

Towards Next-Generation Botnets

Ralf Hund Matthias Hamann Thorsten Holz

Laboratory for Dependable Distributed Systems
University of Mannheim, Germany

Abstract

In this paper, we introduce the design of an advanced bot called *Rambot* that is based on the weaknesses we found when tracking a diverse set of botnets over a period of several months. The main features of this bot are peer-to-peer communication, strong cryptography, a credit-point system to build bilateral trust amongst bots, and a proof-of-work scheme to protect against potential attacks. The goal of this work is to increase the understanding of more advanced botnet designs, such that more efficient detection and mitigation systems can be developed in the future.

1 Introduction

A *bot* is a computer program installed on a compromised machine which allows an attacker to execute arbitrary commands on the infected machine. Botnets, i.e., networks of such bots under a common control infrastructure, are one of the main problems in today's Internet since botnets are often used to carry out Distributed Denial-of-Service (DDoS) attacks, send a large amount of spam or phishing mails, and other nefarious purposes [4, 15, 22].

In response to this problem, researchers developed a diverse set of techniques and tools to either detect or mitigate botnets [2, 8, 11, 12, 18]. Unfortunately, most of these techniques focus on botnets with a central server used for Command & Control (C&C): in these scenarios, the bots connect to a central server from which they receive the commands they are supposed to execute. Mitigation of such botnets can be achieved by locating the central server and then shutting it down [8]. However, in the recent months, the first botnets that use a peer-to-peer (P2P) based communication channel appeared [6, 10, 21, 26, 27]. Dealing with such botnets is a challenging task, still some techniques to track these botnets have been proposed [14].

In the near future, we expect the emergence of more advanced botnets that can defeat the current defense

mechanisms. In this paper, we present the design of one such advanced botnet named *Rambot*, a next-generation botnet whose design is based on the lessons we learned when tracking a diverse set of botnets over a period of several months. We address all the weaknesses of current botnets we noticed and construct a botnet that is hard to track and shut down. While some of our design decisions seem to be obvious, e.g., using strong cryptography, (almost) no current botnet uses them. Furthermore, some aspects of *Rambot*'s design are countermeasures against new techniques to track botnets, e.g., proof-of-work schemes to counter crawling of the whole botnet.

One basic design aspect of the botnet developed in this paper is P2P-based communication. While decentralized botnets overcome many of the problems of botnets with a central server, e.g., there is no single point of failure, also several distinct weak points against someone who wants to shut down the network arise. In a central botnet, many security related requirements, such as, for example, authentication of botmaster commands, are guaranteed as long as the central server has not been compromised. While this has been proved to be a reasonable assumption, things are more complicated concerning a decentralized network since each bot carries additional responsibilities. We illustrate these implications throughout the paper and subsequently offer solutions to such problems. In summary, *Rambot* is an example of the threats and challenges we expect in the near future.

The area of botnets faces – similar to other fields – an arms race between botmasters and defenders. To keep up with latest developments, researchers need to continue to improve detection and mitigation methods and investigate new techniques used by botmasters. The goal of our work is to increase the understanding of more advanced botnet designs. We anticipate that this paper ultimately leads to the development of new, sophisticated techniques, which will help to fend off arising threats.

2 Towards Resilient Botnets

2.1 Peer-To-Peer Communication

One main aspect of Rambot’s design is the usage of P2P technology to accommodate for the unique requirements of botnets. This, on the other hand, brings up additional challenges, which need to be addressed during the design phase, e.g., the architecture should be able to effectively handle the specific communication needs of a decentralized botnet and ideally provide a convenient protocol, while still maintaining a certain degree of flexibility. Therefore, the typical communication scheme of a P2P botnet is rather simple compared to the ones of decentralized networks which were designed with different purposes in mind, e.g., efficient file exchange [3, 20, 23]. The main difference is that the bandwidth needs within a botnet are commonly rather small since the botmaster only occasionally sends commands or updates to the bots. Concerning communication, a bot’s main task is to disperse information (e.g., botmaster commands, updates, or spam templates) to a preferably high number of nodes within a short period of time. By all means, the information should arrive at a high fraction of all bots at least once and should not have already become obsolete at the time of its first delivery.

Proprietary P2P module and fallback system.

Contrary to most existing P2P botnets (e.g. Storm Worm), Rambot is not implemented on top of an existing network like Gnutella. Instead, a customized P2P module, which can be conveniently realized by modifying P2P frameworks like JXTA or GUNet, handles all connections to other members of the botnet. This approach allows for advanced defense mechanisms like a system of trust based on credit points and the prevention of crawling through proof-of-work schemes. Both techniques will be explained in greater detail later on.

Two separate peer-lists form the backbone of Rambot’s P2P module: the so-called *regular peer-list* and a *credit point based peer-list*. The former one is common to many existing P2P networks. It simply contains a specified number of peers who are not behind a firewall or router and have a static IP address. The latter peer-list contains potential connection destinations inside the botnet prioritized by credit points. The exact details of the credit point system as well as its use to strengthen Rambot’s reliability are explained in detail in Section 2.3.

A bot that wants to (re)connect to the network performs two simultaneous, but independent tasks. For one thing, it pseudo-randomly selects peers from the

regular peer-list and tries to connect to them. In case it does not succeed, the unavailable peer is dropped from the list and another one is chosen. This process continues until a sufficient number of connections have been established. As it is essential to replace dropped peers in the regular peer-list with available ones, each successful connection establishment is followed by a peer-list exchange between the involved parties. To prevent defenders from crawling the botnet, only a fragment of each bot’s regular peer-list is exchanged. Furthermore, a technique called proof-of-work will be introduced in Section 2.4 to prevent mass-requests from a defender. However, up to this point, Sybil or similar attacks [7, 24, 25] would still be possible as a valid bot’s peer list could be filled with malicious peers simulated by a defender. To fend this off, a reconnecting bot uses the credit point based peer-list to establish additional connections to bots which have proved their reliability and trustworthiness in the past.

To cover the case that both peer-lists do not contain any available peers (e.g., due to the botnet being very small or a bot reconnecting after a long offline period), Rambot is equipped with a fallback system: the bot which is cut off uses an existing P2P network to call for help. Gnutella (or any other unstructured P2P network) is perfectly suited for this purpose as it permits to broadcast messages throughout the network. The botmaster has to provide a small number of backup servers who join the regular Gnutella network and listen for requests from bots asking for a valid peer-list. Those servers can alter arbitrarily (e.g., changing IP addresses or locations) since the servers use public-key cryptography as explained in Section 2.2 to prove their authenticity. This also prevents malicious backup servers from flooding cut off bots with invalid peer-lists. As defenders might try to misuse the backup system to gain an overview of the botnet, the previously mentioned proof-of-work protocol is employed, which forces bots who ask for a current peer-list to complete a CPU-intensive task first. Moreover, as only a small number of available bots is needed for reconnection purposes, the botmaster’s backup servers should supply very few peer addresses per request. This design offers a high degree of stealthiness since bots join an existing P2P net like Gnutella only in exceptional circumstances. Thus, they are protected by the defense techniques of Rambot’s proprietary P2P module most of the time. Later on, we will discuss these mechanisms in full detail.

Communication structure. Concerning the underlying communication structure, we use a message-based system. That is, bots exchange and synchronize their current state on the basis of P2P messages, which we divide into two different classes hereafter: *push-*

based and *pull-based* messages. The former typically consist of botmaster instructions, for instance DDoS-, spamming-, exploit-commands and the like, sharing the characteristic of being comparatively short in length. The latter class comprises large contiguous data such as, for example, program updates and spam templates. This distinction allows separate rules of propagation to be applied to instances of the corresponding classes. Push-based messages are dispersed by the means of flooding broadcast, that is, upon receiving a valid message of such a type from a neighbor, a bot immediately broadcasts it to all connected bots within his peer-list. The emerging problem of possible massive retransmissions and consequential network overload can be solved by allowing each message to be broadcasted only once. Therefore, each bot maintains an internal list of already transmitted push-messages, ignoring afterwards delivered messages which are already contained in it. The advantage of this approach lies in the consequent maximum rate of propagation. We also exploit this method to build a trust system as we explain in Section 2.3.

In contrast, pull-based message transmissions need to be requested by the opposite party in the first place. It should be noted, however, that a pull-based transmission is usually the consequence of a prior push-based delivered notification. For instance, the existence of a new program update is initially announced by a push-message, indicating its availability as well as providing further information such as the version number, binary hash values and the like. Subsequently, an update transmission may be requested.

Finally, the constant need for synchronization has to be taken into account, since freshly participating bots must be synchronized to the current state of information. We achieve this by means of state exchange during initialization. That is, upon successful connection establishment, both parties exchange the unique identification number of their most recent push-message. Consequent lack of synchronization emerging from this procedure is subsequently overcome by exceptional transmission of the corresponding missing push-messages.

2.2 Strong Cryptography

Tamper-proof command and update scheme. A vital aspect of botnet administration is the *authenticity* and *integrity* of commands: a bot should only accept commands issued by the botmaster. In current botnets, however, we observed that the botmasters commonly use only a very weak form of authenticity, e.g., by using a simple password scheme before sending the actual command. Even if the botnets use stronger authentication schemes, these can typically be broken,

e.g., Storm Worm uses a 64 bit RSA implementation which can be defeated. In centralized IRC botnets, this lack of authenticity could for example be overcome by patching the IRC server used for command distribution in such a way that only the botmaster can send messages in the designated channel. However, when dealing with a decentralized network of equal peers, a botmaster needs to ensure that no hostile parties like defenders or other botnet groups can poison the botnet by injecting malicious commands.

Asymmetric cryptography offers a simple, yet effective way to do this: before releasing a bot in the wild, the botmaster creates a public/private pair of cryptographic keys of which the former one is hardcoded into the bot's binary. Doing so enables the botmaster to securely sign any commands or files using his private key. All peers in the botnet are able to verify the commands employing the hardcoded public key, but given a reasonable key length (e.g. 2048 bits for RSA), no defender will manage to forge the signature.

Rent a botnet. With the help of asymmetric cryptography, a botmaster can take on the role of a trusted certificate authority, which provides an efficient way to rent the botnet to others in parts or as a whole, for a variable amount of time, and for certain purposes. Let us assume some company wants to use the botnet to promote their services using spam mails. The botmaster can sign a certificate for the lessee with his private key in exchange for a certain amount of money. This certificate contains the lessee's public key, a timeframe of validity, and a list of allowed command classes (e.g., spam distribution or DDoS).

From this point on, the renting party can inject its own self-signed commands (bundled with the purchased certificate) into the network via an arbitrary peer. Any bot which receives such a message first verifies the included certificate using the hardcoded public key of the botmaster. In case of validity, it then verifies the distributed command using the lessee's public key obtained from the certificate. If the command belongs to a command class specified in the validated certificate and the timeframe has not yet passed, the command is scheduled or executed and the message bundle, consisting of the signed command and the issuer's certificate, is forwarded to all connected bots.

To protect against malicious lessees, it is advisable to implement a blacklist containing all invalidated public keys. This blacklist is saved on each bot's computer and only the botmaster may add or remove public keys using his private key to sign the order. Thus, all certificates which belong to an attacker can be revoked.

However, such a blacklist is of little use against attacks which require only a short timeframe to be car-

ried out successfully. For example, a malicious lessee could buy a botnet certificate for spam distribution and misuse it by ordering all bots to send an e-mail to a specified address, thereby revealing their IP address or other sensitive data. In effect, an attacker could conveniently obtain valuable information about a botnet's size as well as its overall structure. Therefore, renting a botnet should be considered as an option which has to be used with caution by a botmaster.

Traffic obfuscation. Thus far we have discussed exclusively threats to a botnet from within, that is, by malicious clients. Because of legal reasons, defenders might have to employ a completely different approach and try to fight botnets without interacting with the bots in any way. The most intuitive way of doing so is by blocking a botnet's communication. This can either be achieved by filtering traffic to or from special ports or by recognizing malicious packets based on certain properties of the botnet's communication protocol. The former danger can be defeated by encapsulating the botnet's traffic into well-known application layer protocols like DNS, HTTP or FTP. Especially encapsulating all botnet communication within HTTP requests will lead to a rather stealthy communication since detecting such a communication channel is not easy given the fact that most communication in today's Internet uses TCP port 80. The latter danger is circumvented by encrypting all botnet-specific data previous to transmission.

For reasons of efficiency, the use of a symmetric algorithm like AES is advisable. Securely implementing such an encryption scheme would require that a secret key had already been exchanged via the Internet between the communication partners by the time of their first encounter. As this is obviously impractical, Wang et al. [29] proposed that the initiator of the connection should learn about the symmetric key at the same time as he is told the destination's address by some other bot. Two coherent problems arise out of such an implementation. First, all bots would have to use the same key when connecting to a certain peer as there is no trusted third party in a P2P botnet which could securely issue individual keys. Second, the key would have to be static as any change might prevent a substantial number of peers from reconnecting in the future. Depending on the number of bots connected to a certain peer, these restrictions would crucially weaken traffic obfuscation and data confidentiality. Especially in a botnet which can be easily crawled (e.g. the Storm Worm botnet) non-individual, static symmetric keys would be of practically no use when facing dedicated defenders as they could be harvested with little effort.

Therefore, Rambot uses Diffie-Hellman key exchange to provide symmetric encryption with individual session keys. This way, there is no need for the communication partners to know anything about each other in the run-up and eavesdropping the conversation is virtually impossible for any third party. To prevent a man-in-the-middle attack in such a scenario, each bot should calculate its own public/private keypair upon installation and exchange not only IP addresses but tuples of (IP address, public key) during peer list updates. Based on this information, the initiator of a connection can perform a challenge-response authentication of the server after having established a secure channel via Diffie-Hellman key exchange and symmetric cryptography.

The authentication of bots using asymmetric cryptography also plays an important role in the following paragraph, where we introduce a basic concept of trust relying on credit points to improve botnet robustness.

2.3 Credit Point System

The need for trust. In the previous section, we discussed several cryptographic methods to prevent interfering with a botnet's command distribution mechanism as well as the filtering of its network traffic. When faced with a potent attacker (e.g., another botmaster with many zombie computers at her disposal), several new threats to a botnet's reliability arise. One grave danger is the possibility of a DoS attack based on the P2P network's bootstrapping process: an attacker could try to flood a bot's peer-list with fake clients under her control. This would effectively prevent the attacked bot from reconnecting to the network in the future. If carefully planned and performed at a large scale, such an attack could ultimately lead to the separation of a botnet into relatively small fragments, thereby drastically reducing its capabilities. Accepting only a limited number of contacts per peer exchange is only part of the solution, as we have to assume an attacker who could easily create a significant number of attacking fake clients via emulation. Thus, the attacker against the botnet could control an almost arbitrarily high percentage of clients in a bot's neighborhood.

Cryptographic protection. Sustaining a botnet's robustness in such a challenging situation raises the need for more sophisticated defense mechanisms. Rambot thus includes a trust system which relies on credit points as its metric. Trust is strictly bilateral in our model, that is, we do not create a web of trust by having clients exchange their information about each other. Instead, credit points are confined to pairs of

clients. A major advantage of this design is its straightforward implementability and low error-proneness.

The tamper-proof linkage between credit points and peer identities is guaranteed through the use of asymmetric cryptography. This interconnection allows for the implementation of a “safety net” in the form of a blacklist: should an attacker manage to acquire a menacing amount of points at a substantial number of clients, the botmaster can stop this attacker by issuing an order to add the corresponding public key to the local blacklist of each bot. Thus, the attacker will lose all his credit points throughout the botnet which are tied to a certain public key. But as an attacker might try to evade this situation by using a different public key for each relationship with a valid bot, additional measures have to be taken into account to prevent successful credit point fraud, which we introduce later on.

Acquiring and managing credit points. The basic means of earning credit points is forwarding of commands and updates to other peers. Hence, any malicious client who seeks to accumulate a large number of credit points to perform the flooding attack is forced to contribute substantially to the overall botnet communication process. This might prove to be a big hurdle for organizations like CERTs as they are bound to certain legal obligations. An attacker could bypass this problem through renting a botnet as previously described and issuing a lot of benign commands (e.g., sending spam to non-existent addresses) to give her malicious bots the opportunity to earn credit points safely. In consequence, only forwarding of commands which were signed using the botmaster’s private key are rewarded. To prevent scamming by relaying outdated and therefore worthless commands or by focusing solely on command distribution, a bot only increases the score of all peers that forward a validly signed command whose timeframe has not yet passed. As the credit point based peer-list is meant to reflect the current availability of bots in the network, all peers who have not relayed a command once its timeframe ends lose some of their credit points.

Using credit points. As mentioned earlier, Rambot’s P2P module relies on two separate peer-lists when reconnecting to the botnet. While one of them serves as a source for pseudo-random selection of peers to connect to, the other list contains (public key, IP address, credit point sum) tuples of all peers which have been seen by the reconnecting bot within a reasonable time span (e.g., the last month). This list is ordered by credit points, which gives bots that have been a valuable contact in the past priority as a connection des-

tinuation. Since a high number of credit points closely correlates with a high degree of availability in the past (depending on the chosen timeframe), the trial-and-error process of selecting bots from the credit point based peer-list and connecting to them is feasible. It should be noted that bots are only deleted from this peer-list if they have not been seen for several weeks or months. A connection error which happens before that point in time does not result in deletion as the destination may be unavailable only temporarily. If a bot is permanently unavailable (e.g., due to the subsequent installation of a firewall or antivirus software), it will eventually exceed the time-limit and be removed by the P2P module’s garbage collector. In order to avoid congestion in a peer that has gained a substantial amount of credit points, there is an upper limit of connections that a bot can hold at a time. When this limit has been reached and another bot wants to establish a new connection, this request is refused.

The credit point based peer-list guarantees that even in a situation where the regular peer-list contains only malicious bots due to a flooding attack, a reconnecting bot will still know a sufficient number of benign botnet members. On the other hand, choosing bots in a pseudo-random manner from the regular peer-list protects against the effects of credit point fraud. Taking over both lists simultaneously at a substantial number of bots would require enormous bandwidth (to perform the flooding attack) as well as processing power. The latter requirement is explained in the following section.

2.4 Proof-of-Work

Extending the credit point system. Given enough bandwidth to earn trust in the run-up to a DoS attack, up to this point, it would have been perfectly possible to attack our P2P botnet using a significant amount of emulated bots. As a botnet’s usual field of application is not distributed computing, but mass mailing of spam or performing DoS attacks, the hardware requirements for its members are pretty low. Thus, an attacker could easily assign a range of several thousand IP addresses to only a few real computers, thereby infiltrating the network with a large number of fake clients. Furthermore, these malicious bots could earn a considerable amount of credit points if their software had been optimized to spend the available bandwidth on command distribution rather than executing orders issued by the botmaster. This danger can be tackled by confronting a potential attacker with the substantial computation power of a common botnet, which results from the fact that contemporary computers have a lot of idle CPU cycles. In the following, we show how to utilize these circumstances to provide

real bots with a significant advantage over malicious (simulated) ones when acquiring credit points.

Protocol description. Proof-of-work systems are based on tasks which are moderately hard to solve for a client, but easy to verify on the server side [16, 19]. A perfect example is the partial inversion of hashes: as cryptographic hash functions are preimage resistant by definition, some information about the input (e.g., its size as well as some publicly known, but random, parts to prevent the use of rainbow tables) has to be given to the client beforehand [1]. There are two classes of proof-of-work protocols: challenge-response and solution-verification. We focus on the former as it is particularly suitable for a P2P botnet. An idle bot (hereinafter referred to as *Alice*) who wants to earn some other bot's (called *Bob*) trust asks for a challenge. Bob constructs a random one and presents it to Alice. After a reasonable amount of time (e.g., 15 minutes on a common personal computer) Alice succeeds in solving the problem and transmits the solution to Bob, who verifies it and awards Alice an equivalent in credit points. Again, both the construction and the verification of the task may not be demanding as a Bob would otherwise become vulnerable to DoS attacks through flooding with requests or wrong answers. Since the whole procedure should be as unobtrusive as possible on the client side, it is advisable to have the bot software only take advantage of an idle CPU in case the screensaver is active or if there has not been any user input in a certain amount of time.

Prevention of crawling. The proof-of-work system is not limited to earning trust in the form of credit points. Current P2P botnets like the Storm Worm network can be crawled which provides potential attackers with valuable information to base their actions on [14]. Prepending a proof-of-work to the bots' peer exchange renders almost any form of crawling unfeasible. Even given easy tasks (e.g., equivalent to five minutes of computation on a standard home computer), the whole network would have completely changed its structure through dial-up caused IP address changes and general diurnal effects before an attacker has gathered any significant amount of information about the overall network. There is a disadvantage, though, in that if a client does not stay online long enough to complete a proof-of-work task, it can not join the botnet and there will not receive any updates to its peer-list. This might ultimately lead to the permanent disconnection of a bot from the network if it repeatedly fails to solve a proof-of-work task upon joining the botnet. On the

other hand, such bots would not be of considerable use for the botmaster anyhow.

3 Botnet Tracking for Rambot

Up to now, we have introduced measures aimed at ensuring the botnet's security against possible attackers, who pursue its mitigation, takeover and/or possible destruction. In order to provide a rather comprehensive view on the actual vulnerability, this section summarizes weaknesses and assesses individual threats along with their countermeasures.

Assuming that the depicted measures are implemented in a correct manner, the general vulnerability of the botnet is limited to purposely induced starvation, achieved by peer-list poisoning. In this scenario, an attacker attempts to infiltrate the botnet with a massive amount of fake clients. Having taken over the entire peer-list of a specific bot, the fake clients subsequently refuse to forward any botmaster commands, therefore effectively starving this bot. However, due the presence of the credit point system, fake clients are inevitably forced to forward signed commands for at least a certain amount of time, meanwhile supporting the botnet structure. Otherwise, the fake client would be incapable of taking over the point-based portion of the bootstrapping list. And since the credit point system runs with timeout values, any attempts at optimizing the forwarding are rendered impossible. The attacker thus has to constantly contribute to the botnet's communication scheme. The bottom-line is that an attacker is bound to play his steady role in the botnet-game for a long time, possibly exposing him to legal issues as well, since he unavoidably helps in propagating instructions appointed for a criminal act.

If we assume that the attacker has managed to bypass the depicted obstacles and thus taken over the point-based portion of the peer-list, he would still be exposed to an inevitable uncertainty in choosing the right moment in order to cease forwarding messages to a specific bot. The reason for that lies in the simple fact that each bot connects to random peer-list entries for a randomly chosen number of times as part of the bootstrapping process. Along with the fact that the bot keeps these connections secret from the outer world, the attacker never knows when the peer-list of the bot is completely taken over by his fake clients. However, if the moment of interruption is improperly chosen, i.e., there is still at least one real bot in the vicinity, the credit point system disfavors fake clients with each delivered command due to the subsequent loss of points, eventually bringing back the real bots into the game. Furthermore, once the bots restarts, a completely new

vicinity will be chosen randomly during the bootstrapping process. As a consequence, the only possibility left is to massively infiltrate the botnet. That, however, is prevented by the proof-of-work mechanism.

4 Related Work

We are not the first to present designs for advanced botnets. Compared to the proposed design by Wang et al. [29], our main advantage is that the botnet is more resilient in the sense that a defender can not easily get an overview of all infected machines within the botnet: our design ensures that an attacker can not exploit the P2P structure to crawl the network. Furthermore, the credit point system forces a defender to constantly contribute to the communication within the botnet in order to acquire credit points. This is an extended version of the honeypot-aware botnet construction idea introduced by Zou et al. [30].

The botnet design by Vogt et al. [28] has the main limitation that the botnet can not grow: once the botnet is established, no new bots can easily join the network. This is a severe limitation since it restricts the botnet to a fixed population, which will degrade over time since infected machines are cleaned up. Dagon et al. presented several techniques for botnet communication and studied their resilience [5]. However, no implementation details are provided. Our botnet design is an instance of random graph botnets.

5 Conclusion and Future Work

In this paper, we presented the design of an advanced botnet that takes into account the lessons we learned when tracking a diverse set of botnets over a time period of several months. The core building blocks of our botnet are:

- Using strong cryptography to safeguard the communication channel within the botnet and guarantee authenticity and integrity of the botmaster's commands.
- A credit-point system to build bilateral trust amongst the bots.
- A proof-of-work scheme to protect against attacks targeting the P2P based network structure.

Besides the presented design decisions, there are more techniques that need to be taken into account when designing next-generation botnet detection systems. For example, the communication pattern used by

NNTP [17] is much more suitable for anonymous communication than DHT-style routing as used by Storm Worm. This communication pattern is the core of fault-tolerant information dissemination like in Golding's *weak consistency replication* [9] or the reliable broadcast algorithms by Hadzilacos and Toueg [13]. Botnets based on these techniques or the ideas introduced in this paper will be much harder to mitigate than current botnets.

In the future, we thus need to develop botnet detection systems that keep such advanced botnet designs into account. Today's botnets can be defeated by mitigating the central botnet C&C server or exploiting weaknesses in the P2P protocol of the botnet. In the future, such attacks may not work since botnets like the one presented in this paper are immune against all botnet tracking methods proposed so far. But one fundamental property of bots will presumably also remain in the future: the attackers abuse the compromised machine to execute illicit tasks like sending of spam or performing Distributed Denial of Service attacks. Botnet detection system that focus on the *effects* of a compromise might thus be able to also detect future botnets. Instead of focussing on the detection of the C&C channel, it may thus be promising to focus on behavior-based detection.

References

- [1] A. Back. HashCash, 1997. <http://hashcash.org/>.
- [2] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of Second Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'06)*, pages 43–48, July 2006.
- [3] BitTorrent. <http://www.bittorrent.com/>.
- [4] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*, June 2005.
- [5] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A taxonomy of botnet structures. In *Proceedings of 23rd Annual Computer Security Applications Conference (ACSAC'07)*, pages 325–339, 2007.
- [6] D. Dittrich and S. Dietrich. Command and Control Structures in Malware. *login.*, 32(6), 2007.
- [7] J. R. Douceur. The Sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, March 2002.

- [8] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proc. of 10th European Symposium On Research In Computer Security (ESORICS'05)*, 2005.
- [9] R. A. Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California at Santa Cruz, Dec. 1992. UCSC-CRL-92-52.
- [10] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proc. of Hot Topics in Understanding Botnets (HotBots'07)*, 2007.
- [11] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. of the 16th USENIX Security Symposium*, 2006.
- [12] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.
- [13] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5. Addison-Wesley, second edition, 1993.
- [14] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08)*, 2008.
- [15] HoneyNet Project. Know your Enemy: Tracking Botnets, March 2005. <http://www.honeynet.org/papers/bots>.
- [16] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks*, 1999.
- [17] B. Kantor and P. Lapsley. Network news transfer protocol. Internet RFC 977, available at <http://www.faqs.org/rfcs/rfc977.html>, Feb. 1986.
- [18] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of Hot Topics in Understanding Botnets (HotBots'07)*, April 2007.
- [19] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proc. of the 12th USENIX Security Symposium*, 2003.
- [20] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer information system based on the XOR metric. In *Proceedings of the First Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.
- [21] 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.
- [22] 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.
- [23] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, 2001.
- [24] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. Infocom 06*, Apr. 2006.
- [25] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting KAD: Possible Uses and Misuses. *Computer Communication Review*, 37(5), Oct 2007.
- [26] J. Stewart. Storm worm DDoS attack. Internet: <http://www.secureworks.com/research/threats/storm-worm>, 2007.
- [27] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache Trojans: P2P Is Here. *login.*, 32(6), 2007.
- [28] R. Vogt, J. Aycock, and M. Jacobson. Army of botnets. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS'07)*, 2007.
- [29] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets (HotBots)*, pages 2–2, 2007.
- [30] C. C. Zou and R. Cunningham. HoneyPot-aware advanced botnet construction and maintenance. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2006.