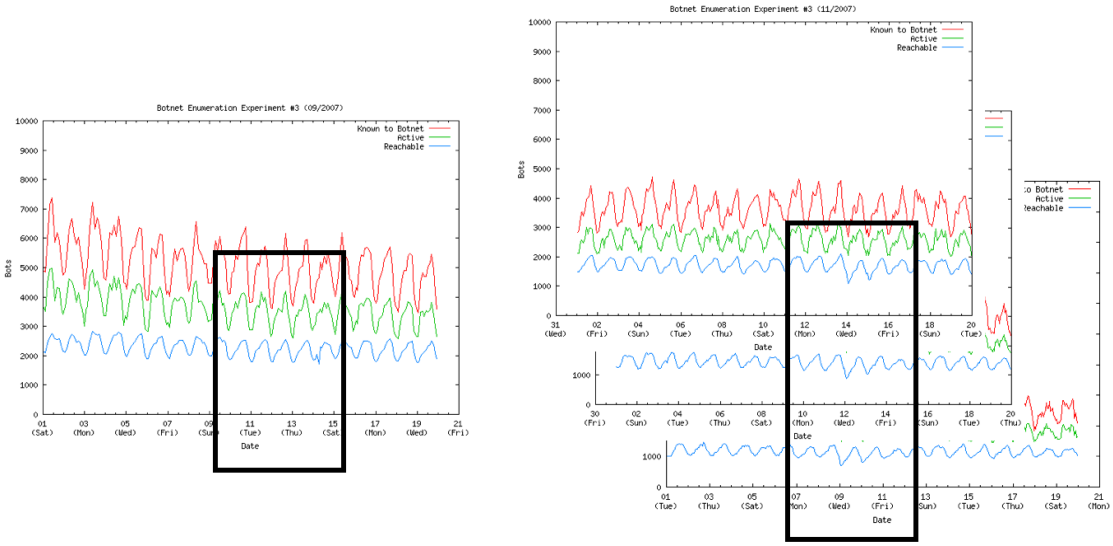


Figure 8: Effect of Microsoft *Patch Tuesday* on bot accessibility

The lack of spikes in September and March correlate with light Patch Tuesdays that did not require reboots for most systems: In September, 2007, only some Windows 2000 systems required a reboot after applying the patches, and no reboots were required in March, 2008, as only Microsoft Office and Microsoft Office Web Components were patched (not any operating system features that would require a reboot.)

It was reported that a recent Skype outage was due to the millions of rebooting PCs active in the Skype system, which was reported to last for two days. This matches our findings in the enumeration experiment, which show a significant percentage of the Nugache P2P network was unavailable at any given time over a two day period. [6]

[6] See `http://www.theregister.co.uk/2007/08/20/skype_outage_post-mortem/` for more on the Skype outage.

4 Conclusion

We have described the various types of malware networks and the respective approaches for discovering the hosts that are part of those malware networks, or botnets. We have had the unique opportunity to track Nugache, a P2P encrypted botnet which let us determine to some degree the number of adjacent nodes, with the qualitative aspect of *seen* (either as an initiator or receiver at some point) and *active* (recently had interactions with the current node). While this detail itself does not give us a fully accurate count by default, it has given us an adequate estimate of the botnet, as well as allowed for topology discovery for Nugache. One interesting form of behavior is the "dropping out" of the botnet during Microsoft Patch Tuesday, which to our knowledge had not been discussed before. For future work, we will adapt these discovery techniques for other types of P2P botnets, encrypted or not.

Acknowledgments

The authors would like to thank Brian Eckman, Matt Wilson, Joe Stewart, John Hernandez, Adam Turoff, Phil Groce, Michael Collins, Ross Kinder, Lawrence Baldwin, and others for their contributions and valuable comments.

References

[1] Les Cottrell and Connie Logg. Throughput Time Series Patterns (Diurnal and Step Functions), July 2004. `http://www.slac.stanford.edu/comp/net/pattern/diurnal.html`.

[2] David Dagon, Guofei Gu, Chris Lee, and Wenke Lee. A taxonomy of botnet structures. In *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07)*, December 2007.

[3] David Dagon, Cliff Zou, and Wenke Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[4] Dave Dittrich and Sven Dietrich. Command and control structures in malware: From Handler/Agent to P2P. In *USENIX ;login: vol. 32, no. 6*, December 2007. `http://www.usenix.org/publications/login/2007-12/pdfs/dittrich.pdf`.

[5] David Dittrich. *The DoS Project's trinoo distributed denial of service attack tool*. Available at `http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt`.

[6] Brandon Enright. Exposing storm worm. `http://noh.ucsd.edu/~bmenrigh/`, 2007.

[7] Joris Evers. 'Bot herders' may have controlled 1.5 million PCs, 2005. `http://news.com.com/Bot+herders+may+have+controlled+1.5+million+PCs/2100-7350_3-5906896.html`.

[8] Patrick Gray. Risky business #47 — botnet command and control meets web 2.0, June 2008. `http://www.itradio.com.au/security/?p=56`.

[9] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: overview and case study. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association. `http://portal.acm.org/ft_gateway.cfm?id=1323129&type=pdf&coll=&dl=&CFID=24169234&CFTOKEN=79754371`.

[10] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System, February 2006. `http://saikat.guha.cc/pub/iptps06-skype.pdf`.

[11] Thorsten Holz, Christian Gorecki, and Felix Freiling. Detection and mitigation of fast-flux service networks. In *Proceedings of NDSS 2008*, 2008.

[12] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm, April 2008. http://www.usenix.org/events/leet08/tech/full_papers/holz/holz_html/.

[13] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst W Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm wor. In *LEET'08: First USENIX Workshop on Large-Scale Exploits and Emergent Threats, April 15, 2008, San Francisco, CA, USA*, Apr 2008.

[14] The Honeynet Project. Fast-Flux Service Networks, 2007. http://www.honeynet.org/papers/ff/fast-flux.html.

[15] Brian Krebs. Just How Bad is the Storm Worm?, October 2007. http://blog.washingtonpost.com/securityfix/2007/10/the_storm_worm_maelstrom_or_te.html.

[16] Brian Krebs. Kraken spawns a clash of the titans, April 2008. http://blog.washingtonpost.com/securityfix/2008/04/kraken_creates_a_clash_of_the.html.

[17] Nikolaus Kühn and Susanne Jucknath. Visualizing large and dynamic communications networks, April 2003. http://swt.cs.tu-berlin.de/lehre/vsp/Seminar/NetworkVisualization_600dpi.pdf.

[18] Jelena Mirković, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, 2004.

[19] Jose Nazario. BlackEnergy DDoS Bot Analysis, October 2007. http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf.

[20] Jose Nazario. Botnet Tracking: Tools, Techniques, and Lessons Learned, July 2007. http://www.blackhat.com/presentations/bh-dc-07/Nazario/Presentation/bh-dc-07-Nazario.pdf.

[21] Arbor Networks. Hitpop ddos malware analysis. http://atlas-public.ec2.arbor.net/docs/Hitpop_DDoS_Malware_Analysis_PUBLIC.pdf, May 2008.

[22] Department of Justice. Criminal Complaint: United States of America v. Paul G. Ashley, Jonathan David Hall, Joshua James Schichtel, Richard Roby and Lee Graham Walker, 2004. http://www.reverse.net/operationcyberslam.pdf.

[23] Department of Justice. Over One Million Potential Victims of Botnet Cyber Crime, 2007. http://www.ic3.gov/media/initiatives/BotRoast.pdf.

[24] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. Fluxor: detecting and monitoring fast-flux service networks. In *Proceedings of the 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, DIMVA, Paris, France*, Lecture Notes in Computer Science. Springer, July 2008. To appear.

[25] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My Botnet Is Bigger Than Yours (Maybe, Better Than Yours): Why Size Estimates Remain Challenging. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, April 2007. http://www.usenix.org/events/hotbots07/tech/full_papers/rajab/rajab_html/.

[26] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–51, New York, NY, USA, 2006. ACM.

[27] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. In *USENIX ;login: vol. 32, no. 6*, December 2007. http://www.usenix.org/publications/login/2007-12/pdfs/stover.pdf.

[28] Vanja Svajcer. Mayday botnet bigger than dorf/storm?, February 2008. http://www.sophos.com/security/blog/2008/02/1057.html.

[29] United States Department of Justice. U.S. v. James Jeanson Ancheta. http://news.findlaw.com/hdocs/docs/cyberlaw/usanchetaind.pdf.