Leveraging Honest Users: Stealth Command-and-Control of Botnets

Diogo Mónica
INESC-ID/IST

Carlos Ribeiro
INESC-ID/IST

Abstract

Botnets are large networks of infected computers controlled by an attacker. Much effort has already been invested in the detection and analysis mechanisms, capable of defusing this type of threat. However, botnets have been constantly evolving, and will certainly continue to do so. We must, therefore, make an effort to foresee and study possible future designs, if we are to be capable of timely development of adequate defense mechanisms.

Many of the most recent methods to detect and analyze botnets are based upon the vulnerabilities of their command-and-control (C^2) infrastructure. We thus believe that attackers will follow a predictable evolutionary pattern, and start using designs with more robust and stealth C^2 channels, thus minimizing the risk of shutdown or infiltration. In this paper, we will therefore analyze in detail a new kind of botnet C^2 infrastructure, where bots do not possess any information concerning command-and-control mechanisms. These stealth, isolated bots are controlled through honest participants not pertaining to the botnet. This architecture eliminates the possibility of estimation of the botnet size, minimizes the probability of detection of individual bots, and eliminates the possibility of researcher infiltration.

1 Introduction

Botnets are one of the dominant threats in the internet, since they harness the power of thousands of infected hosts, and may use them to perform abusive or fraudulent activities, such as: spamming, phishing, click-fraud, distributed denial-of-service (DDoS), or stealing personal data, such as email accounts or credit-card data [5]. A significant amount of research has been invested in this area, leading to both a better understanding of the botnet phenomenon, and to the development of new detection techniques.

Most of the initially proposed methods for botnet detection are passive, and work either by detecting specific malicious activity of bots (spam, DoS) [27], or by detecting activity on the communication channels used to control the botnet [11, 24, 10]. However, these passive approaches can only monitor a small portion of the internet; also, they will only detect botnets with the malicious behavior or the type of C^2 targeted by that particular analysis.

More recently the research community adopted a more active approach to the study of botnets: infiltration. They create a client capable of mimicking the C^2 protocol, and join the botnet. This privileged position can then be leveraged, to obtain accurate estimates of the size of the botnet, or to gain inside information needed to support an eventual hijacking or dismantling of the botnet [23, 12, 15].

Unfortunately, botnets are also evolving. The first kind of C^2 infrastructure used by botnets was based on central IRC servers [8]. This allowed researchers to obtain the IP addresses of the bots, and, sometimes, even control the botnet itself [5]. Botmasters then upgraded the servers to stripped-down IRC or HTTP servers, which reduced some of the vulnerabilities of the previous servers (namely, the possibility of identification and estimation of the number of bots in the botnet). However, botnets still had a single point of failure: the C^2 server itself. Therefore, botnets began shifting towards peer-to-peer command-and-control infrastructures, thus becoming more resilient to takedown attempts. Forecasting the evolutionary adaptive changes in botnet operation profiles is, therefore, of paramount importance, to allow timely development of appropriate countermeasures.

Of particular concern are the stealth strategies that attackers may employ, to reduce the probability of detection of their bots, and/or of infiltration of their botnets. The intuition is that botmasters will try to avoid the usage of permanent command-and-control infrastructures, since they have been one of the main targets of recent research on botnet detection and analysis [11].

To create this new kind of botnet, attackers can leverage the capability of modern browsers to run pieces of embedded code (such as Javascript or Flash), thus creating a layer of expendable hosts, composed by honest visitors of malicious (or compromised) websites. These hosts will be responsible for doing activities that are verbose in nature, such as disseminating botmaster's commands. As will be seen, the existence of this layer can constitute a formidable obstacle to the analysis, and/or infiltration of the underlying botnet.

Contributions The main contribution of this paper is the exposure and analysis of a possible path of botnet evolution, capable of higher degrees of stealthiness and resilience against infiltration. Hopefully, this will lead to the timely development of adequate countermeasures against this next generation of botnets. In particular, several novel results are presented:

- We present and analyze new type of botnet C^2 architecture, capable of achieving much higher degrees of resilience to infiltration and size estimation by the research community;
- We show that such an architecture can be leveraged to create an overlay (for stolen information retrieval) that cannot be used by anyone but the botmaster;
- We analyze authentication scheme that allows bots to unambiguously identify honest hosts being used by the botmaster to disseminate commands, using only the botmaster's public key as a trust anchor;
- We also discuss a tunneling scheme that may allow the botmaster to retrieve information without the risk of exposure of his own identity;
- A case-study implementation of such an architecture is made;
- Also, the performance of this architecture is analyzed, both analytically and via Monte Carlo analysis, thus obtaining the practical bounds within which this overall botnet scheme may constitute a threat to network operations.

2 Background Model

As stated, our approach is focused on stealth botnets, whose command-and-control architecture can avoid infiltration, size estimation, and reduce the detection probability of the individual bots. In this section, we will state both the model and the assumptions to be made throughout the paper.

The details of how bots are infected will not be addressed. We will assume the existence of a population of computers infected with a malicious binary, which contains: the code needed to perform the malicious activities; the public key of the botmaster; the initial botmaster's commands (e.g. send spam e-mail). Note that bots do not possess any data (e.g. IP address, domain-names) concerning the botmaster, other than its public key.

We further assume that bots can receive commands from the botmaster through a common TCP port open to the internet. Attackers can achieve this even in hosts sitting behind NATs, through the use of the Internet Gateway Device (IGD) Protocol, implemented via UPnP, thus providing port mapping capabilities. Also, it is common in malware to have firewall killers, and, in general, to eliminate some of the restrictions that a well configured host may present.

Finally, and in order to simplify calculations, we assume that the IP addresses of infected hosts are uniformly distributed throughout the fraction of the IP addressing space currently assigned, and that the whole population of infected hosts is online.

3 Architecture

Contrarily to usual botnets, in our architecture bots do not attempt to contact any controlling third-party server; they simply execute the original botmaster's orders, and passively listen for commands. This can be seen as a change in the normal communication paradigm. Usually, bots actively participate in some control infrastructure, both to retrieve information from the botmaster (e.g. updates to the malicious binaries, new commands, etc.), and to send information back (e.g. stolen credit card data, email addresses, etc.). However, in the considered architecture, bots are passive in terms of C^2, and commands are pushed directly into the compromised machines. Command origin authentication is done by verifying if the incoming requests are signed with the botmaster's private key (remember that all the bots have the corresponding public key). This type of authentication is already being used in the wild [21].

This design presents a much greater threat than the usual usual pull model: Researchers controlling individual bots cannot infiltrate the C^2 infrastructure, since, from the perspective of a bot, there is none. This means that, by controlling a bot, researchers cannot estimate botnet size, or gain any type of privileged position in the botnet that might lead to a possible attack vector. Moreover, none of the research methods for bot identification by detection of the C^2 communication pattern is applicable, since there is no outbound (from bots to botmaster) C^2 traffic.

However, attackers face two obvious difficulties with this overall design:

Command dissemination. The botmaster does not possess an established command dissemination channel, and, thus, needs to individually send the commands to each and every infected bot; however, since bots never "phone home", the attacker does not know the IP addresses of the infected hosts, and commands will thus have to be sent to the bots by a random (or semi-random) scanning strategy. Command dissemination will thus become a problem of statistical nature;

Information retrieval. Since bots do not possess the IP address of the botmaster, creating an upstream information flow (if information retrieval is required) is a particularly difficult problem.

Unfortunately, both these difficulties may be overcome by a motivated botmaster.

3.1 Command dissemination

Direct command dissemination by the botmaster, while feasible, has some disadvantages for the botmaster:

- The botmaster would be exposing himself, by directly scanning the internet. Even if the attacker uses a compromised host to act as a proxy, exposing a host directly controlled by the botmaster may lead to exposure of the botmaster's true IP address;
- The botmaster may take too long to reach all the participants in the botnet. This is particularly important when the botmaster wants a command to be executed as soon as possible.

The botmasters might consider establishing a two-leveled hierarchy of infected hosts, with different objectives on both levels: one lower level to perform the malicious activities (*worker bots*), and a higher intermediate level, whose sole purpose would be to disseminate the botmaster commands to the worker bots (as was done in [25]). If botmasters opt for this solution, one may try to impersonate one of these information relay hosts which might allow us to thwart the propagation of the commands, gain enough trust to be able to launch an eclipse attack, or launch other disrupting attacks to the C^2 infrastructure [13].

Unfortunately, we believe that botmasters may try to circumvent both theses difficulties through the use of a new kind of intermediate C^2 layer, between the botmaster and the bots, composed solely of *honest* participants. These participants are called *honest*, since they will not be infected with the malicious binary, and are, thus, technically not part of the botnet. This intermediate C^2 layer

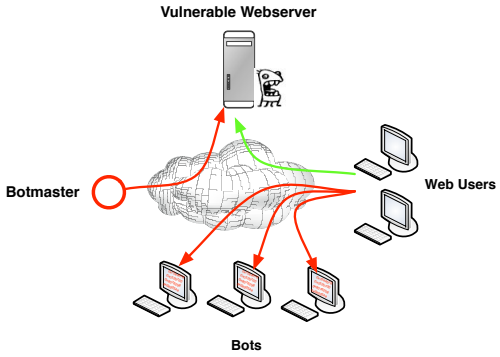

Figure 1: Global architecture.

works in the following way: The botmaster deploys a website (or infects an existing one) possessing a specific malicious code. This malicious code will not attempt to infect the host through a drive-by-download, gain control of the web user's hosts, or exploit any vulnerability. Instead, it will have an obfuscated piece of code, to be ran by the browser, that will make the web user's browser unwillingly disseminate command messages to the bots.

This may be done by leveraging the capabilities of scripting languages like javascript. All users with javascript enabled on their browsers will execute the malicious code, and will, thus, become a part of the command dissemination layer. Such a command dissemination mechanism can be seen in Figure 1.

Bots will receive commands from the intermediate *honest* website users, without acknowledging their reception[1]. The only traffic sent by the bots is the one generated by the TCP handshake (one SYN/ACK message). Furthermore, attackers may chose commonly open ports belonging to potentially legitimate services, hindering researchers from simply enumerating bots by the presence of an uncommon open port. This makes the task of detecting C^2 traffic very difficult. Moreover, a researcher cannot achieve anything by replaying the received commands, except further spreading of the command throughout the infected population.

This intermediate layer of command dissemination presents several advantages for the attacker.

- Command dissemination is not done by the botmaster, reducing the probability of it being detected.
- Detection of a command dissemination attempt, only reveals the address of a honest user accessing some malicious website. It does not reveal any part of the botnet, nor the malicious website itself.

[1]For the moment this assumption creates a worst-case scenario which will be abandoned at a later section.

– Since there will be typically many victims accessing the malicious website, the time taken to deliver the commands to the bot population decreases dramatically.

A skeleton version of this command dissemination scheme (including the intermediate layer build-up via a malicious website) has been implemented, for *case-study* purposes. Implementation details, and a detailed performance analysis will be presented in Section 4 to appreciate it's threat potential.

3.2 IPv6

It has been repeatedly shown that a full scan of all of the IPv4 space is feasible. With the advent of IPv6, the IP address space will change from from 32 to 128 bits, dramatically increasing the amount of work needed by random scanning worms to cover the whole internet by a factor of 2^{96} [18]. However, this increase in addressing space might not be effective in stopping this method of propagation. In [14], the propagation speed of a worm in an IPv6 world is analyzed, and the authors present results indicating that such a worm can exhibit propagation speeds comparable to an IPv4 random-scanning worm. In this work, the authors realized that the directory and naming services necessary for the functioning of the internet can be exploited, thereby considerably reducing the required scanning range. More specifically, instead of scanning random IPs, worms can perform DNS random scanning, i.e., guessing DNS names instead of IP addresses. Others works like [28, 18, 29] present several other techniques that allow worms to reduce the search space: scanning only assigned space; using public information about the currently active IP ranges; using existing peer-to-peer networks to gather IP addresses; using search engines and DNS zone transfers, amongst others.

Furthermore, there is one advantage that IPv6 networks bring to random scanning worms, and therefore, to the botnet architecture presented: the effective disappearance of NATs. With the increase of the use of public IPv6 addresses for regular hosts, the number of bots reachable from the internet will also increase, benefiting potential attackers.

The advent of IPv6 will, therefore, not restrain malicious attackers from using random scanning schemes on their future botnets and might end up being beneficial for for attackers.

3.3 Information Upstream

Until now, we have been describing how the attacker may attempt to disseminate commands to the bots, in a stealth manner. However, bots are often required to send information back to the botmaster (e.g. credit-card data). In the discussed architecture bots cannot send information to the botmaster or some rendezvous host, since they do not possess the required IP addresses, or links to any other botnet participant. This isolation ensures a high degree of robustness and stealthiness of the botnet, but creates a challenging environment if an upstream channel is required. However, even this difficulty may be circumvented by the botmaster.

We will describe two possible implementations of an upstream information channel in these stringent conditions. Each one of them is capable of providing a workable upstream channel, while preserving the three following characteristics:

1. The fact that they cannot be infiltrated by a researcher through the C^2 channel;
2. The stealthiness (and, therefore, detection avoidance) conferred by the absence of a traditional C^2 channel;
3. The fact that compromising any number of bots does still not allow botnet size estimation.

3.3.1 Spamming Botnets

If the main objective of the botnet is sending spam, one rather ingenious way of solving the upstream problem, is sending the information encoded along with the spam email, thus hiding the upstream communication in plain sight. This can be done without allowing researchers to read the information, by encrypting all the transmitted information with the botmaster's public key. With this upstream solution, there is virtually no communication done by the bot that doesn't fit the "sending spam" profile, since bots are never required to respond to a command, or have any kind of interaction with the rest of the botnet. To access the data, the botmaster only has to receive spam, something very easy to accomplish these days.

Obviously, researchers can use the spamming activity to detect the active participants of a botnet, but an active bot will, at most, only compromise itself.

This upstream solution raises again the issue of randomness and performance in the information retrieval process. The analysis of this specific information upstream channel will not be presented in this paper.

3.3.2 Information gathering Botnets

In the case of botnets whose purpose is solely covert information gathering (and not spam), the previous scheme is not an option. In this type of botnet, information is usually stolen locally from infected hosts, through the use of key-loggers, and other mechanisms that usually do not require interaction with external systems. This

means that, to send information upstream, bots will have to break silence, even though, assumedly, the amount of involved information is not big enough, in itself, to lead to bot detection. However, it is still possible to create an upstream channel that:

1. Minimizes the risk of detection;

2. Allows the bots to keep operating undetected;

3. Does not expose the botmaster.

To achieve this, the command dissemination mechanism must change. In particular, bots receiving a valid command must now send an acknowledge to the dissemination layer host from whom the command originated. This information will allow the hosts in the dissemination layer to detect which contacted hosts are part of the botnet, something that will be crucial for allowing the botmaster to retrieve data from the bots, as will be seen further ahead in the text.

Clearly, requiring bots to acknowledge commands, opens a path for researchers to identify the participants in the botnet. Since commands are disseminated randomly over the internet, researchers can easily collect them. Therefore, to determine if a specific IP address is part of the botnet, a researcher could, in principle, simply replay one of the received commands, and verify if an acknowledge was sent back. Unfortunately, even this weakness can be circumvented by the botmaster, as discussed below.

Dissemination layer authentication If we could forge or replay commands, prompting acknowledgement from the bots, the botnet size could be estimated, and the infected machines identified. However, botmasters must ensure that bots only acknowledge commands directly originating from hosts in the dissemination layer. The authentication must be done by the bots without access to any information other than the botmaster's public key (that is the only trust anchor they possess), and without the need to transmit (in order to avoid detection).

Such an authentication scheme can be devised, based on a simple assumption: researchers are unable to access the code of the malicious website used to gather dissemination layer hosts, before expiration of the command issued by the botmaster (commands sent by the botmaster will thus possess a time-validity stamp). This scheme requires, not only the existence of a public/private key-pair owned by the botmaster (K_{bm}/K_{bm}^{-1}), but also one key-pair for each of the deployed dissemination websites (K_w/K_w^{-1}). The protocol is depicted in Figure 2. The following abbreviations are used in this figure: K_x/K_x^{-1} as x's public/private keys; $\{m\}K_x^{-1}$ representing m signed

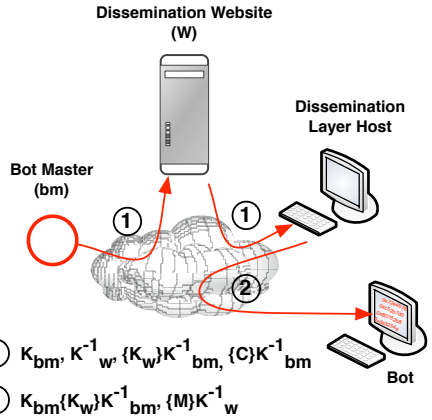

Figure 2: Dissemination layer authentication.

with x's private key; C represents the command to disseminate; finally, M is the message sent to the bot. Protocol operation progresses as follows:

1. After compromising a vulnerable website, the botmaster inserts the malicious code, which includes the following information: the botmaster's public key (K_{bm}); the site-specific private key (K_w^{-1}); the site-specific public key, signed with the botmaster's private key ($\{K_w\}K_{bm}^{-1}$); and the command to be sent, also signed with the botmaster's private key ($\{C\}K_{bm}^{-1}$);

2. When any host accesses the infected website, the malicious code is sent to the host, to be executed;

3. This host (which is now part of the command dissemination layer) sends the command message (M), signed with the site-specific private key ($\{M\}K_w^{-1}$), to possible bots. It also sends the site-specific public key, signed with the botmaster's private key ($\{K_w\}K_{bm}^{-1}$). Message (M) contains the command to be executed (C) and the destination bot address;

4. The bot verifies the authenticity of M and C, and the associated time stamp. If M and C are authentic and not yet expired, command C is executed, and an acknowledge is sent to the intermediate layer host.

This protocol creates a verifiable trust chain between the botmaster's public key, and message M: the bot verifies if the site-specific public key received is signed by the botmaster, using the known botmaster's public key; then, it uses that public key to verify the signature of message M; finally, it verifies if the Dst IP corresponds to it's own IP address. If these three steps are successful, the bot knows that the host from which this message was received, has had access to one of the botmaster's

command dissemination websites. Note that the bot still has to verify the authenticity of C, and the time-stamp associated with it. Under the assumption that researchers will not be able of accessing the code in the dissemination website before command expiration (expiration time may be chosen in such a way as to guarantee that this assumption holds true with a desired high probability), reception of an authentic, non-expired command allows the bot to reply with an acknowledge message, without exposing itself to researchers. If this assumption does not hold true, and researchers are somehow successful in obtaining access to the dissemination website prior to the expiration of C, all we obtained is the capability to identify IP addresses pertaining to the botnet, with no better strategy than random scanning within a short time interval. The insertion of false commands will still not be possible.

Overlay Construction We will now discuss how the botmaster can retrieve bot information from the dissemination layer without becoming exposed.

One possible way of implementing the upstream channel is to have the hosts in the dissemination layer retrieve the information from each contacted bot, and post it back to the compromised websites. However, this would create one (or several) places to which the botmaster would have to access for final information retrieval. If one or more of these points could be identified, we could wait for botmaster access, and accomplish direct botmaster identification. Moreover, by taking these websites down, we could also prevent the botmaster from retrieving the stolen information.

The botmaster may, however, build an overlay within the bots of the botnet, which only the botmaster can locate and use, to retrieve upstream information from the bots. The mechanism is as follows: when a host in the dissemination layer receives an acknowledge from a bot to which it sent a command, it encrypts the bot's IP address with the botmaster's public key. Then, it pushes this information to every other bot found in subsequent dissemination attempts. Each time a new bot is found, its encrypted address is appended to the list, and pushed to subsequently found bots. Each host in the dissemination layer is, therefore, creating a path (a multiply-linked list, in fact) containing all the bots it finds while disseminating the commands. Each bot will possess a list of the encrypted addresses of the bots previously contacted by the same dissemination layer host. Since the list is encrypted with K_{bm}, only the botmaster is capable of using it. The mechanism is depicted in Figure 3.

As will be seen in the analysis section, for realistic values of the parameters, each one of these individual paths will not possess, on average, more than a couple dozens of IP addresses. Since the IP addresses scanned by the

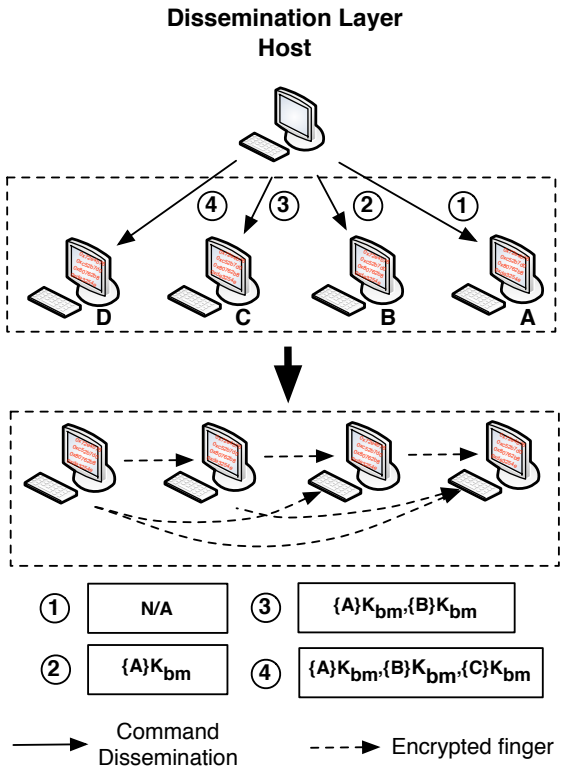

Figure 3: Construction of an encrypted bot overlay by the intermediate hosts.

dissemination layer are chosen at random, the individual paths created by different dissemination hosts may, however, intercept. If the number of intersections is large enough, the overlay may present a high degree of connectivity, extending throughout the set of contacted bots. The degree of connectivity of the overlay, as a function of the involved parameters, will be seen in the analysis section (Section 4). This overlay can be easily extended to have bidirectional paths, by having the dissemination hosts sending the encrypted IP address of a newly found bot, to the last bot (or k bots) contacted.

With the overlay in place, retrieving information from the botnet becomes a trivial task to the botmaster: he simply has to find one of the bots of the overlay (by random scanning), and then travel through the overlay to find the rest of the bots in the overlay, pulling the information stored in each bot along the way. We stress the fact that, since the fingers are encrypted with K_{bm}, only the botmaster can navigate the overlay.

Accessing the Overlay By simply joining the botnet with honeypots, and waiting for information retrieval requests, we might be capable of recording the IP address of the botmaster, if we were part of the overlay, or if we

were contacted by the botmaster while trying to reach a bot in the overlay. This might give us an edge, even though only a probabilistic one.

The botmaster can, however, circumvent this weakness. Let us consider the following algorithm:

1. The botmaster (to be referred as Bot_0) contacts one of the bots in the overlay (let us call it Bot_1), and requests its overlay list (which is encrypted with K_{bm}, as discussed above).

2. With probability ρ, Bot_0 nominates Bot_1 to be responsible for information retrieval, and the algorithm proceeds to Step 5. With probability $(1-\rho)$, the botmaster returns one of the overlay bot addresses (randomly chosen, and in plain text) to Bot_1, and Bot_1 is requested to connect with the bot in that address (call it Bot_2), and obtain its overlay list (which Bot_1 will forward back to Bot_0);

3. With probability ρ, Bot_0 nominates Bot_2 (through Bot_1) to be responsible for information retrieval, and the algorithm proceeds to Step 5. With probability $(1-\rho)$, the botmaster returns one of the overlay bot addresses (randomly chosen, and in plain text) to Bot_2 (via Bot_1), and Bot_2 is requested to connect with the bot in that address (call it Bot_3), and obtain its overlay list (which Bot_2 and Bot_1 will forward back to Bot_0);

4. And so on, for Bot_3, Bot_4, ...Bot_q;

5. The bot nominated as responsible for information retrieval (Bot_q) crawls the full overlay, using the botmaster (via the tunnel) to decrypt the overlay address fingers of each successive bot, and sending the collected information back through the established tunnel. This step will be further refined below.

With this algorithm, the botmaster is creating a tunnel through which the information will be retrieved, the retrieval itself being a task of the last bot in the tunnel Bot_q. Since the retrieved information is sent to Bot_q already encrypted with K_{bm}, only the botmaster will be able to read it.

The intuition of this approach is that each bot in the tunnel looks to its successor in the exact same way the botmaster looks to the first contacted bot. Hence, there is no way for a contacted researcher to determine if his antecessor is the botmaster, or simply an innocent bot at an intermediate position in the tunnel. On the other hand, nodes in this tunnel only know the IP addresses of the previous and the next bots in the tunnel, and, thus, the tunnel cannot be traced back to its origin. This mechanism is depicted in Figure 4.

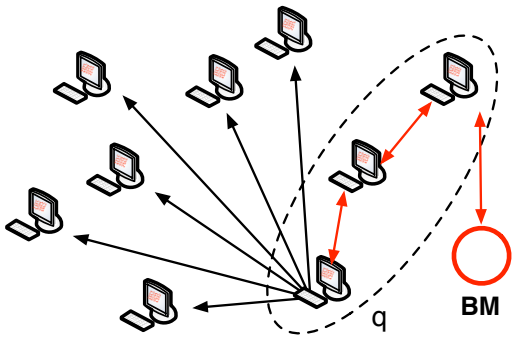

Figure 4: Construction of a tunnel through q randomly chosen bots.

4 Analysis

In this section, we will analyze the performance of both the downstream and upstream techniques described in the previous sections. This analysis will highlight the impact on performance of the several system parameters, and provide the practical bounds within which this overall botnet scheme may constitute a threat to network operations. The performance of both the command dissemination and the overlay construction mechanisms cannot be validated in a real-world scenario, since this would require controlling several thousands of nodes, which largely exceeds the number of nodes at our disposal, even when using platforms such as Planetlab. The performance analysis will, thus, use both analytical and simulated results without the benefit of real-life deployment.

We will assume a botnet with a static population of $N = 150000$ bots. This number of bots is easily achievable, and in fact, there are botnets that largely exceed this number [23]. These bots will be considered to be uniformly distributed throughout the presently assigned IPv4 address space. Using the Bogon list ([2]), which contains unallocated or reserved address blocks in the IPA address space, only $S = 3086889768$ possible IP addresses will be considered. We must also consider: the rate at which the browsers running on the intermediate hosts are capable of sending out requests (r); the number of days the dissemination website is up, or the malicious code is active on the dissemination site (d); the number of visitors the dissemination website receives per day (v); and the average number of minutes that a user spends on the dissemination website (m).

Without loss of generalization, two of these parameters, r and d, will be considered constant throughout the analysis. Since the scanning rate r varies from browser to browser, we tested it on three browsers (Firefox, Safari, and Chrome), and on two distinct platforms (Windows and OS X). We obtained approximately the same perfor-

mance in all cases (about 250 requests per second). This is well below the reported performance of some previous well known cases (e.g. Code Red, [30]). Parameter d will be assumed to be $d = 1$, which means that we will assume only 24 hours of dissemination layer activity, until command (C) expiration. This choice is based on the assumption that researchers need at least 24 hours to get the source code of the dissemination website. This seems to be a conservative assumption, since, as reported in the literature (e.g. [19]), it has been difficult to collect scripts even from malicious websites that stay online for longer periods of time.

4.1 Case-study implementation

For a case-study, we implemented some parts of the described architecture. To recruit intermediate layer hosts, we created a Javascript program that was loaded and ran on the *victims*' browsers[2]. The reason for the choice of Javascript is the fact that it is both present in all modern browsers, and turned on by default.

We started by implementing a script capable of bypassing the browser's *same origin* policy. This policy prevents access to most methods and properties across pages on different websites [4]. Unfortunately, there are widely available tools such as EasyXDM [3], that allow bypass of this security mechanism across all mainstream browsers[3]. Then, the malicious script was made to scan a given IP address range, by sending AJAX requests to all the IP addresses, and collecting correct replies from the simulated bots. We also tested the rate at which our malicious script was able to generate AJAX requests, to obtain parameter r (the scan-rate of the intermediate hosts). The significative difference between the scan-rate obtained and the scan rates of usual worms can be explained by the fact that browsers are not optimized to send large numbers of simultaneous requests. Finally, we tested public-key encryption libraries in Javascript, to verify the feasibility of the dissemination layer authentication mechanism.

Adobe Flash could also have been used. It is still ubiquitous in modern browsers, and it also possesses the capability of doing HTTP requests to hosts, and, therefore, the capability of command dissemination. One interesting characteristic of Flash is Adobe Stratus (see [1]), a service that allows clients to send data from client to client, without passing through a server (P2P communication). This capability would allow cooperation between the intermediate hosts, thereby increasing the number of bots reached in each dissemination campaign, as will be seen further ahead in the text.

On the issue of how to get honest website visitors to run malicious code, there are two basic approaches: attackers may create their own websites with that specific purpose in mind [26], and advertise them through spam email, trending topics on twitter, search engine poisoning, abusing URL shorteners, etc; alternatively, existing websites can be used. We note that common vulnerabilities like XSS and SQL injection are sufficient to get legitimate users running malicious code. The most common method for malware infection is precisely drive-by downloads, in which attackers create malware-serving websites to recruit bots. This is, in fact, commonly done. Using these sites to also recruit intermediate layer hosts would imply only a minor addition to common attack procedures.

4.2 Command Dissemination

One of the first performance issues to be addressed is the degree of completeness of command diffusion. Since bots are completely stealth (they simply listen for commands from the botmaster), their IP addresses are not known to the botmaster, or to the intermediate hosts, and commands must be sent randomly to the internet address space, in the hope of reaching a considerable portion of the universe of existing bots. In this analysis, we will assume that a single host does not repeat IP addresses, even though any address may turn out to be scanned as much as v times, once by each host in the dissemination layer. This inefficiency in the global scanning strategy could be fixed, if the botmaster would allow cooperation between the intermediate hosts. Attempting to reach all bots in practical time is an impossible goal, but the possibility of reaching a significant percentage of bots must be investigated.

Since each one of the hosts in the dissemination layer has a fixed scanning rate r, and d is considered fixed, there are only two parameters influencing the mean percentage of bots that will receive the disseminated commands: v and m. Denoting by $s(n)$ the number of bots reached after n randomly addressed requests (n being the sequential number of a given request within the global set of all requests transmitted by the v intermediate hosts), the equation governing bot discovery by random scan of the used IPv4 address space is given by Equation 1:

$$E\{s(n)\} = E\{s(n-1)\} + \frac{N - E\{s(n-1)\}}{S - \frac{n}{v}}, \quad (1)$$

$E\{\cdot\}$ being the expected value operator.

The solution to this difference equation, when written as a function of the number of hosts in the dissemination layer (v) is given by Equation 2:

[2]The victims referred here, are test computers allocated for that specific purpose.

[3]HTML5 formalizes a method for this: the postMessage interface, removing the need for external tools.

$$E\{s(v)\} = N\left(1 - \left(1 - \frac{m \cdot r \cdot 60 \cdot d}{S}\right)^{v}\right). \qquad (2)$$

In Figure 5, we plotted the mean fraction of bots receiving the disseminated command sent out by the botmaster, as a function of the number of hosts in the dissemination layer (Equation 2), for three different values of m ($m = 10$, $m = 15$ and $m = 20$).

As a side note, and as a justification for our choices of the value of m, once a victim visits a malicious website, attackers can use a combination of Clickjacking and Tabnabbing to ensure that the malicious javascript is kept running for as long as possible. Additionally, the introduction of WebWorkers by HTML5 gives attackers the ability of executing threads running malicious code without slowing down or making the victim's browser unresponsive, which also helps attackers maintain their code running without detection. These tricks, together with the concept of tabbed browsing, mean that most users will have each individual tab remain open throughout the browsing session, which could stretch for hours. This easily puts the average lifetime of an individual instance of the malicious javascript in the order of tens of minutes.

As previously stated, the dissemination strategy could be made considerably more efficient if the botmaster allowed cooperation between the intermediate hosts, since in that case no address would be scanned twice, implying a constant bot discovery rate. To appreciate this increase in efficiency, the bot discovery performance of such an approach is also represented in Figure 5, for the case of $m = 20$. From Figure 5, we can conclude that the non-cooperative approaches require, for effectiveness, a medium sized dissemination site ($2000 < v < 20000$). Sites with lower visitor rates will generate low populated dissemination layers, and low levels of access to the bot population (percentages of botnet coverture smaller than 20%). Alternatively, a set of smaller sized web sites would be needed.

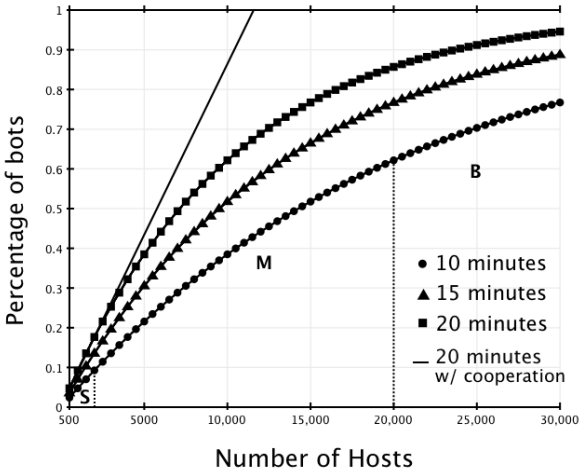

Figure 5: Fraction of population reached with a varying number of dissemination hosts for (S)mall, (M)edium and (B)ig websites.

4.3 Overlay Connectivity

The total number of bots reached in the command dissemination phase is not, however, the only parameter from which the threat level depends. Another issue that must be addressed is the capability to form an overlay of sufficient size to allow proper operation of the upstream information retrieval mechanism (see Section 3). The problem is two sided: we must evaluate the mean size and statistical distribution of the path of bots reached by each one of the intermediate hosts; also, we must consider the connectivity between these different individual paths, since both parameters will determine the maximum size of the connected overlay and, hence, its information retrieval capability. The worst case overlay will be one that includes all the bots reached in the command dissemination phase, a situation that corresponds to a fully connected set of individual paths.

The number k of successes (k contacted bots) in the process of randomly scanning the IPv4 address space by a single host follows a hypergeometric distribution. That is:

$$P(k) = \frac{\binom{N}{k}\binom{S-N}{m \cdot r \cdot 60 \cdot d - k}}{\binom{S}{m \cdot r \cdot 60 \cdot d}} \qquad (3)$$

Its mean value (mean length of the individual bot paths) is given by $(m \cdot r \cdot 60 \cdot d \cdot N)/S$ and is, therefore, a linear function in both r and m.

Evaluating the connectivity of the global overlay (that is, the connectivity of the set of individual bot paths) now becomes a typical problem of graph analysis. To perform this evaluation, we simulated several dissemination campaigns using Mathematica. For each campaign, v bot chains of random IP addresses (from the set of S used addresses) were generated. The length of each individual chain was obtained with (3) with the appropriate parameters. We then evaluated the overall connectivity of the generated graph, and determined the mean size of the biggest cluster of bots. As a preliminary note, we note that the parameter with the biggest impact on the connectivity (length) of the overlay is the number of minutes spent on the dissemination website by the visitors (m), due to its impact on the length of the individual paths.

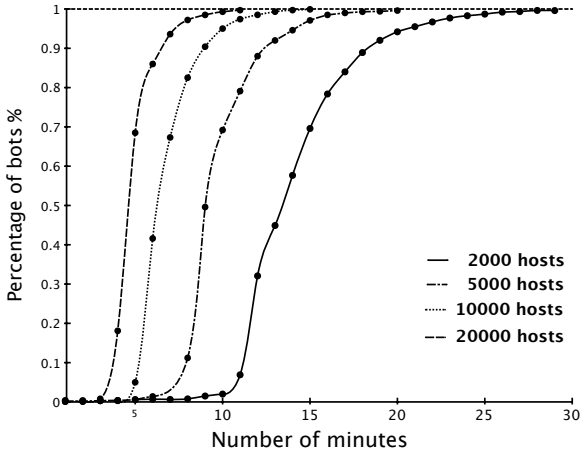

Figure 6: Percentage of overlay connectivity with a varying number of minutes of dissemination.

In Figure 6 we present the mean size of the largest connected cluster of bots (as a percentage of the total number of bots present in the graph), plotted as a function of m. The Monte Carlo simulation to generate this curve consisted of 100 different simulated dissemination campaigns, with $N = 150000$, $S = 3086889768$ and $v = \{2000,\ 5000,\ 10000,\ 20000\}$. We can see that, even for the smallest value of v, $m > 20$ implies that the set of bots contacted in the dissemination command becomes almost fully connected, thus creating a worrisome overlay, highly capable of supporting information retrieval.

A final performance issue should be addressed: the mean amount of time required for the botmaster to contact the overlay. We should note, however, that this is not a critical issue. Once the overlay is set up, there is no particular timeline for the botmaster to comply with in this last information retrieval phase. However, the existence of dynamics in the bot population may introduce breaks in the connected overlay (bots going off-line), splitting it into smaller sub-paths and, thus, hampering the effectiveness of the retrieval process. Hence, we should also consider this additional parameter l - the mean number of scan attempts it takes the botmaster to contact the overlay, by randomly scanning the used IP address space (l can then be directly translated into time). Since the tunneling scheme doesn't affect the probability structure of the scan, the scan process is still governed by a hypergeometric arrival process (Equation 3), but with N replaced by the mean overlay max size (p). This implies that the probability mass distribution for the number of scan attempts needed to reach one of the bots of the overlay (u) is:

$$P(u) = \left[\frac{p}{S-p-u+1}\right] \prod_{i=0}^{u-1} \frac{S-p-i}{S-i} \qquad (4)$$

The mean number of tries required to reach one of the bots of the overlay becomes:

$$l = E\{u\} = S/p. \qquad (5)$$

With the results of Equation 5, Figure 6 and Equation 2, the effectiveness of the overall botnet downstream and upstream flows can be evaluated and, thus, the overall danger to network operations properly considered.

5 Discussion

These results show that the possibility of emergence of this type of botnet constitutes a serious threat, requiring new defensive approaches. From the previous section, we note that the time taken for researchers to identify the malicious website plays a critical role on the feasibility of the proposed architecture. Currently, the detection of malicious websites takes several days, and if a malicious website stays online for a smaller period of time, it will probably remain undetected. Furthermore, there is a lot of fragmentation on the available website blacklists, allowing users to access websites that were already detected as malicious in other blacklists. We need, therefore, to focus our efforts in developing improved mechanisms to detect and blacklist malicious websites, in a faster and more efficient manner.

Also, the discussed architecture reduces the usefulness of "client-side" honeypots emulating normal end-user systems. Since bots do not possess information regarding the C^2 infrastructure, capturing them is of limited use. Therefore, in order to be able to detect and mitigate this threat, we should switch our focus to "server-sided" honeypots, detecting and analyzing the creation of the malicious websites themselves, instead of bot infections.

Finally, there is a vastly unexplored space concerning heuristics that allow browsers to detect malicious behavior that does not directly target the user's browser. As of now, having javascript code sending thousands of back-to-back connections is not considered malicious by the majority of detection methods, since they focus on the detection of specific attacks (exploit attempts, for example). Heuristics that enable detection of code that creates a large number of connections to different remote destinations, with a potentially large number of those being failed requests, should be put in place. These heuristics would also be useful in the detection of other types of attacks, such as browser-based denial-of-service, where malicious code is also sent to hundreds of legitimate clients, having them send thousands of requests to some specific remote location, thereby effectively creating a distributed denial-of-service attack.

6 Related Work

Botnet detection has been a major research topic in recent years. Lately, researchers have been focusing on detection of the command-and-control traffic, as a way to detect the presence of infected machines, and analyse the botnet infrastructure [24, 7, 17, 6].

Centralized botnets are the most commonly known and studied botnets. Most of these botnets use IRC servers as their centralized control point, having the bots joining in as normal clients. The existence of a centralized control point is the biggest weakness of this kind of infrastructure, since it represents a single point of failure in the architecture. Also, this particular kind of C^2 infrastructure allows researchers to extract information about the botnet, such as its size and the participants's IP addresses.

Several techniques have been found to detect this type of C^2 infrastructure. In [11], the authors proposed several statistical algorithms to detect these botnets, based on their multiple crowd-like behavior. An extension of this work was done in [10], where the authors presented a framework to perform clustering on the monitored C^2 communication and malicious activities, performing a cross-correlation between them to attain the final results. Also, some researchers have studied particular details of the IRC protocol, enabling the detection of this specific kind of botnets [9]. Unfortunately, botnets are also in constant evolution, and, more recently, there was a shift in C^2 architectures towards the use of P2P networks. This new C^2 infrastructure is focused on survivability, and is therefore, considerably harder to take down. However, due to the open and decentralized nature of peer-to-peer protocols, the academic community has also found several ways of detecting and infiltrating them.

In [16], researchers were able to infiltrate the Storm botnet by impersonating proxy peers in the overlay network used for the C^2 infrastructure. They were able to take advantage of this privileged position in the overlay, to rewrite URLs in the spam sent by the bots. Also, there are several results on botnet size estimation that exploit the openness of the C^2 infrastructure overlay to estimate the botnets' size, or to extract the stolen information from the bots [13]. One other way of disrupting the operation of these botnets has been to attack the bootstrap procedure, and tamper with the domain name service (DNS), as bots typically resolve domain names to connect to their C^2 infrastructure. Therefore, by collaborating with the domain registrar, it is possible to change the mapping of a botnet domain to point to a machine controlled by the researchers.

It is our intuition that botmasters will try to camouflage their C^2 infrastructure, and maybe avoid its use altogether, if possible. One such example of camouflage can be found in [22]. Overbot is a P2P C^2 infrastructure proposed by Starnberger et al., that camouflages bots as normal P2P clients, thwarting researchers from distinguishing a compromised host from a honest P2P participant. This solution turns out to be impractical for several reasons, one of them being the fact that it requires the existence of several sensor bots inside the P2P network, possessing the botmaster's private key, and sniffing all control communications that passes trough them. A similary stealthy approach is also described in [20], where the authors design a botnet architecture that makes use of the Skype P2P network as a C^2 infrastructure.

In the literature, there are some references to the principle of no single bot knowing more about the infrastructure than its own local information [8]. This would allow the detection and compromise of an unbounded number of bots, since none of them possesses enough knowledge to allow the compromise or exposure of the full botnet. However, to the best of our knowledge, this is the first work to show that this principle can be implemented in a practical way.

7 Conclusions

Most of the recent approaches to botnet detection and analysis are based upon the vulnerabilities of the botnet's internal C^2 infrastructure. In this paper, we showed that, with the present capabilities of scripting languages like javascript, it is already possible to devise a botnet which does not, in fact, possess an internal C^2 infrastructure, and is, therefore, impervious to such methods of analysis.

An example of such an architecture was discussed, were bots do not possess any information concerning command-and-control mechanisms. These stealth, isolated bots are controlled through honest participants not pertaining to the botnet. This architecture eliminates the possibility of estimation of the botnet size, minimizes the probability of detection of individual bots, and eliminates the possibility of researcher infiltration. The impact on performance of the several system parameters was analyzed, the bounds within which such an architecture may constitute a threat were determined, and a case-study architecture was implemented. We believe that, given the recent focus on detecting the C^2 communication pattern of botnets by the academic community, botmasters will follow a predictable evolutionary pattern, and try to employ this type of architecture to harden their botnets against researchers. As was shown, for some combinations of the relevant parameters, these botnets may constitute serious, effective threats, very difficult to analyze and defuse. An effort is, therefore, needed, if we are to preclude the emergence of this type of botnet.

References

[1] Adobe labs - stratus. http://labs.adobe.com/technologies/stratus, 2010.

[2] The bogon reference. http://www.team-cymru.org/Services/Bogons, Aug. 2010.

[3] Easyxdm - cross-domain messaging. http://easyxdm.net, 2010.

[4] Same origin policy - web security. http://www.w3.org/Security/wiki/Same_Origin_Policy, Jan. 2010.

[5] ABU RAJAB, M., ZARFOSS, J., MONROSE, F., AND TERZIS, A. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2006), ACM, pp. 41–52.

[6] AKIYAMA, M., KAWAMOTO, T., SHIMAMURA, M., YOKOYAMA, T., KADOBAYASHI, Y., AND YAMAGUCHI, S. A proposal of metrics for botnet detection based on its cooperative behavior. In *SAINT-W '07: Proceedings of the 2007 International Symposium on Applications and the Internet Workshops* (Washington, DC, USA, 2007), IEEE Computer Society, p. 82.

[7] CHOI, H., LEE, H., LEE, H., AND KIM, H. Botnet detection by monitoring group activities in dns traffic. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on* (2007).

[8] COOKE, E., JAHANIAN, F., AND MCPHERSON, D. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop* (Berkeley, CA, USA, 2005), USENIX Association, pp. 6–6.

[9] GOEBEL, J., AND HOLZ, T. Rishi: identify bot contaminated hosts by irc nickname evaluation. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (Berkeley, CA, USA, 2007), USENIX Association, pp. 8–8.

[10] GU, G., PERDISCI, R., ZHANG, J., AND LEE, W. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *SS'08: Proceedings of the 17th conference on Security symposium* (Berkeley, CA, USA, 2008), USENIX Association, pp. 139–154.

[11] GU, G., ZHANG, J., AND LEE, W. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)* (February 2008).

[12] HOLZ, T., STEINER, M., DAHL, F., BIERSACK, E., AND FREILING, F. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats* (Berkeley, CA, USA, 2008), USENIX Association, pp. 1–9.

[13] HOLZ, T., STEINER, M., DAHL, F., BIERSACK, E., AND FREILING, F. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats* (Berkeley, CA, USA, 2008), USENIX Association, pp. 1–9.

[14] KAMRA, A., FENG, H., MISRA, V., AND KEROMYTIS, A. The effect of dns delays on worm propagation in an ipv6 internet. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (march 2005), vol. 4, pp. 2405 – 2414 vol. 4.

[15] KANG, B. B., CHAN-TIN, E., LEE, C. P., TYRA, J., KANG, H. J., NUNNERY, C., WADLER, Z., SINCLAIR, G., HOPPER, N., DAGON, D., AND KIM, Y. Towards complete node enumeration in a peer-to-peer botnet. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security* (New York, NY, USA, 2009), ACM, pp. 23–34.

[16] KANICH, C., KREIBICH, C., LEVCHENKO, K., ENRIGHT, B., VOELKER, G. M., PAXSON, V., AND SAVAGE, S. Spamalytics: an empirical analysis of spam marketing conversion. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security* (New York, NY, USA, 2008), ACM, pp. 3–14.

[17] KARASARIDIS, A., REXROAD, B., AND HOEFLIN, D. Wide-scale botnet detection and characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (Berkeley, CA, USA, 2007), USENIX Association, pp. 7–7.

[18] KEROMYTIS, A. D., BELLOVIN, S. M., AND CHESWICK, B. Worm propagation strategies in an ipv6 internet. *USENIX 31* (February 2006), 70–76.

[19] LIKARISH, P., AND JUNG, E. A targeted web crawling for building malicious javascript collection. In *DSMM '09: Proceeding of the ACM first international workshop on Data-intensive software management and mining* (New York, NY, USA, 2009), ACM, pp. 23–26.

[20] NAPPA, A., FATTORI, A., BALDUZZI, M., DELL'AMICO, M., AND CAVALLARO, L. Take a deep breath: a stealthy, resilient and cost-effective botnet using skype. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment* (Berlin, Heidelberg, 2010), DIMVA'10, Springer-Verlag, pp. 81–100.

[21] P. PORRAS, H. S., AND YEGNESWARAN, V. A foray into conficker's logic and rendezvous points. In *LEET'09: Proceedings of the 2nd Usenix Workshop on Large-Scale Exploits and Emergent Threats* (2009), USENIX Association.

[22] STARNBERGER, G., KRUEGEL, C., AND KIRDA, E. Overbot: a botnet protocol based on kademlia. In *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication networks* (New York, NY, USA, 2008), ACM, pp. 1–9.

[23] STONE-GROSS, B., COVA, M., CAVALLARO, L., GILBERT, B., SZYDLOWSKI, M., KEMMERER, R., KRUEGEL, C., AND VIGNA, G. Your botnet is my botnet: analysis of a botnet takeover. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), ACM, pp. 635–647.

[24] STRAYER, W., WALSH, R., LIVADAS, C., AND LAPSLEY, D. Detecting botnets with tight command and control. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on* (14-16 2006), pp. 195 –202.

[25] WANG, P., SPARKS, S., AND MEMBER, C. C. Z. An advanced hybrid peer-to-peer botnet. In *In Proceedings of the First Workshop on Hot Topics in Understanding Botnets* (2007).

[26] WONDRACEK, G., HOLZ, T., PLATZER, C., KIRDA, E., AND KRUEGEL, C. Is the internet for porn? an insight into the online adult industry. In *Proceedings of the Ninth Workshop on the Economics of Information Security (WEIS 2010)* (2010).

[27] XIE, Y., YU, F., ACHAN, K., PANIGRAHY, R., HULTEN, G., AND OSIPKOV, I. Spamming botnets: signatures and characteristics. *SIGCOMM Comput. Commun.* (2008).

[28] YANG, W., SHEN, X.-M., CHANG, G.-R., AND YAO, Y. A p2p-based worm in next gernation network. In *Proceedings of the 2010 Fourth International Conference on Genetic and Evolutionary Computing* (Washington, DC, USA, 2010), ICGEC '10, IEEE Computer Society, pp. 418–421.

[29] YANGUI, X., JIACHUN, Z., XIANGCHUN, L., AND HUANYAN, Q. Worm detection in an ipv6 internet. In *Computational Intelligence and Security, 2009. CIS '09. International Conference on* (dec. 2009), vol. 2, pp. 366 –370.

[30] ZOU, C. C., TOWSLEY, D., AND GONG, W. On the performance of internet worm scanning strategies. *Perform. Eval. 63*, 7 (2006), 700–723.