Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm

Thorsten Holz[*] Moritz Steiner[*†] Frederic Dahl[*] Ernst Biersack[†] Felix Freiling[*]

[*] University of Mannheim
{holz,dahl,freiling}@informatik.uni-mannheim.de

[†] Institut Eurécom, Sophia Antipolis
{steiner,biersack}@eurecom.fr

Abstract

Botnets, i.e., networks of compromised machines under a common control infrastructure, are commonly controlled by an attacker with the help of a central server: all compromised machines connect to the central server and wait for commands.

However, the first botnets that use peer-to-peer (P2P) networks for remote control of the compromised machines appeared in the wild recently. In this paper, we introduce a methodology to analyze and mitigate P2P botnets. In a case study, we examine in detail the Storm Worm botnet, the most wide-spread P2P botnet currently propagating in the wild. We were able to infiltrate and analyze in-depth the botnet, which allows us to estimate the total number of compromised machines. Furthermore, we present two different ways to disrupt the communication channel between controller and compromised machines in order to mitigate the botnet and evaluate the effectiveness of these mechanisms.

1 Introduction

A *bot* is a computer program installed on a compromised machine which offers an attacker a remote control mechanism. Botnets, i.e., networks of such bots under a common control infrastructure, pose a severe threat to today's Internet: Botnets are commonly used for Distributed Denial-of-Service (DDoS) attacks, sending of spam, or other nefarious purposes [5, 24, 15].

The common control infrastructure of botnets in the past was based on Internet Relay Chat (IRC): The attacker sets up an IRC server and opens a specific channel in which he posts his commands. Bots connect to this channel and act upon the commands they observe. Today, the standard technique to mitigate IRC-based botnets is called *botnet tracking* [11, 15, 14] and includes three steps. The first step consists of acquiring and analyzing a copy of a bot. This can be achieved for example using honeypots [1] and special analysis software [4, 32]. In the second step, the botnet is infiltrated by connecting to the IRC channel with a specially crafted IRC client. Using the collected information, it is possible to analyze the means and techniques used within the botnet. More specifically, it is possible to identify the central IRC server which, in the third and final step, can be taken offline by law enforcement or other means [9]. An attacker can also use an HTTP server for distributing commands: in this setup, the bots periodically poll this server for new commands and act upon them. The botnet tracking methodology outlined above can also be applied in this scenario.

Today we are encountering a new generation of botnets that use P2P style communication. These botnets do not have a central server that distributes commands and are therefore not directly affected by botnet tracking. Probably the most prominent P2P bot currently spreading in the wild is known as *Peacomm*, *Nuwar*, or *Zhelatin*. Because of its devastating success, this worm received major press coverage [13, 17, 22] in which — due to the circumstances of its spreading — it was given the name *Storm Worm* (or *Storm* for short) [30]. This malware is currently the most wide-spread P2P bot observed in the wild.

In this paper we study the question, whether the technique of botnet tracking can be extended to analyze and mitigate P2P based botnets. Roughly speaking, we adapt the three steps of botnet tracking in the following way using Storm Worm as a case study: In the first step, we must get hold of a copy of the bot binary. In the case of this botnet, we use spam traps to collect Storm Worm generated spam and client side honeypots to simulate the infection process. The second step, the infiltration of the botnet, is adopted since we need to use a P2P protocol instead of IRC, HTTP, or other client/server protocols. The third step, the actual mitigation, is the most difficult: In the case of Storm Worm we exploit weaknesses in the protocol used by the bot to inject our own content into the botnet, in an effort to disrupt the communication between the bots. We argue later that this method is effective against P2P botnets using content-based publish/subscribe-style communication.

Our measurements show that our strategy can be used as a way to disable the communication within the Storm botnet to a large extent. As a side effect, we are able to estimate the size of the Storm botnet, in general a hard task [25]. Our measurements are much more precise than previous measurements [12, 17]. This is because measurements previously were based on *passive* techniques, e.g., by observing visible network events like the number of spam mails supposedly sent via the bots. We are the first to introduce an *active* measurement technique to actually enumerate the number of infected machines: We crawl the P2P network, keep track of all peers, and distinguish an infected peer from a regular one based on characteristic behavior of the bots.

To summarize, the contributions of this paper are threefold:

1. We extend the method of botnet tracking [11] to P2P based botnets. We argue that the method is applicable to analyze and mitigate *any* botnet using P2P publish/subscribe-style communication.
2. We demonstrate the applicability by performing a case study of Storm Worm, thereby being the first to develop ways to mitigate Storm Worm.
3. In doing this, we present the first empirical study of P2P botnets giving details about their propagation phase, their malicious activities, and other features.

2 Botnet Tracking adapted to P2P Botnets

We now present a general method to analyze and mitigate specific P2P botnets.

2.1 Class of Botnets Considered

The class of botnets we consider are those which use unauthenticated content-based publish/subscribe style communication. This communication paradigm is popular in many of the well-known file sharing systems like Gnutella, eMule, or BitTorrent. The characteristics of such systems are:

- Peer-to-peer network architecture: These networks have in common that all network nodes are both clients and servers: Any node can provide and retrieve information at the same time. This feature makes P2P networks extremely robust against node failures, i.e., they provide *high resilience*.
- Content-based publish/subscribe-style communication: In such systems the network nodes do not directly *send* information to each other. Instead, an information provider *publishes* a piece of information i, e.g., a file, using an identifier which is derived solely from i. An information consumer can then *subscribe* to certain information using a filter on such identifiers. In practice, such identifiers can be derived from specific content of i or simply computed using a hash function. The P2P system matches published information items to subscriptions and delivers the requested information to the consumer.
- Unauthenticated communication: Content providers do not authenticate information, but authentication is usually implicit: If the information received by a peer matches its subscription, then it is assumed to be correct. None of the popular file sharing systems does provide authentication.

Note that in such systems communication is very *loosely coupled*. Neither do information consumers in general know, which node published the information they receive, nor does an information provider know, which nodes will receive their published information. Both points, loose coupling and high resilience, make these networks attractive technologies for running botnets.

2.2 Botnet Tracking Extended

We now introduce a widely applicable method to analyze and mitigate any member of the class of botnets described above. We generalize the botnet tracking method introduced for botnets with a central server to botnets that use P2P networks and exemplify the method in Section 4 with the help of a case study on Storm Worm.

Step 1: Exploiting the P2P Bootstrapping Process. A bot spreading in the wild must contain information to bootstrap itself within the botnet. In the case of P2P botnets, the bot *must contain* sufficient information on how to *connect* to the botnet and how to *receive* commands from the attacker. Usually this information includes a number of IP addresses of initial peers, service ports and application-specific connection information. By getting hold of and analyzing a bot, it is possible to extract this information by either active or passive means.

Getting hold of a bot means to simulate the infection process, which is a technique known from the area of honeypot technology. The main difficulties here are (1) to find out the infection vector and (2) to simulate vulnerable applications. While (1) may take some time and is hard to automate, (2) can be efficiently automated, e.g., using sandbox or network analysis techniques. The result of this step is a list of network locations (IP address / port) of peer services that form part of the P2P botnet.

Step 2: Infiltration and Analysis. As a result of step 1, we also retrieve connection information to actually *join* the botnet. Joining the botnet means to be able to receive botnet commands issued by the attacker. By crafting a specific P2P client, infiltration of the botnet remains a dangerous, but technically manageable process. It can be dangerous since the attacker could notice the infiltration process and start to specifically attack us.

Step 3: Mitigation. The mitigation of botnets must attack the control infrastructure to be effective, i.e., either the servers or the communication method. We now argue that publish/subscribe-style communication has weaknesses which can be generally exploited. In a botnet, the attacker wishes to in some way send commands to the bots. This is the characteristic of remote control. However, in publish/subscribe systems, there is no way to send information directly. Instead, a broadcast is simulated, as we now explain. The attacker defines a set $C = \{c_1, c_2, \ldots\}$ of botnet commands. At any point in time, whenever he wishes to send a command c_i to the bots he publishes c_i in the P2P system. The bots must be able to receive all the commands from the attacker so they subscribe to the entire set C and can then accept commands.

Note that since we consider *unauthenticated* publish/subscribe systems, any member of the P2P system can publish c_i. This is the idea of our mitigation strategy: Using the client from step 2, we can now try to either inject commands into the botnet or disrupt the communication channel. In general, disruption is possible: We can flood the network with publication requests and thus "overwrite" publications by the attacker. In order to actually inject commands, we need to understand the communication process in detail and then publish a specially crafted c_i.

3 Inside Storm Worm

Before exemplifying our methodology of tracking P2P botnets, we provide an overview of Storm Worm. Please note that this description is a summary of the behavior we observed when monitoring the Storm botnet for a period of several months. The attackers behind this network quite frequently change their tactics and move to new attack vectors, change the communication protocol, or change their behavior in other ways. The results from this section describe several important aspects of Storm and we try to generalize our findings as much as possible. Together with the technical report by Porras et al. [23], this is currently the most complete overview of Storm Worm.

3.1 Propagation Mechanism

A common mechanism for autonomous spreading malware to propagate further is to exploit remote code execution vulnerabilities in network services. If the exploit is successful, the malware transfers a copy of itself to the victim's machine and executes this copy in order to propagate from one machine to another. This propagation mechanism is used for example by CodeRed [21], Slammer [20], and all common IRC bots [3]. Storm Worm, however, propagates solely by using e-mail, sim-

ilar to mail worms like Loveletter/ILOVEYOU and Bagle. The e-mail body contains a varying English text that tries to trick the recipient into either opening an attachment or clicking on an embedded link. The text uses social engineering techniques in order to pretend to be a legitimate e-mail, e.g., we found many e-mails related to Storm that feign to be a greeting card.

With the help of *spamtraps*, i.e., e-mail addresses not used for communication but to lure spam e-mails, we can analyze the different spam campaigns used by Storm for propagation. We have access to a spamtrap archive between September 2006 and September 2007 which receives between 2,200 and 23,900 spam messages per day (8,500 on average). The first Storm-related message we are aware of was received on December 29, 2006: It contained best wishes for the year 2007 and as an attachment a copy of the Storm binary. An analysis of this archive shows that Storm is quite active and can generate a significant amount of spam: we found that the botnet was in some period responsible for more than 10% of all spam received in the spamtraps.

The attackers behind Storm change the social engineering theme quite often and adopt to news or events of public interest. For example, the name "Storm Worm" itself relates to the subject used in propagation mails during January 2007 which references the storm Kyrill, a major windstorm in Europe at that time. For events of public interest (e.g., Labor Day, start of NFL season, or public holidays), the attackers use a specific social engineering scam. Furthermore, they also use general themes (e.g., privacy concerns or free games) to trick users into opening the link in the e-mail message. In total, we counted more than 21 different e-mail campaigns for the period between December 2006 and January 2007.

To study the next step in the propagation phase, we examined the links from Storm-related e-mails with the help of *client honeypots*. A client honeypot is a system designed to study attacks against client applications, in our case attacks against a webbrowser [31]. We implemented our own client honeypot which can be used to analyze a given website with different kinds of browsers on top of CWSandbox [32]. Based on this system, we can determine whether or not the visited site compromised our honeypot. During five of the different spam campaigns we examined several URLs referenced in the e-mails. We used different releases of three webbrowsers, resulting in a total of eight different browser versions. The results indicate that Storm exploits only webbrowsers with a specific `User-Agent`, a HTTP request header field specifying the browser version. If this header field specifies a non-vulnerable browser, the malicious server does not send the exploit to the client. However, if the client seems to be vulnerable, the server sends between three and six different exploits for vulnerabilities commonly found in this browser or in common browser-addons. The goal of all these exploits is to install a copy of the Storm binary on the visitor's machine. We observed that the actual exploit used in the malicious websites is polymorphic, i.e., the exploit code changes periodically, in this case every minute, which complicates signature-based detection of these malicious sites.

If the malicious website successfully compromises the visitor's webbrowser or the visitor falls for the social engineering scam and intentionally installs the binary, the victim is infected. The binary itself also shows signs of polymorphism: When continuously downloading the same binary from the same webserver, the size (and accordingly the MD5 checksum) changes every minute. An analysis revealed that the changes are caused by periodically re-packing the binary with an executable packer which is responsible for the change in size.

3.2 System-level Behavior

Storm Worm itself is a sophisticated malware binary and uses several advanced techniques, e.g., the binary packer is one of the most advanced seen in the wild [10], the malware uses a rootkit in order to hide its presence on the infected machine, and it has a kernel-level component in order to remain undetected on the system. We do not provide a complete overview of the system-level behavior due to space limitations and since some of this information is already available [23, 30].

We only mention two aspects that are important to understand the network-level behavior, which is a key part in understanding how to infiltrate and mitigate Storm. First, during the installation process, the malware also stores a configuration file on the infected system. This file contains in an encoded form information about other peers with which the binaries communicates after the installation phase. Each peer is identified via a hash value and an IP address/port combination. This is the basic information needed to join the P2P network, for which we provide details in the next section. Second, Storm synchronizes the system time of the infected machine with the help of the Network Time Protocol (NTP). This means that each infected machine has an accurate clock. In the next section, we show how this synchronization is used by Storm for communication purposes.

3.3 Network-Level Behavior

For finding other bots within the P2P network and receiving commands from its controller, the first version of Storm Worm uses OVERNET, a Kademlia-based [19] P2P distributed hash table (DHT) routing protocol. OVERNET is implemented by Edonkey2000, that was officially shut down in early 2006, but still benign peers are online in this network, i.e., not *all* peers within OVERNET are bots per se.

In October 2007, the Storm botnet changed the communication protocol slightly. From then on, Storm does not only use OVERNET for communication, but newer versions use their own P2P network, which we choose to call the *Stormnet*. This P2P network is identical to OVERNET except for the fact that each message is XOR encrypted with a 40 byte long key. Therefore, the message types enumerated below remain the same, only the encoding changed. All algorithms introduced in this paper and the general methodology are not affected by this change in communication since the underlying weakness – the use of unauthenticated content-based publish/subscribe style communication – is still present. Note that in Stormnet we do not need do distinguish between bots and benign peers, since only bots participate in this network.

In the following, we describe the network-level communication of Storm and how it uses OVERNET to find other infected peers. As in other DHTs, each OVERNET or Stormnet node has a global identifier, referred to as DHT ID, which is a randomly generated 128 bit ID. When the client application starts for the first time, it generates the DHT ID and stores it. Storm Worm implements the same mechanism and also generates an identifier upon the first startup.

Routing Lookup. Routing in OVERNET and Stormnet is based on prefix matching: A node a forwards a query destined to a node d to the node in its routing table that has the smallest XOR-distance with d. The XOR-distance $d(a, b)$ between nodes a and b is $d(a, b) = a \oplus b$. It is calculated bitwise on the DHT IDs of the two nodes, e.g., the distance between $a = 1011$ and $b = 0111$ is $d(a, b) = 1011 \oplus 0111 = 1100$. The entries in the routing tables are called *contacts* and are organized as an unbalanced *routing tree*. Each contact consists of the node's DHT ID, IP address, and UDP port. A peer a stores only a few contacts to peers that are far away in the DHT ID space (on the left side of the tree) and increasingly more contacts to peers closer in the DHT ID space (on the right side of the tree).

Routing to a given DHT ID is done in an *iterative way*. P sends `route requests` to three peers (to improve robustness against node churn), which may or may not return to P `route responses` containing new peers even closer to the DHT ID, which are queried by P in the next step. The routing lookup terminates when the returned peers are further away from the DHT ID than the peer returning them.

Publishing and Searching. A *key* in a P2P system is an identifier used to retrieve information. In many P2P systems, a key is typically published on a single peer that is closest to that key according to the XOR metric. In OVERNET, to deal with node churn, a key is published on twenty different peers. Note that the key is not necessarily published on the peers *closest* to the key. To assure persistence of the information stored, the owner periodically republishes the information.

As for the publishing process, the search procedure uses the routing lookup to find the peer(s) closest to the key searched for. The four most important message types for the publish and search process are first `hello`, to check if the other peer is still alive and to inform the other peer about one's existence and the IP address and DHT ID. Second, `route request/response(kid)`, to find peers that are closer to the DHT ID `kid`. Third, `publish request/response`, to publish information. And fourth, `search request/response(key)`, to search for information whose hash is `key`.

The basic idea of the Storm communication is that an infected machine searches for specific keys within the network. The controller knows in advance which keys are searched for by the infected machines and thus he publishes commands at these keys. These keys can be seen as *rendezvous points* or *mailboxes* the controller and infected machines agree on. In the following, we describe this mechanism in more detail.

Storm Worm Communication. In order to find other Storm-infected machines within the OVERNET network, the bot searches for specific keys using the procedure outlined above. This step is necessary since the bot needs to distinguish between regular and infected peers within the network. The key is generated by a function $f(d, r)$ that takes as input the current day d and a random number r between 0 and 31, thus there can be 32 different keys each day. We found this information in two different ways: First, we reverse engineered the bot binary and identified the function that computes the key. The drawback of this approach is that the attacker can easily change f and then we need to analyze the binary again, thus we are always one step behind and have to react once the attacker change his tactics.

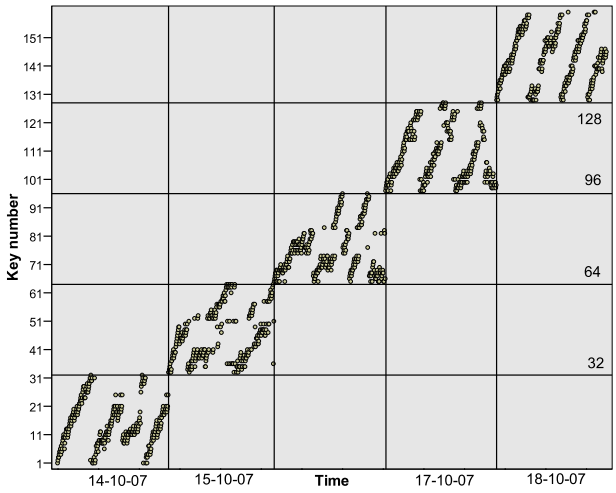


Figure 1: Keys generated by Storm in order to find other infected peers within the network (October 14-18, 2007)

The second way to retrieve this information is by treating the bot as a *black box* and repeatedly force it to re-connect to the network. This is achieved by executing the bot within a honeynet, i.e., a highly controlled environment. The basic idea is to execute the binary on a normal Windows machine, set up a modified firewall in front of this machine to mitigate risk involved, and capture all network traffic. Since the bot can hardly identify that it runs within a strictly monitored environment, it behaves normal, connects to the P2P network, and then starts to search for keys in order to find other infected peers and the commands from the controller. We monitor the communication and extract from the network stream the key the bot searches for. Once we have captured the search key, we revert the honeypot to a clean state and repeat these steps. Since the bot can not keep any state, it generates again a key and starts searching for it. By repeating this process over and over again, we are able to enumerate the keys used by Storm in a black-box manner, without actually knowing the function f used by the binary.

Figure 1 shows the keys found during a period of five days. We see a clear pattern: On each day, there are 32 unique keys which are generated depending on the time, and for different days there is no overlap in the search keys. This result confirms the results of our reverse engineering approach. The keys are important to actually identify Storm-infected machines and we can also uses them for mitigation purposes. Another important implication is that we can pre-compute the search keys in advance: On day d, we can set the system time to $d + n$ and perform our black-box enumeration process as outlined above. As a result, we collect all keys the bot will search on day $d + n$.

If the attackers change the function that generates the key, e.g., by using other inputs for f, we can still determine which keys are *currently* relevant for the communication within the botnet with the help of our honeypot setup: By analyzing the network communication, we can obtain the current search key relevant for the communication. In general, we can use this setup to learn the keys a bot searches for in a black-box manner, regardless of the actual computation.

The keys are used by the bot to find the commands which should be executed: The attacker has in advance published con-

tent at these keys since he knows which keys are searched for by an infected peer. The keys are similar to rendezvous point which both the controller and the bot know. In DHT-based P2P networks, this is a viable communication mechanism. The actual content published in OVERNET at these keys contains a filename of the pattern "`*.mpg;size=*;`" [23]. No other meta tags (like file size, file type, or codec) are used and the asterisks depict 16-bit numbers. Our observations indicate that the bot computes an IP address and TCP port combination based on these two numbers and then contacts this *control node*. However, up to now we do not know how to compute the IP address and port out of the published numbers. Only bots participate in Stormnet, thus they do not need to authenticate themselves. Publications in Stormnet do not contain any meta tags. The IP address and port of the machine that send the publish request seem to be the actual information.

All following communication just takes place between the bot and the control node, which sends commands to the bot. This is similar to a two-tier architecture where the first-tier is contained within OVERNET or Stormnet and used to find the second-tier computers that send the actual commands. Once the Storm infected machine has finished the TCP handshake with the control node, this node sends a four byte long challenge c in order to have a weak authentication scheme. The bot knows the secret "key" k = `0x3ED9F146` and computes the response r via $r = c \oplus k$. This response is then sent to the control node and the bot is successfully authenticated. All following communication is encoded using *zlib*, a software library for data compression.

The infected machine receives via this communication channel further commands that it then executes. Up to now, we only observed that infected machines are used to either send spam e-mails or to start DDoS attacks. In order to send spam, the infected machines receive a spam template and a list of e-mail addresses to be spammed. We found two different types of mails being sent by Storm: propagation mails that contain different kinds of social engineering campaigns as introduced in Section 3.1 or general spam messages that advertize for example pharmaceutical products or stocks. The attackers behind Storm presumably either earn money via renting the botnet to spammers, sending spam on behalf of spammers, or running their own pharmacy shop. The DDoS attacks we observed were either SYN or ICMP flooding attacks.

4 Case Study: Tracking Storm Worm

After an overview of the behavior of Storm Worm, we now present a case study of how to apply the extended botnet tracking methodology outlined in Section 2 for this particular bot. We show that we can successfully infiltrate and analyze the botnet, even though there is no central server like in traditional botnets. Furthermore, we also outline possible attacks to mitigate Storm and present our measurement results.

4.1 Exploiting the P2P Bootstrapping Process

At the beginning, we need to capture a sample of the bot. As outlined in Section 3.1, we can use spamtraps to collect spam mails and then client honeypots to visit the URLs and obtain a binary copy of the malware. Based on this copy of Storm Worm, we can obtain the current peer list used by the binary via an automated analysis (see Section 3.2).

In the first step, we also use the honeynet setup introduced in Section 3.3. With the help of the black-box analysis, we are able to observe the keys that Storm Worm searches for. As explained before, the controller can not send commands directly to the bot, thus the bot needs to search for commands and we exploit this property of Storm to obtain the search keys. During this step we thus obtain (at least a subset of) the current search keys, which allows us to infiltrate and analyze the Storm botnet. With a single honeypot, we were able to reliably acquire all 32 search keys each day for a given Storm binary.

4.2 Infiltration and Analysis

Based on the obtained keys and knowledge of the communication protocol used by Storm, we can start with the infiltration and analysis step to learn more about the botnet, e.g., we can enumerate the size of the network. First, we introduce our method to learn more about the peers in OVERNET and Stormnet and about the content announced and searched for in these networks. Afterwards we present several measurement results.

4.2.1 Crawling the P2P Network

To measure the number of peers within the whole P2P network, we have developed our own crawler for OVERNET and *Stormnet*. It uses a principle similar to the KAD crawler we developed [29]. Our crawler runs on a single machine and uses a breadth first search issuing `route requests` to find the peers currently participating in OVERNET or Stormnet. The speed of our crawler allows us to discover all peers within 20 to 40 seconds (depending on the time of day).

The crawler runs two asynchronous threads: one to send the `route requests` (Algorithm 1) and one to receive and parse the `route responses` (Algorithm 2). One list containing the peers discovered so far is maintained and used by both threads. The receiving thread adds the peers extracted from the `route responses` to the list, whereas the sending thread iterates over the list and sends 16 `route requests` to every peer. The DHT ID asked for in the `route requests` are calculated in such a way that each of them falls in different zones of the peer's routing tree. This is done in order to minimize the overlap between the sets of peers returned.

4.2.2 Spying in OVERNET and Stormnet

The main idea of the *Sybil* attack [7] is to introduce malicious peers, the *sybils*, which are all controlled by one entity. Positioned in a strategic way, the *sybils* allow us to gain control over a fraction of the P2P network or even over the whole network. The *sybils* can monitor the traffic, i.e. act as spies (behavior of the other peers) or abuse the protocol in other ways. For example, `route requests` may be forwarded to the wrong endhosts or rerouted to other *sybil* peers. We use the Sybil attack to infiltrate OVERNET and the Stormnet and observe the communication to get a better understanding of it.

Assume that we want to find out in the least intrusive way what type of content is published and searched for in the one of both networks. For this, we need to introduce *sybils* and make them known, such that their presence is reflected in the routing tables of the non-sybil peers. We have developed a light-weight implementation of such a "spy" that is able to create thousands of *sybils* on one single physical machine. We achieve this scal-

Algorithm 1: send thread (is executed once per crawl)

```
Data: peer: struct{IP address, port number, DHT ID}
Data: shared list Peers = list of peer elements
/* the list of peers filled by the receive thread and worked on by the send thread */
Data: int position = 0
/* the position in the list up to which the peers have already been queried */
Data: list ids = list of 16 properly chosen DHT ID elements
1 Peers.add(seed); /* initialize the list with the seed peer */
2 while position < size(Peers) do
3     for i=1 to 16 do
4         dest DHT ID = Peers[position].DHT ID ⊕ ids[i]; /* normalize bucket to peer's position */
5         send route requests(dest DHT ID) to Peers[position];
6     position++;
```

Algorithm 2: receive thread (waits for the `route response` messages)

```
Data: message mess = route response message
Data: peer: struct{IP address, port number, DHT ID}
Data: shared list Peers = list of peer elements
/* the list shared with the send thread */
1 while true do
2     wait for (mess = route response) message; foreach peer ∈ mess do
3         if peer ∉ Peers then
4             Peers.add(peer);
```

ability since the *sybils* do not keep any state about the interactions with the non-sybil peers [28]. We introduce 2^{24} *sybils* into OVERNET and Stormnet: the first 24 bits are different for each *sybil* and the following bits are fixed, they are the *signature* of our *sybils*. The spy is implemented in the following steps:

1. Crawl the DHT ID space using our crawler to learn about the set of peers $\mathcal{P}$ currently online.
2. Send `hello requests` to the peers $\mathcal{P}$ in order to "poison" their routing tables with entries that point to our *sybils*The peers that receive a `hello request` will add the *sybil* to their routing table.
3. When a `route request` initiated by non-sybil peer P reaches a *sybil*, that request will be answered with a set of *sybils* whose DHT IDs are closer to the target. This way, P has the impression of approaching the target. Once P is "close enough" to the target DHT ID, it will initiate a `publish request` or `search request` also destined to one of our *sybil* peers. Therefore, for any `route request` that reaches one of our *sybil* peers, we can be sure that the follow-up `publish request` or `search request` will also end-up on the same *sybil*.
4. Store the content of all the requests received in a database for later evaluation.

Using the Sybil attack, we can now monitor requests within the whole network.

4.2.3 Results for Crawling and Spying

Other Studies related to Storm Worm. Concurrent to our work, Storm Worm has become the subject of intense studies at many places around the world [8, 2]. By looking at the (DHT ID, IP address) pairs collected by our crawler, we found several instances where DHT IDs that contain a well chosen pattern covered the whole DHT ID space and the IP addresses map all to the same institution. We could observe experiments (or worm activities) going on in San Diego (UCSD), Atlanta (Georgia Tech) and many other places (also on address spaces we could not resolve). We filtered out all these (DHT ID, IP address) pairs before doing our analysis.

Storm bots in OVERNET. During full crawls from October 2007 till beginning of February 2008, we found between 45,000 and 80,000 concurrent online peers in OVERNET. We define a peer as the combination of an IP address, a port number and a DHT ID. In the remaining part, we use the term "IP address" for simplicity. If one DHT ID is used on several IP addresses, we filter these instances out. We also filter out IP addresses that run more than one DHT ID simultaneously.

Note that the same machine participating in OVERNET can, on the one hand, change its IP address over time. This fact is known as *IP address aliasing*. On the other hand, it can also change its DHT ID over time. This is known as DHT ID *aliasing*. Due to this reason, simply counting the number of different DHT IDs or IP addresses provides only a rough estimate of the total number of machines participating in OVERNET. Nevertheless, we present for the sake of completeness the total number of DHT IDs and IP addresses observed during the month of October 2007: With our crawler, we could observe 426,511 different DHT IDs on 1,777,886 different IP addresses. These numbers are an upper bound for the number of Storm-infected machines: Since we enumerate the whole DHT ID space, we find all online peers, from which a subset is infected with Storm.

About 75% of all peers are not located behind NATs or firewalls and can be *directly contacted* by our crawler. We used the

MaxMind database [18] to map the IP addresses to countries. We saw clients from 210 countries. These split up in 19.8% that can not be resolved, 12.4% from the US, 9.4% from Uruguay, 6% from Germany etc.

Lower Bound for Storm-infected Machines in OVERNET. When spying on OVERNET, the benign peers can be distinguished from the bots of the Storm botnet: Bots publish files with characteristic filenames and no other meta tags (see Section 3.3 for details). However, not every bot does make such announcements. This allows us to obtain a lower bound of the size of the botnet since only peers with this characteristic pattern are definitely infected with Storm. Note that only the Storm bots with a public IP address publish content in OVERNET.

Every day we see between 5,000 and 6,000 distinct peers that publish Storm related content. About the same number of peers publish real, e.g., non-Storm related, content. Around 30,000 peers per day did perform searches.

Most of the clients that published Storm content (the bots running on public IP addresses) come from the US (31%) followed by India (5.3%) and Russia (5.2%). Note, however, that 21% of the IP addresses could not be mapped to any country. In total, we observed bots from over 133 countries. Due to the fact that all social engineering campaigns we observed contain English text, it is not surprising that the majority of Storm-infected machines are located in the US.

Estimating the Number of Storm-infected Machines in OVERNET. All we can measure with confidence are upper and lower bounds of the number of concurrently active bots in OVERNET. The lower bound being around 5,000 – 6,000 and the upper bound being around 45,000 – 80,000 distinct bots.

Storm content was published using roughly 1,000 different keys per day. This indicates that there are many different version of Storm in the wild, since each binary only searches for 32 keys per day. Some of these keys were used in more than 1,500 publications requests, whereas the majority of keys was used in only few publications. During the observation period in October 2007, we observed a total of 13,307 keyword hashes and 179,451 different file hashes. We found that 750,451 non-Storm related files were announced on 139,587 different keywords.

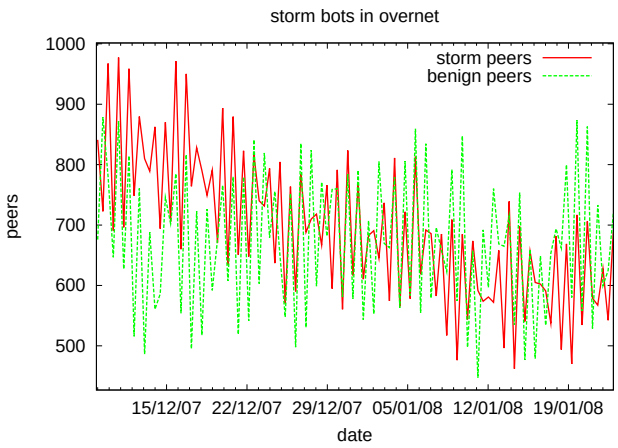


Figure 2: The number of bots and benign peers that published content in OVERNET.

From the end of the year 2007 on, the number of storm bots using OVERNET and the Storm activity in OVERNET decreased. Figure 2 shows the number of bots and benign peers that published content in OVERNET in December 2007 and January 2008. The number of benign peers remains constant, while the number of storm bots decreases. We think this is due to the fact that the whole botnet now shifts to Stormnet.

Size Estimation for Stormnet. We can apply the algorithms outlined above to enumerate all peers within Stormnet. The important difference between Stormnet and OVERNET is the fact that OVERNET is used by regular clients and Storm bots, whereas Stormnet is used only by machines infected with Storm Worm. Hence, we do not need to differentiate between benign and infected peers.

We crawled Stormnet every 30 minutes since beginning of December 2007 till beginning of February 2008. During this period, we saw between 5,000 and 40,000 peers concurrently online. There was a sharp increase in the number of storm bots at the end of 2007 due to a propagation wave during Christmas and New Years Eve (Figure 3). After that increase, the number of bots varied between 25,000 and 40,000 before stabilizing in the beginning of January 2008 around 15,000 to 30,000. In total, we found bots in more than 200 different countries. The biggest fraction comes from the US (23%). As seen in the figure, Storm Worm also exhibits strong diurnal patterns like other botnets [6].

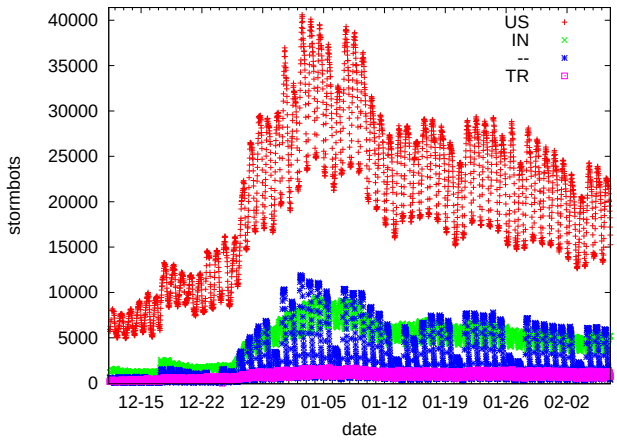


Figure 3: Number of bots in Stormnet, split by geolocation.

Figure 4 depicts the number of distinct IP addresses as well as the number of distinct "rendez-vous" hashes searched for in Stormnet. At the end of 2007, the number of peers searching in Stormnet increased significantly and the search activity stabilized at high level in the middle of January 2008. Similar to the diurnal pattern of the number of storm bots, also the search activity within Stormnet shows a distinct diurnal pattern.

The publish activity shows exactly the same behavior over time compared to the search activity (Figure 5). However, the number of "rendez-vous" hashes that are searched for is nearly of an order of magnitude higher than the number of hashes that are published. For the number of distinct IP addresses, especially in the week from 29/12/2007 to 05/01/2008 and starting again on 19/01/2008, the distinct number of IP addresses launching search queries are two orders of magnitudes higher than the the number of IP addresses publishing. The number of IP addresses searching for content is around three times bigger than

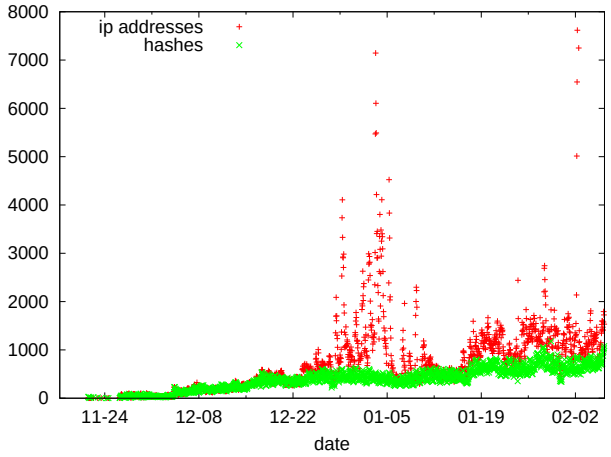


Figure 4: Search activity in Stormnet.

the number of IP addresses publishing content. It is somehow intuitive that the number of IP addresses that search is bigger than those publishing, since the goal is to propagate information.

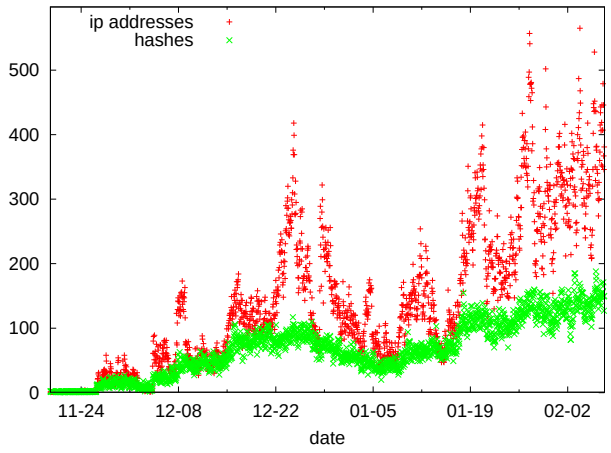


Figure 5: Publish activity (distinct IP addresses and rendez-vous hashes) in Stormnet.

4.3 Mitigation

Based on the information collected during the infiltration and analysis phase, we can also try to actually mitigate the botnet. In this section, we present two theoretical approaches that could be used to mitigate Storm Worm and our empirical measurement results for both of them.

4.3.1 Eclipsing Content

A special form of the *sybil* attack is the *eclipse* attack [26] that aims to separate a part of the P2P network from the rest. The way we perform an eclipse attack resembles very much that of the *sybil* attack described above, except that the DHT ID space covered is much smaller.

To eclipse a particular keyword K, we position a certain number of *sybils* closely around K, i.e., the DHT IDs of the *sybils* are closer to the hash value of K than the DHT IDs of any real peer. We then need to announce these *sybils* to the regular peers in order to "poison" the regular peers' routing tables and to attract all the `route requests` for keyword K. Unfortunately, using this technique we could – in contrast to similar experiments in KAD [27] – not completely eclipse a particular keyword. This is due to the fact that in OVERNET and Stormnet the content is spread through the entire hash space and not restricted to a zone around the keyword K. As a consequence, in OVERNET and Stormnet, the eclipse attack can thus not be used to mitigate the Storm Worm network.

4.3.2 Polluting

Since eclipsing content is not feasible in OVERNET or Stormnet, we investigated another way to control particular content. To prevent peers from retrieving search results for a certain key K, we publish a very large number of files using K. The goal of the pollution attack is to "overwrite" the content previously published under key K. Since the Storm bots continue to publish their content as well, this is a race between the group performing mitigation attempts and the infected machines.

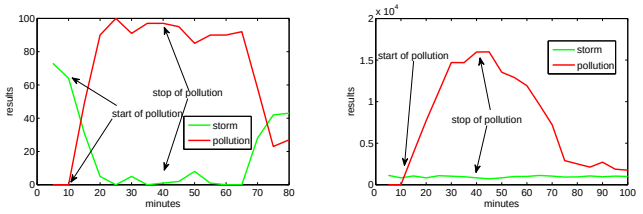
To perform this attack, we again first crawl the network, and then publish files to all those peers having at least the first 4 bits in common with K. This crawling and publishing is repeated during the entire attack. A publishing round takes about 5 seconds, during which we try to publish on about 2,200 peers, out of which about 400 accept our publications. The peers that do not respond did either previously leave the network, could not be contacted because they are behind a NAT gateway, or are overloaded and could not process our publication.

Once a search is launched by any regular client or bot, it searches on peers closely around K and will then receives so many results (our fake announcements) that it is going to stop the search very soon and not going to continue the search further away from K. That way, publications of K that are stored on peers far away from K do not affect the effectiveness of the attack as they do for the eclipse attack.

We evaluate the effectiveness of the pollution attack by polluting a hash used by Storm and searching at the same time for that hash. We do this using two different machines, located at two different networks. For searching we use `kadc` [16], an open-source OVERNET implementation, and an exhaustive search algorithm we developed. Our search method is very intrusive, it crawls the entire network and asks every peer for the specified content with key K. Figure 6a shows that the number of Storm content quickly decreases in the results obtained by the regular search algorithm, then nearly completely disappears from the results some minutes after the attack is launched, and finally comes back after the attack is stopped. However, by modifying the search algorithm used, by asking all peers in the network for the content and not only the peers close to the content's hash, the storm related content can still be found (Figure 6b). Our experiments show that by polluting all those hashes that we identified to be *storm hashes* (see Section 4.2.2), we can disrupt the communication of the botnet.

5 Conclusion

In this paper, we showed how to generalize the methodology of botnet tracking for botnets with central server to botnets which use P2P for communication. We exemplified our methodology with a case study on Storm Worm, the most wide-spread P2P bot currently propagating in the wild. Our case study focussed on the communication within the botnet and especially the way



(a) Using the standard search. (b) Using the exhaustive search.

Figure 6: The number of publications by Storm bots vs. the number of publications by our pollution attack.

the attacker and the bots communicate with each other. Storm Worm uses a two-tier architecture where the first-tier is contained within the P2P networks OVERNET and the Stormnet and used to find the second-tier computers that send the actual commands. We could distinguish the bots from the benign peers in the OVERNET network and identify the bots in the Stormnet and give some precise estimates about their numbers. Moreover, we presented two techniques how to disrupt the communication of the bots in both networks. While eclipsing is not very successful, polluting proved to be very effective. In future work, we plan to analyze in detail the second-tier computers and try to find ways to identify the operators of the Storm Worm.

References

[1] P. Baecher, M. Koetter, T. Holz, F. Freiling, and M. Dornseif. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of 9th International Symposium On Recent Advances in Intrusion Detection (RAID'06)*, 2006.

[2] J. Ballard. Storm Worm, October 2007. NANOG 41, `http://www.nanog.org/mtg-0710/kristoff.html`.

[3] P. Barford and V. Yegneswaran. *An Inside Look at Botnets*, volume 27 of *Advances in Information Security*, pages 171–191.

[4] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2:67–77, 2006.

[5] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*, pages 39–44. USENIX, June 2005.

[6] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, 2006.

[7] J. R. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, LNCS, pages 251–260, March 2002.

[8] B. Enright. Exposing Stormworm, October 2007. Toorcon 9, `http://noh.ucsd.edu/~bmenrigh/`.

[9] Federal Bureau of Investigation (FBI). Operation Bot Roast, February 2007. `http://www.fbi.gov/pressrel/pressrel07/botnet061307.htm`.

[10] Frank Boldewin. Peacomm.C - Cracking the nutshell, September 2007. `http://www.reconstructer.org/`.

[11] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proceedings of 10th European Symposium On Research In Computer Security (ESORICS'05)*, July 2005.

[12] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[13] L. Grossman. The worm that roared. Internet: `http://www.time.com/time/magazine/`, September 2007.

[14] 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*.

[15] Honeynet Project. Know your Enemy: Tracking Botnets, March 2005. `http://www.honeynet.org/papers/bots`.

[16] KadC. `http://kadc.sourceforge.net/`.

[17] B. Krebs. Storm worm dwarfs world's top supercomputers. Internet: `http://blog.washingtonpost.com/securityfix/`, August 2007.

[18] Maxmind. `http://www.maxmind.com/`.

[19] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer informatiion system based on the XOR metric. In *Proceedings of the 1st Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[20] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.

[21] D. Moore, C. Shannon, and k claffy. Code-red: A case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 273–284, New York, NY, USA, 2002. ACM Press.

[22] J. Naughton. In millions of windows, the perfect storm is gathering. `http://observer.guardian.co.uk/`, Oct 2007.

[23] P. Porras, H. Saidi, and V. Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, Computer Science Laboratory, SRI International, October 2007.

[24] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th Internet Measurement Conference*, 2006.

[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 *Proceedings of 1st Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[26] A. Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. Infocom 06*, Apr. 2006.

[27] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting KAD: Possible Uses and Misuses. *Computer Communication Review*, 37(5), Oct 2007.

[28] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack. Load reduction in the kad peer-to-peer system. In *Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2007)*, 2007.

[29] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proceedings of the Internet Measurement Conference (IMC)*, 2007.

[30] J. Stewart. Storm worm DDoS attack. Internet: `http://www.secureworks.com/research/threats/storm-worm`, 2007.

[31] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[32] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), 2007.