# SoK: P2PWNED — Modeling and Evaluating the Resilience of Peer-to-Peer Botnets

Christian Rossow*‡, Dennis Andriesse‡, Tillmann Werner¶,
Brett Stone-Gross†, Daniel Plohmann§, Christian J. Dietrich*, Herbert Bos‡
* *Institute for Internet Security, Gelsenkirchen, Germany*
{*rossow,dietrich*}*@internet-sicherheit.de*
‡ *VU University Amsterdam, The Netherlands*
{*d.a.andriesse,h.j.bos*}*@vu.nl*
§ *Fraunhofer FKIE, Bonn, Germany, daniel.plohmann@fkie.fraunhofer.de*
† *Dell SecureWorks, bstonegross@secureworks.com*
¶ *CrowdStrike, Inc.*

***Abstract*—Centralized botnets are easy targets for takedown efforts by computer security researchers and law enforcement. Thus, botnet controllers have sought new ways to harden the infrastructures of their botnets. In order to meet this objective, some botnet operators have (re)designed their botnets to use Peer-to-Peer (P2P) infrastructures. Many P2P botnets are far more resilient to takedown attempts than centralized botnets, because they have no single points of failure. However, P2P botnets are subject to unique classes of attacks, such as node enumeration and poisoning. In this paper, we introduce a formal graph model to capture the intrinsic properties and fundamental vulnerabilities of P2P botnets. We apply our model to current P2P botnets to assess their resilience against attacks. We provide assessments on the sizes of all eleven active P2P botnets, showing that some P2P botnet families contain over a million bots. In addition, we have prototyped several mitigation strategies to measure the resilience of existing P2P botnets. We believe that the results from our analysis can be used to assist security researchers in evaluating mitigation strategies against current and future P2P botnets.**

## I. Introduction

Criminals use botnets for a myriad of malicious activities, including denial-of-service attacks, spam, and banking fraud. The most common type of architecture for existing botnets is based around a central *Command-and-Control* (C&C) server. Consequently, these C&C servers have received an increasing amount of attention from security researchers and law enforcement for takedown attempts [24, 6]. In response, botnet controllers (botmasters) have designed and implemented new architectures to make their botnets more resilient. Some botnets use fast-flux DNS to assign addresses from a large pool of IPs belonging to compromised systems to C&C domains [16]. In addition, attackers have implemented domain generation algorithms (DGA) to dynamically generate domain names used for C&C (e.g., depending on seed values such as the current date/time and Twitter trends) [1]. A more radical and increasingly popular way to increase botnet resilience is to organize the botnet as a *Peer-to-Peer* (P2P) network. In a P2P botnet, bots connect to other bots to exchange C&C traffic, eliminating the need for centralized servers. As a result, P2P botnets cannot be disrupted using the traditional approach of attacking critical centralized infrastructure.

Even though active P2P botnets such as the Zeus P2P variant, Sality, ZeroAccess and Kelihos have survived in the wild for as long as five years, relatively little is known about the sizes of these botnets. It is difficult to estimate a P2P botnet's size, for a number of reasons. P2P botnets often use custom protocols, so that researchers must first reverse engineer the protocol and encryption before they can track the botnet's population. In addition, approximations based on IP addresses alone have been shown to be inaccurate unless care is taken to account for IP address churn [24].

Another significant problem is that there is currently no systematic way to analyze the resilience of P2P botnets against takedown attempts. Previous work has shown significant weaknesses in some P2P botnets, such as Storm [9] and Waledac [23], but these studies focus on specific design flaws that are unique to these P2P botnets.

In this paper, we present a graph-theoretical model of P2P botnets that aids in analyzing the resilience of these botnets. The proposed model captures the fundamental characteristics of P2P botnets, covering intrinsic properties of their self-organizing nature. We extend our model with mitigation strategies that are generally applicable to any P2P botnet. Our model highlights two resilience aspects:

1) *Intelligence gathering resilience*: We model intelligence gathering (also referred to as reconnaissance) methods and evaluate the resilience of current P2P botnets to these methods. That is, we evaluate to what extent the P2P botnets are able to deter malware analysts from enumerating the bots in the network.
2) *Disruption resilience*: We formalize attacks that can be used to disrupt P2P botnets. Examples of such attacks include *sinkholing*, where all bots are redirected to an attacker-controlled machine called a *sinkhole*, and *partitioning*, which aims to split a botnet into unusable sub-networks.

We apply our methods to real-world P2P botnets to evaluate their resilience against the modeled attacks.

We use the reconnaissance methods described in our model to gather intelligence about real-world P2P botnets. In particular, we deploy two P2P node enumeration techniques, namely *crawling* and *sensor injection*, to measure the population sizes of current P2P botnets. Our experiments show that some of these P2P botnets contain in the order of a million infected systems, illustrating the magnitude of the P2P botnet threat. Interestingly, we found significant differences in the number of bots found by our crawlers compared to our sensors. As we will show, crawling may underestimate a botnet's size by two orders of magnitude. In addition, we identify an inherent limitation to crawling, namely that crawlers cannot verify the authenticity of the 60–87% of enumerated nodes which are behind NATs, proxies, or firewalls.

We also apply the disruption attacks defined in our model to real-world P2P botnets. We show that some current P2P botnets are quite resilient to disruption—a clear improvement over early botnets like Storm and Waledac. For instance, Sality employs a peer reputation scheme which significantly complicates attacks. As another example, the Zeus P2P botnet uses automatic blacklisting of sinkholing servers that communicate too aggressively. Moreover, as we will show, several P2P botnets are able to repel initially successful attacks against their P2P layers over the long term through the use of backup C&C channels. Additionally, we investigate to what extent P2P botnets are susceptible to attacks like command injection (as opposed to attacks against the botnet infrastructure).

In summary, our contributions are:

1) We propose a formal graph model for P2P botnets and define attack methods that are applicable to any P2P botnet. This model can be used to assist malware analysts in P2P botnet mitigation efforts.
2) Based on our model, we implement intelligence gathering attacks against current P2P botnets. We evaluate and compare the accuracy of two P2P botnet enumeration techniques, namely crawling and sensor injection. We provide lower-bound approximations on the populations of all current P2P botnets, and show that crawling results for these botnets are limited in accuracy.
3) We evaluate the resilience of all active P2P botnets by prototyping our formalized attacks. We find that some P2P botnets are susceptible to disruption, while others have a considerably more robust design that makes takedown efforts more challenging.

The remainder of this paper is structured as follows. Section II gives an overview of the P2P botnet landscape. In Section III, we propose a model for P2P botnets, covering intrinsic characteristics of P2P botnets. Section IV expands our model by formalizing intelligence gathering and disruption attacks against P2P botnets. In Section V, we describe new insights into P2P botnet node enumeration by comparing results from crawling and sensor injection. Section VI provides an analysis of the P2P resilience of active and previously active botnets. Section VII discusses ethical issues and the future of mitigation efforts against P2P botnets. Finally, we outline related work in Section VIII and summarize our work in Section IX.

## II. Overview of P2P Botnets

This section provides an overview of the most important P2P botnet families that emerged between the beginning of 2007 and the middle of 2012. Our overview is based on insights from the dynamic malware analysis platform SANDNET [19], combined with reverse engineering, first-hand takedown experience and technical reports from malware analysis companies. We restrict ourselves to botnets which use a P2P channel as their primary means of communication. We do not consider Conficker.C [17] or TDL4 [20], as these botnets only use their P2P components as backup channels. In the remainder of this paper, we distinguish between *botnet families*, *botnet variants* and *botnets*. We use the term *botnet family* to denote a specific strain of a botnet. The term *botnet variant* is used to denote a variant within a botnet family. Finally, we use the term *botnet* to refer to a coherent collection of hosts infected with a specific botnet variant. Some botnet variants contain several disjoint botnets.

### A. P2P Botnet Characteristics

In this paper, we closely analyze twelve P2P botnet variants. Figure 1 shows each of the P2P botnet variants that are still active as of November 2012. As shown in the figure, four P2P botnet families are currently active in the wild. Each of the studied families, except Zeus, consists of several major versions. Finally, for each major version, one or more disjoint botnets exist. Botnets which have already been successfully sinkholed are shown with a dark background in the figure.

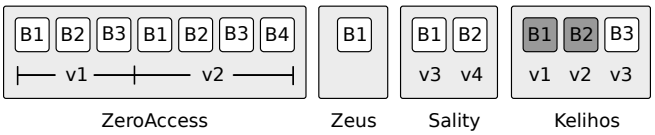

Figure 1: Active P2P botnet families, their variants, and instances.

Figure 2 shows the lifespans of the botnet variants we study in detail, as well as the most important P2P botnets which are no longer active. We see that the minimum botnet family lifetime encountered so far is eight months (Miner), while Sality has been active five years. The large differences shown in P2P botnet lifespans can partly be explained by the fact that some botnet families are more resilient than others. However, the degree of resilience is not a sufficient metric to explain the lifespans of all P2P botnets. For instance, we show in Section VI that the ZeroAccess botnet family, one

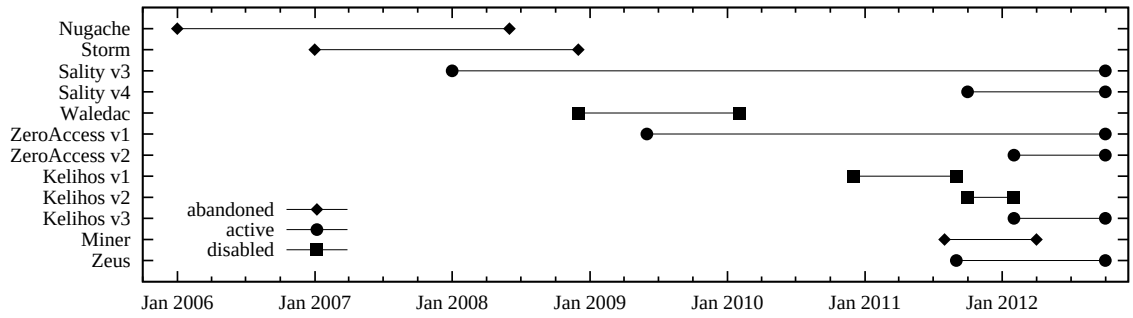


Figure 2: Lifespans of P2P botnet variants.

of the longest surviving P2P botnet families, is in principle relatively straightforward to sinkhole.

Table I shows the communication protocol, message propagation method, communication direction, C&C architecture, and the purpose of each P2P botnet. Note that all recent P2P botnets have *unstructured* P2P protocols, that is, they use message gossiping to propagate information. To date, most academic work on P2P botnets has focused on *structured* architectures [22, 31, 32, 10]. Bots in structured P2P botnets typically maintain a distributed hash table that is used to store and route commands. Unstructured networks are not susceptible to many of the mitigation strategies for structured networks discussed in the literature. Motivated by the large number of unstructured P2P botnets that are used in the wild today, we propose attacks that are generally applicable to structured *and* unstructured P2P botnets.

The P2P botnets listed in Table I are used for a variety of purposes, including malware distribution, spam, credential theft, and Distributed Denial-of-Service (DDoS) attacks. The table also shows the C&C architectures of all P2P botnets. Nugache, Sality and ZeroAccess are purely P2P based. The other botnets rely on hybrid architectures, which incorporate centralized servers, for instance to collect stolen data. Unfortunately, shutting down these centralized components usually has a minimal effect, as the P2P layer can easily be used to redirect bots to alternative servers. Thus, we focus specifically on the resilience of the P2P layer itself.

| Family | Protocol | Prop. | Dir. | C&C | Purpose |
|---|---|---|---|---|---|
| Kelihos | custom | gossip | pull | hybrid | C,D,M,N,**S** |
| Miner | custom | gossip | pull | hybrid | D,**M**,P |
| Nugache | custom | gossip | push | P2P | D,**T** |
| Sality | custom | gossip | pull | P2P | D,N,**P** |
| Storm | Overnet[1] | routing | pull | hybrid | D,**S**,T |
| Waledac | custom | gossip | pull | hybrid | D,**S**,T |
| ZeroAccess | custom | gossip | pull[2] | P2P | **P** |
| Zeus | custom | gossip | both | hybrid | D,P,**T** |

Table I: Overview of P2P botnet families showing their protocol, message propagation method, communication direction, C&C architecture, and purpose. The main purpose is highlighted in bold. C = Click Fraud, D = DDoS, M = Bitcoin Mining, N = Network Services, P = Pay-Per-Install, S = Spam, T = Credential Theft.

### B. Botnet Descriptions

In this section, we describe the P2P botnets that we study in this paper, ordered by the date of their introduction.

*1) Nugache:* Nugache was a P2P botnet based on a custom protocol, which first appeared in the beginning of 2006 [25]. The principle use of Nugache was theft of financial credentials via keystroke logging. Earlier versions used IRC for C&C, but over time the C&C infrastructure was migrated to a P2P-based protocol. The Nugache botnet was one of the first to use strong cryptography to protect its communication. Commands were signed with a 4096-bit RSA key to prevent unauthorized control. The inter-peer communication was encrypted using individually negotiated session keys derived using a hybrid RSA/Rijndael scheme.

*2) Storm:* Storm (a.k.a. Peacomm) was a structured P2P botnet based on Overnet, a Kademlia implementation. Storm appeared in the wild in January 2007. The first version of Storm was built upon an existing Overnet network used for file sharing, which the Storm bots shared with benign clients. Storm bots retrieved commands by using a time-based algorithm to compute the IDs under which new commands would be published by the botmaster, and then searching for these IDs in the DHT. Holz et al. [9] showed that in principle Storm could be mitigated by overwriting the command IDs in the DHT.

*3) Sality P2P:* The P2P version of Sality first appeared in early 2008 and is a variant of the centralized Sality malware downloader. Sality uses a pull-based unstructured P2P network to spread URLs where payloads are to be downloaded. Peers regularly contact their neighbors to exchange new URLs. We distinguish two disjoint Sality botnets, denoted as version three and version four. Both networks share the same P2P protocol, but version four of Sality fixes a critical vulnerability in the file downloading mechanism [7].

*4) Waledac:* Waledac originated in December 2008, and is assumed to be the successor of Storm [23]. The Waledac network used a tiered infrastructure, with an upper layer of centralized nodes providing commands and spam templates

[1] A later added XOR encryption separated the Storm botnet from Overnet.
[2] Bots can trigger pulls at other peers, which are effectively pushes.

to a middle layer of router nodes. Nodes at the router layer were responsible for relaying messages to a lower network layer consisting of regular P2P worker bots. Bots at the worker layer formed the majority of the nodes in the Waledac P2P network, and were connected via a pull-based unstructured P2P network to the router layer.

*5) Kelihos:* Kelihos (a.k.a. Hlux) is an unstructured P2P botnet mainly used for spamming and ID theft [3]. Kelihos first appeared in late 2010, and is assumed to be the successor of Waledac. Researchers successfully sinkholed the first two variants of Kelihos using peer list manipulation attacks. The two sinkholing attacks took place in September 2011 and March 2012, respectively. A subsequent third Kelihos variant is still operational, and is similar in architecture to the first two Kelihos variants [29].

*6) ZeroAccess:* ZeroAccess (a.k.a. Sirefef) is a malware downloader which uses an unstructured P2P architecture. It exists in two variants and is organized into seven disjoint networks. The two variants, ZeroAccess v1/v2, appeared in mid 2009 and early 2012, respectively. ZeroAccess bots regularly query their neighbors for new malware payloads. Lists of new peers are pushed to all bots using a broadcast mechanism [15, 30].

*7) Miner:* Miner was an unstructured P2P botnet which appeared in August 2011, and included facilities for generating ("mining") a digital currency called Bitcoins [18]. The Miner botnet consisted of two disjoint networks containing around 38.000 non-NATed peers according to Kaspersky [28]. Miner ceased to operate in March 2012.

*8) Zeus P2P:* The first two variants of Zeus were centralized, and were extensively tracked and attacked by malware researchers[3]. The development of Zeus forked into multiple variants, including a new P2P variant of Zeus, which appeared in the wild in September 2011. This P2P variant of Zeus appears to be designed to withstand the attacks routinely executed against traditional Zeus botnets. Note that while the centralized variant of Zeus typically forms several distinct botnets, the P2P variant spans one coherent botnet. The P2P variant of Zeus uses an unstructured push/pull-based P2P network to relay commands, stolen data, and configuration/binary updates. Zeus is designed to steal credentials (particularly for financial institutions) from infected systems. This stolen data is sent to dropzones via frequently changing proxy bots, the locations of which are gossiped via the Zeus P2P network.

## III. A Formal Model for P2P Botnets

This section presents a formal model to capture the fundamental characteristics of all previously described P2P botnets. We define a *non-routable peer* as a peer that cannot be reached by other peers, but has the ability to contact one or more peers (e.g., the peer is behind a NAT or proxy). We define a *routable peer* as a peer that can also be contacted by other peers (via an ingress connection). Furthermore, we define an *unreachable peer* as a peer that cannot be reached by any peers nor contact other peers (e.g., the peer is offline), but is still known to one or more peers.

**Definition 1.** *A* peer-to-peer (P2P) botnet *is a directed graph $G := (V, E)$, where $V$ is a set of peers and $E \subseteq V \times V$ edges $(u, v)$ with $u, v \in V$. The set of peers $V := V_r \dot\cup V_n \dot\cup V_u$ is the disjoint union of routable peers $V_r$, non-routable peers $V_n$ and unreachable peers $V_u$.*

Note that the graph is not required to be a *connected digraph*[4] and that several disjoint connected components are in accordance with Definition 1. This conforms with our definition of P2P botnets as we consider two machines infected with the same bot to be part of the same botnet, even if they belong to separate partitions of the P2P graph at a given point in time.

All P2P botnets implement the concept of *peer lists* to keep track of neighboring peers. From Definition 1, we can now describe a peer list as follows:

**Definition 2.** *Let $G = (V, E)$ denote a P2P botnet. The set of edges $E_v := \{(v, u) \in E\}$ for a peer $v \in V$ is called the* peer list *of $v$.*

A peer list expresses relationships of neighboring peers in the graph at a given time. In reality, peer lists can be highly dynamic and do not necessarily have to be stored explicitly. Note that a peer list may contain edges to routable, non-routable and unreachable peers.

**Definition 3.** *We call $deg^+(v) := |E_v|$ the* out-degree *of $v$. $deg^-(v) := |\{(u, v) \in E\}|$ is called* in-degree *of $v$.*

For P2P botnets, $G$ is hardly *complete*, i.e., $\exists u, v \in V : (u, v) \notin E$. This follows from changes in the network (peers joining or leaving the network) that would otherwise require expensive operations to maintain the graph topology. In fact, botnets usually implement *sparse* graphs, i.e., $\forall v \in V : deg^+(v) \ll |V|$. Since peer lists are usually limited in size, the maximum possible out-degree is normally restricted as well. The in-degree of a peer is not easily determined. One would have to explore the entire graph and count the edges $(u, v) \in E$ for a peer $v$. However, the in-degree is an important measure for the *popularity* of a peer. The more popular a peer is in a botnet, the more influence it has on the process of propagating information, like commands.

Using our formal model as a tool, we can express several operations. The deletion of an edge $(u, v)$ in the graph is represented by a transformation $D : G \to G'$ with $G' := (V, E')$ and $E' := E \setminus (u, v)$. This operation occurs, for instance, when a peer performs its regular maintenance

[3] https://zeustracker.abuse.ch

[4] For simplicity reasons, we call a digraph $G$ *connected* if its undirected representation is connected, i.e., $\forall v_1, v_n \in V \; \exists (v_1, v_2, ..., v_n)$ with $(v_i, v_{i+1}) \in E, i = 1..(n-1)$.

for peer connectivity, and removes unreachable peer entries from its peer list. $D^* = D_n \circ D_{n-1} \circ \ldots \circ D_1$ denotes the composition of multiple delete operations. The insertion of a new edge $(u, v)$ in the graph (the introduction of a new peer-to-peer relationship) can be phrased as a transformation $I : G \to G'$ with $G' := (V', E')$ where $V' := V \cup \{v\}$ and $E' := E \cup \{(u, v)\}$. $I^*$ is the composition of multiple inserts.

A restriction of our model is that edge updates (modifying existing peer list entries with more recent information) cannot be expressed directly. This would require the ability to parametrize and distinguish edges and significantly increase the complexity of our model. However, if we regard edge updates as part of a protocol logic at a higher level, we can alter our model to include an update operation $U := I \circ D$, defined as an edge deletion followed by an edge insertion. $U^*$ denotes multiple subsequent updates.

The insert, update and delete operations on a P2P botnet graph provide us with the primitives necessary to describe the reconnaissance and mitigation strategies discussed in the following section.

## IV. Attacks against P2P Botnets

This section presents generic attacks against P2P botnets and presents a formal description thereof. The attacks are based on the following two observations: (1) In order for a P2P botnet to be functional, participating peers must be cooperative, i.e., they must communicate with other peers. (2) Peers cannot be authenticated, as a secure authentication scheme (which would involve a central trusted authority) conflicts with the dynamic, self-organizing nature of P2P networks. In summary, P2P botnets rely on the *cooperation* of *untrusted parties*, two weaknesses that can be exploited.

In this section, we define generic attack methods, which can be applied to any P2P botnet that is compliant with our model. Then, we introduce classes of attacks against the two intrinsic weaknesses that build on these methods: *Intelligence Gathering* and *Disruption and Destruction*.

### A. Attack Methods

The more basic attacks are represented by the *Insert*, *Update*, and *Delete* primitives defined in the previous section. For example, deleting an edge from a P2P graph results in reduced overall connectivity and potentially has an influence on the speed at which information propagates. Below, we discuss four more advanced attack methods.

*1) Graph Search:* Many attacks rely on knowledge about the P2P topology of a botnet. One approach to reconstruct the P2P graph is visiting all nodes, requesting their peer lists, and enumerating all edges. This can be achieved through a graph search. We call the result a *crawl graph* of $G = (V, E)$ and denote it as $G^C = (V^C \subseteq V,\ E^C \subseteq E)$. Peers $v \in V : deg^-(v) = 0$ are invisible during the graph search. This is a significant limitation of a graph search. Furthermore, only routable peers can be contacted. Consequently, the best possible graph search result is a graph $G^C$ with $E^C = \{(u, v) : u \in V_r,\ v \in V\}$. In other words, in the optimal case, the graph search has explored the peer lists of all routable peers. This is hard in practice, since most P2P botnet topologies are so dynamic that they change during the graph search, which leads to inaccurate results. Also depending on the seed peer list, some peers may not be explorable via graph search: Let $V = V' \cup V''$ and the graph search start with a peer in $V'$. Peers in $V''$ cannot be explored during graph search if $\forall v \in V' \cap V'' : v \in V_r \Rightarrow \nexists (v, u \in V'') \in E$.

Algorithm 1 describes a generic P2P botnet graph search algorithm. In line 1, it is initialized with a set of seed peers, which can be obtained through reverse engineering bot samples or dynamic analysis. Peers that reply to peer list requests are added to the set of cached routable peers $V_r^C$ in line 5. Next, their neighboring peers are added to the peer cache $V^C$. The list of edges is updated in line 7. One can obtain a snapshot of the P2P botnet's state by stopping the search process if it has not explored any new edges during a pass over the peer cache. Better performance generally leads to more accurate exploration results, because the longer a cycle takes, the higher the chance that the topology changes in the meantime. Another parameter is the peer selection strategy (line 9). Most searches are performed breadth-first by implementing $V^C$ as a queue to prioritize exploration of local environments in rapidly changing topologies. Using a stack instead would change it into a depth-first search.

**Algorithm 1** P2P Botnet Graph Search

```
1:  V^C ← seed peer list
2:  E^C ← ∅
3:  while true do
4:    if peer list L_u ⊆ E_u received then
5:      V_r^C ← V_r^C ∪ u
6:      V^C ← V^C ∪ {v : (u, v) ∈ L_u}
7:      E^C ← E^C ∪ L_u
8:    else
9:      select u ∈ V^C
10:     request peer list from u
```

*2) Peer Injection:* Most attacks against P2P botnets are based on changes of the graph topology by manipulating the set of edges or the set of vertices. The latter alone is not reasonable, as a newly added peer can not affect the topology as long as it is unknown to other peers. So in order to affect the topology, manipulations of $E$ are mandatory. If we let $I(v) : G \to G' = (V', E')$ denote a parametrized insert operation with $V' = V \dot\cup v, E' = E \dot\cup \{(u, v)\}, u \in V$, we can define the injection of a peer $v$ as a composition $I^*(v) := I_n(v) \circ I_{n-1}(v) \circ \ldots \circ I_1(v)$. It follows that $deg^-(v) \geq 1$. Note that $v$ does not have to be routable.

*3) Peer List Destruction:* In contrast to peer injection, peer list destruction describes "corrupting changes" to a

peer's peer list. The context here is an individual peer, not the entire P2P graph. To destroy a peer list, entries can either be deleted or replaced with invalid (unreachable or non-routable) entries, i.e., peers from $V \setminus V_r$. We let $deg^+_G(v)$ denote the out-degree of $v$ in $G$ and define $D(v) : G \rightarrow G' = (V, E')$ as a parametrized delete operation with $E' = E \setminus (v, u), u \in V$. $D^*(v)$ is the parametrized composition thereof, $U(v)$ and $U^*(v)$ are the equivalent update operations. The destruction of $v$'s peer list is a transformation $R(v) := U^*(v) \circ D^*(v) : G \rightarrow G' = (V', E')$. Note that updates may create new vertices, thus the set of peers may change. It follows that $deg^+_{G'}(v) \leq deg^+_G(v)$ (reduced out-degree if edges were deleted) and $|\{(v, u)\} \cap E'| < |\{(v, u)\} \cap E|, u \in V_r$ (fewer edges to routable peers through invalidation).

### B. Class I: Intelligence Gathering

Attacks against P2P botnets are often preceded by attempts to enumerate the infected hosts and collect information about them. We distinguish two complementary approaches, *crawling* and *sensor nodes*, and define attacks in these categories based on the previously discussed methods.

*1) Crawling:* A standard approach for analyzing P2P botnets is to visit as many peers as possible and collect information about them. The collected information can be anything that is accessible to other peers, which depends on the specific communication protocol. For example, one protocol may exhibit the operating system version, another might include the current local time. Crawling is based on graph search and represents an efficient way to gather intelligence. However, the coverage that can be reached with this approach depends heavily on the P2P protocol details. If, for example, only routable peers are included in local peer lists, the crawler's view is very limited.

*2) Sensor Nodes:* If knowledge about the peers has priority over graph reconstruction, an alternative strategy can be used. Kang et al. introduced a method for enumerating *structured* P2P botnets [12]. The authors proposed special *sensor* nodes in the DHT space of a botnet. We generalize this idea based on the observation that in current P2P botnets peers are periodically contacted by their neighboring peers, e.g., during regular peer list verification cycles. Introducing a sensor can be achieved through peer injection. Sensors can also be contacted by non-routable peers $v \in V_n$, which potentially overcomes some of the shortcomings of crawling. However, a sensor's coverage depends on its popularity. The in-degree $deg^-(s)$ of a sensor can be increased, for example, by crawling the P2P graph and injecting $s$ into any visited peer's peer list. Depending on the botnet protocol, $s$ may then be propagated further due to communication between peers, thereby also reaching peers in $V_n$. Whether it is necessary to continuously announce the sensor to maintain its popularity, or peer list entries are long-lived and automatically propagate, depends on the botnet's peer list dynamics. Thus, the characteristics of a botnet's P2P protocol are vital for the coverage achieved with this approach.

### C. Class II: Disruption and Destruction

Reconnaissance is often only the first step in a series of attacks that aim to render a botnet inoperable. We further distinguish three disruptive attacks, of which two are performed on the infrastructure layer (*partitioning* and *sinkholing*) and one affects both the infrastructure and the communication layer (*poisoning*).

*1) Partitioning:* One generic attack against a P2P botnet's infrastructure tries to prohibit the distribution of information by partitioning the graph. For example, if certain nodes can be identified as the source of new commands, one can try to isolate these nodes by eliminating all edges with other peers. To invalidate an edge, it can be deleted or replaced by applying the peer list destruction method. The concrete list of edges to eliminate depends on whether the protocol is push or pull-based: If commands are to be pushed from a peer $v$ to its neighbors, one would execute $R(v)$. If commands are to be pulled from a peer $v$, one would execute $R(u)\ \forall u : (u, v) \in E$. Note that in the latter case the edges $(u, v)$ are not easily determined, and that the attack may require preceding reconnaissance steps.

Let $R^*$ denote a series of consecutive peer list destruction operations. We define *partitioning* as a transformation $R^* : G \rightarrow G' = (V' \dot\cup V'', E)$ such that $(u, v) \in E : u \in V' \Rightarrow v \notin V'' \wedge u \in V'' \Rightarrow v \notin V'$. In other words, the set of peers is split into two disconnected subgraphs by removing all edges between them.

In the most extreme scenario, each peer represents its own partition, which makes propagation of information completely impossible. That is, the set of edges to routable peers in the P2P graph is empty: $\forall u \in V, v \in V_r : (u, v) \notin E$. This is generally hard to achieve, as the foremost goal of most P2P protocols is to keep a peer's connectivity up.

Note that partitioning requires knowledge about the graph topology, i.e., the edges to eliminate. Such knowledge can be obtained by crawling, for example. However, the more edges are eliminated, the harder it becomes to crawl the network. Injecting a sensor node can help alleviate this problem.

A more general destructive transformation does not necessarily create partitions, but it decreases the popularity of nodes by deleting certain edges from the P2P graph, resulting in a reduced in-degree for a peer. The respective notation for this kind of attack is $R^* : G \rightarrow G' = (V, E'), \exists v \in V : deg^-_{G'}(v) < deg^-_G(v)$. Note that a reduced in-degree of $v$ implies a reduced out-degree of some $u \in V$ with $(u, v) \in E$, and vice versa. The more peers are attacked in this manner, the more sparse the resulting graph becomes, which potentially slows down information propagation. Attacks against "hot spots" (very popular nodes) can have a significant effect on the time required to propagate data.

*2) Sinkholing:* Another disruptive attack against a P2P botnet infrastructure is called *sinkholing*. In this attack, all edges are either invalidated or replaced with edges pointing to special nodes called *sinkholes*. A sinkholing attack effectively transforms the infrastructure into a centralized network, with the set of sinkholes $S := \{s_1, s_2, \ldots, s_n\}$ being the central component for all P2P communication. The attack can be described as a transformation $T : G \to G'$ where $T := I^* \circ R^*$ consists of peer injections and peer list destructions. The goal is to reach a state where every live peer knows at least one sinkhole but no other routable peer: $\forall v \in V_r \dot{\cup} V_n \; \exists s \in S : (v, s) \in E_v \wedge (v, u \neq s) \in E_v \Rightarrow u \notin V_r$.

A sinkholing attack is usually combined with a strategy to announce the sinkholes' existence in the botnet. Sometimes, this part is performed by the sinkhole itself, since it has comprehensive knowledge about other peers: Due to its popularity, lots of peers contact the sinkhole on a regular basis. If the sinkhole also performs graph searches (i.e., it requests peer lists from connecting peers), it can reconstruct parts of the botnet infrastructure and use this information to announce itself. Other options are to utilize crawling, or special injected peers that announce the sinkholes.

The sinkhole peers do not necessarily have to exist; sometimes, they merely act as a "black hole" that absorbs all incoming traffic. However, depending on a specific P2P botnet's program logic, sinkholes may have to implement a subset of the protocol to retain their popularity. For example, a sinkhole may have to exchange actual P2P messages to remain in the local peer list of a bot.

*3) Communication Layer Poisoning:* The term *poisoning* describes a class of attacks where specially crafted information is injected into a botnet. This requires access to the P2P infrastructure, which can be achieved by peer injection. The range of poisoning attacks is huge: Depending on a botnet's command protocol one could distribute commands to other bots or transmit invalid messages that put recipients in a non-functional state. Botnets can be attacked both on the infrastructure and on the communication layers. Well-known examples are the *Sybil Attack*, where an attacker introduces a peer with multiple identities, and the *Eclipse Attack*, in which peers are strategically placed in the botnet graph such that certain communication has to pass through them on the route to their destination [26]. Since the concrete implementation of poisoning attacks depends heavily on the C&C protocol used, they cannot easily be captured in our infrastructure-focused model without losing simplicity and generality. We will provide examples of poisoning attacks against specific botnets in Section VI-A.

## V. P2P Botnet Intelligence Gathering

A P2P botnet topology offers unique possibilities to gather intelligence about the bots. For example, botnet researchers may attempt to estimate the population size of a P2P botnet by enumerating infected hosts. Lists of enumerated bot addresses can be used by CERTs and ISPs to inform infected customers, or by banks to identify fraudulent transactions. Furthermore, it is usually possible to collect additional information, such as the operating system version of infected systems, current C&C commands, or hints about the geographical location of bots. In this section, we experimentally evaluate and compare the reconnaissance techniques described in Section IV, namely *crawling* and the injection of *sensor nodes*, by applying them to the botnets from Section II.

### A. Resilience Against Peer Enumeration

We performed an initial analysis of how botnets are protected against peer enumeration by reverse engineering the communication protocols of six active botnet variants (Figure 1). Table II summarizes our analyses of the different peer list management strategies, augmented with data from technical reports [2, 30, 7, 3, 21]. A vital aspect for peer enumeration is the ability to uniquely identify peers. Our comparison shows that not all botnet protocols use unique peer identifiers. The lack of unique IDs can skew enumeration results considerably. If a bot changes its IP address during a crawl, the number of counted bots will be too high. Similarly, if multiple bots share an Internet-facing IP address, e.g., because they are behind a common NAT gateway, the number of infections is underestimated. Storm, Waledac, Zeus and Kelihos implement unique identifiers which we can use to distinguish between bots. The Miner botnet does not have any notion of peer IDs, while ZeroAccess, Nugache and Sality do not typically exchange IDs.

| Botnet | IDs | $V_n$ | # | Preference | Period |
|---|---|---|---|---|---|
| Kelihos v1 | 16 bytes | | 250 | more recent | 10m |
| Kelihos v2 | 16 bytes | | 250 | more recent | 10m |
| Kelihos v3 | 16 bytes | | 250 | more recent | 10m |
| Miner | none | | all | n/a | 30m |
| Nugache | (not shared) | | 100 | more recent | random |
| Sality v3 | (not shared) | | 1 | random | 40m |
| Sality v4 | (not shared) | | 1 | random | 40m |
| Storm | 16 bytes | ✓ | 10 | small distance | 10m |
| Waledac | 20 bytes | | 100 | more recent | 30s |
| ZA v1 | (not shared) | ✓ | all | n/a | 15m |
| ZA v2 | (not shared) | ✓ | 16 | more recent | 256s |
| Zeus | 20 bytes | ✓ | 10 | small distance | 30m |

Table II: P2P botnet properties relevant for crawling. *IDs* shows if bots have unique identifiers. $V_n$ indicates if non-routable peers are included in peer lists. # shows the maximum number of peers that can be exchanged at once. *Preference* describes how new peers are included in a peer list. *Period* is the period between communication rounds of peers with their neighbors.

In general, the set of peers $V^C$ learned through crawling is limited to routable peers and their neighbors: $V^C \subseteq \{(u, v)\} : u \in V_r$. Most botnets listed in Table II limit peer list entries even further to only routable peers, with three exceptions. Storm activated port forwarding using UPnP to tunnel through NAT gateways. The two ZeroAccess variants

do not filter peer list entries in any way, and thus have no restrictions on what entries they store.

The number of peers shared with other peers (as opposed to the number stored in the local peer list $E_v$) also influences peer enumeration. Only Miner and ZeroAccess v1 share their complete list $E_v$; all others select a subset of it.

The peer selection strategy influences enumeration results as well. Table II shows how many and which peers are shared by the various botnets. The Sality variants choose peers from their list $E_v$ at random, whereas Storm and Zeus compute XOR distances to a target ID (contained in the peer list request) and share peers close to the target. The remaining botnets share a subset of their $E_v$ that contains peers which have recently been verified to be alive.

Lastly, we look at the frequency with which peers contact their neighbors. This frequency is especially important for sensor nodes, as they rely on communication established by other peers. The shorter the period shown in Table II, the more timely the intelligence gathered at a sensor. Waledac has an extremely short period of 30 seconds; Sality contacts its neighbors every 40 minutes. Nugache supports attacker-invoked commands to control peer list exchange behavior.

### B. Peer Enumeration: Real-World Observations

To evaluate how the peer enumeration methods (Section IV-B) perform in practice, we implemented crawling and sensor injection attacks for all four active P2P botnet families. In total, we enumerated eleven botnets that were active in November 2012. We chose a measurement period of 24 hours to account for diurnal patterns and to limit the effects of IP churn. Due to the nature of connectionless protocols, UDP-based botnets can generally be crawled faster, resulting in greater coverage. Similarly, peer injection is more efficient for UDP. Thus, we deployed sensor nodes only in the seven UDP-based botnets, where protocol limitations have less impact on the comparability of the results. A comparison of the two methods for TCP-based botnets (Kelihos v3 and ZeroAccess v1) is work in progress. We injected our sensor nodes three weeks before the measurements so that they had sufficient time to become popular in each botnet. We monitored the daily number of peers contacting our sensors to verify that the popularity of our sensors had stabilized before performing our measurements.

Table III details the enumeration results for a typical weekday in November 2012 from 00:00 GMT on. All values are based on the number of unique IP addresses that were logged during the respective time period. While using bot IDs would be more accurate, we chose to use IP addresses in this comparison, as most botnets do not share peer IDs. The *Botnet* column names the enumerated botnets, augmented with the bot version number and the fixed UDP/TCP port the botnet binds to, where applicable. The *Crawling* section of the table represents our crawling results, and the *Sensor* section displays the results for our sensor injection experiments. In the crawling results, we distinguish between peers that responded to peer list requests ($V_r^C$) and all peers found during crawling ($V^C$). The column $|V_r^C|/|V^C|$ contains the ratio of routable peers. This value varies between 13% (Kelihos) and 40% (Sality) and depends on the protocol properties in Table II, as discussed above. The number $|V^S|$ in the Sensor column is the number of peers that connected to our sensor during the 24 hour period. The section *Overlap* shows the number of peers that were identified by both methods. The numbers in parentheses display the percentage of peers found by the crawler that were also logged by the sensor. Finally, the *Sensor Gain Factor* is the ratio of the number of peers found by the sensor divided by the number of peers found through crawling.

The results show that crawling is less complete compared to sensor based enumeration in all P2P botnets we have evaluated. In particular, the Sality crawlers identified less than 3% of the IP addresses that our sensor found, and got responses from less than 1%. This means that crawls on the Sality network underestimate the population by a factor of 110.6 (version 3), and 113.3 (version 4), respectively. A similar effect can be observed for other botnets that rarely propagate peers $v \notin V_r$, such as Zeus. If, on the other hand, a botnet allows for such entries in its peer lists, as ZeroAccess does, the divergence between crawling and sensor based enumeration is reduced. This shows that one has to be careful when extrapolating the size of the total botnet population from crawling results.

Table III shows that the sensors identified 87.1–98.5% of the peers verified by the crawlers ($V_r^C$). Only in the case of ZeroAccess does the sensor show less than 80% completeness compared to the crawl of all peers ($V^C$), presumably because of the significant fraction of invalid IP addresses obtained by the crawler. Our sensors enumerated more than a million infections per day (unique IP addresses) for ZeroAccess and Sality. Another interesting detail that can be seen from the enumeration results is the distribution of CPU architectures for ZeroAccess v2: 37% of the bots run on 64 bit (ports 16465 and 16470), and 63% on 32 bit (ports 16464 and 16471). Despite biases caused by IP address churn, we believe that our combination of crawling and sensor injection results provides reasonable lower bounds for the sizes of the botnets.

Depending on the protocol, a sensor may have the opportunity to perform additional validity checks for peers in $V_n$. By sending packets to the sensor, a peer behind a NAT establishes a punchhole which the sensor can use to send requests to the peer to check if it responds in a protocol-conformant way. Whether this is possible depends on the transport layer protocol and the message dialogue used for P2P communication. For example, if the botnet uses one TCP session per message, NAT traversal is not possible. In the case of ZeroAccess v2, verification of NATed bots

| Botnet | Crawling | | | Sensor | Overlap | | | | Sensor Gain Factor | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\|V^C\|$ | $\|V_r^C\|$ | $\|V_r^C\|/\|V^C\|$ | $\|V^S\|$ | $\|V^S \cap V^C\|$ | | $\|V^S \cap V_r^C\|$ | | $\|V^S\|/\|V^C\|$ | $\|V^S\|/\|V_r^C\|$ |
| Sality v3 | 22,351 | 8244 | 36.9% | 912,090 | 19,583 | (87.6%) | 7508 | (91.1%) | 40.8 | 110.6 |
| Sality v4 | 2113 | 846 | 40.0% | 95,809 | 1928 | (91.2%) | 833 | (98.5%) | 45.3 | 113.3 |
| ZAv2 16464 | 346,069 | 92,531 | 26.7% | 438,511 | 241,052 | (69.7%) | 87,251 | (94.3%) | 1.3 | 4.7 |
| ZAv2 16465 | 186,290 | 50.684 | 27.2% | 227,328 | 147,554 | (79.2%) | 35,654 | (91.9%) | 1.2 | 4.5 |
| ZAv2 16470 | 271,534 | 46,512 | 17.1% | 294,871 | 147,780 | (54.4%) | 44,062 | (94.7%) | 1.9 | 6.3 |
| ZAv2 16471 | 350,436 | 88,804 | 25.3% | 443,039 | 254,946 | (72.8%) | 85,646 | (96.4%) | 1.3 | 5.0 |
| Zeus | 63,976 | 8306 | 13.0% | 193,495 | 55,989 | (87.5%) | 7234 | (87.1%) | 3.0 | 23.3 |
| Kelihos v3 | 15,017 | 1877 | 12.5% | n/a | n/a | | n/a | | n/a | n/a |
| ZAv1 22292 | 7854 | 2757 | 35.1% | n/a | n/a | | n/a | | n/a | n/a |
| ZAv1 25700 | 4701 | 3057 | 65.0% | n/a | n/a | | n/a | | n/a | n/a |
| ZAv1 34354 | 35,046 | 9723 | 27.7% | n/a | n/a | | n/a | | n/a | n/a |

Table III: Comparison of P2P node enumeration techniques, measured in numbers of IP addresses found in 24 hours.

by sensors is possible. Our results show that 98% of the ZeroAccess v2 peers contacting the sensor also responds to peer list requests. This type of peer verification is generally impossible for crawlers, as they cannot reach peers in $V_n$.

Unverified IP addresses are a problem when reporting potential infections to CERTs and affected institutions for incident response. For this reason, we provide high quality feeds of verified IP addresses based on our enumeration efforts to the security community. We also take great care of identifying other researchers participating in the analyzed P2P botnets. For example, we identified two parties who crawled the Zeus botnet and we excluded their randomly generated peer IDs from our counts.

We summarize that, although helpful in certain situations, crawling can generally only provide a limited view on the overall botnet population. As a result, previously published botnet population estimates based on crawling may be skewed. Having said that, we acknowledge that crawling does have its purpose, as it is not always feasible to apply sensor injection to a botnet. Another methodological difference is that crawlers actively enumerate peers, while sensors are reactive in that they wait to be contacted by peers. Thus, for an ideal enumeration, one may need to combine crawling and sensor injection.

### C. Convergence Analysis

Another challenge when estimating the size of a botnet is discussed by Kanich et al. [13]. Infected machines may have dynamic IP addresses that change regularly. If an IP address switch happens during the peer enumeration period, a peer may be counted multiple times, unless the botnet uses unique IDs (cf. Table II). Therefore, we counted both IP addresses and peer IDs in our experiments, where possible. Figure 3 compares these two values for the Zeus botnet. The upper two lines show the peers that contacted the sensor, the middle two lines show all crawled nodes ($V^C$), and the lower two lines depict the number of reachable nodes that were found during crawling ($V_r^C$). Each set of two lines shows the number of IP addresses (upper line) and peer IDs (lower line). For both enumeration methods, the lines showing IP

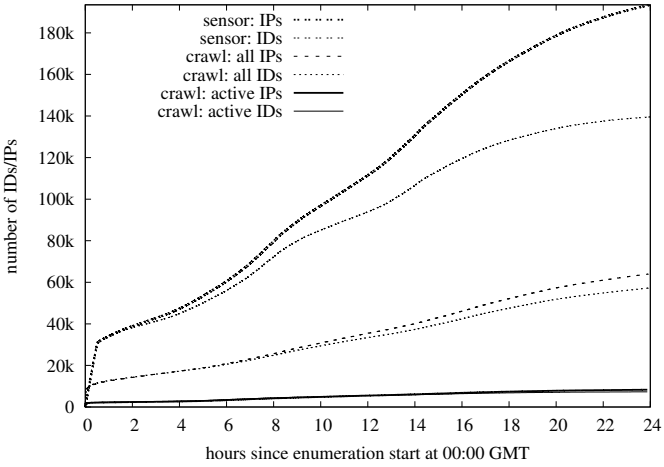


Figure 3: Comparison of enumeration methods for Zeus.

addresses and IDs deviate over time due to IP address churn. While the number of IDs collected at the sensor converges towards 140.000 near the end of the time period, there is no obvious convergence in the other lines. An analysis of a long-term enumeration run showed that the IP address churn is significant for Zeus. On average, 19% of the bot IDs were observed on multiple IP addresses during a 24 hour period. We measured that 3/5 of these IDs appeared on a single /8 network (possibly IP address churn due to ISP-enforced redialing), whereas 2/5 of the IDs were observed in multiple networks (possibly laptops moving among networks). Thus, the IP address count is a less accurate metric for the true number of infected hosts than the ID count.

Figure 3 also shows how the crawling and the sensor node counts develop over time. The steep increase of the sensor count at the beginning is caused by the fact that all Zeus peers contact their neighbors every 30 minutes. After this period, only peers that just learned about the sensor's presence and peers that just entered the botnet still contact the sensor for the first time, and the curve flattens. The slight variations in the lines for both methods are caused by the start of working hours in Europe and in the United States.

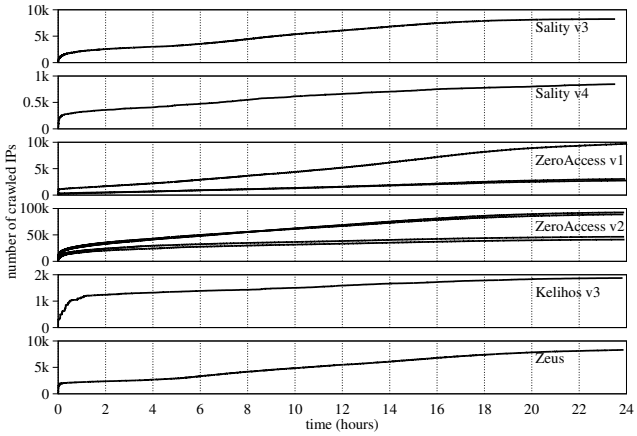


Figure 4: IP address count convergence during a 24h crawl.

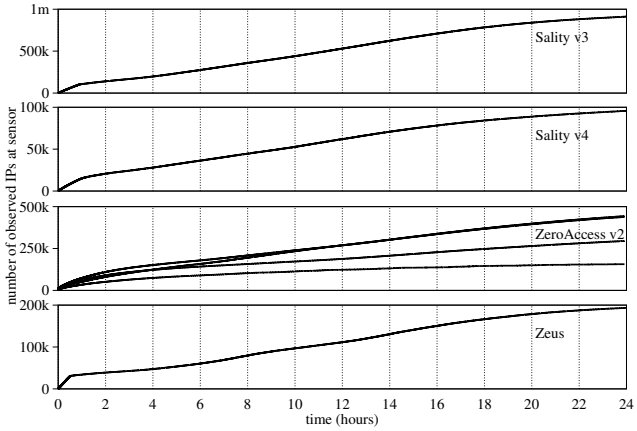


Figure 5: IP address count convergence at a sensor over 24h.

Note that we were forced to rate-limit the crawling process, as Zeus blacklists IP addresses that contact a peer too often. Consequently, the crawling curve is much less steep than the sensor node curve.

Figure 4 compares the IP address count convergence over 24 hours for all the active P2P botnets. A first observation is that only for Kelihos this number converges relatively quickly. This is due to the size of the set $\{v : \exists(u, v) \in E\}$. Kelihos prefers more recent peers and shares them with timestamps, thus new peers spread rapidly over the majority of all peer lists. In all other botnets, crawling converges only slowly – independent of their actual sizes. Figure 5 shows that enumerations via sensor nodes converge at a similar pace. In all cases, though, the sensors enumerate peers faster than the crawlers, and the sensors find many more peers than the crawlers. While IP address churn is one of the reasons for slow node enumeration convergence, it is not the only reason. The next subsection discusses our additional observations regarding the dynamics of botnet populations.

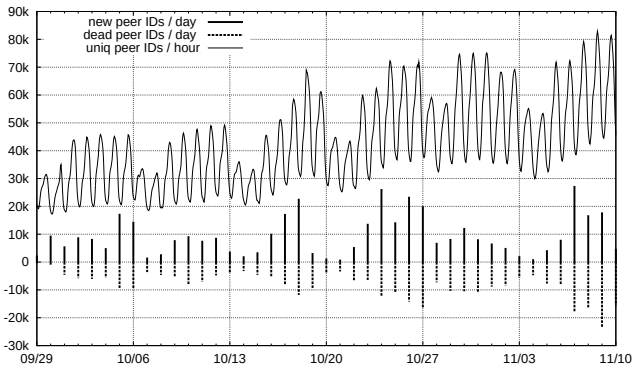


Figure 6: Zeus population footprint graph over six weeks. The curve shows the hourly number of peer IDs contacting our sensor. The bars show the daily number of peers joining (positive bars) and leaving (negative bars) the botnet.

### D. Dynamics of Botnet Populations

Apart from IP address churn, machines joining and leaving the network (e.g., through new infections and disinfections) also cause a steady churn of peers. We assessed this phenomenon for the Zeus botnet, by looking at sensor data from six consecutive weeks. Zeus derives unique peer IDs from host-based information – the peer ID is a hash over the Windows *ComputerName* and the first hard drive's volume ID string, meaning that it remains static even across cleanups and reinfections. We used the static IDs to uniquely identify infected machines in order to measure population sizes independently from IP address churn. The curve in Figure 6 shows the number of peers that contacted our sensor per hour. Clearly, the network footprint is larger on weekdays and during working hours in Europe and the United States. Similar diurnal patterns were observed for all other monitored P2P botnets. The bars in Figure 6 represent the number of joining (positive values) and leaving (negative values) bots per hour. The figure shows that there is a steady change in the network, with up to 25,000 new infections per day. This highly dynamic behaviour means that P2P botnets change significantly during node enumeration runs, so that enumeration efforts are never completely accurate.

To summarize, current P2P botnets are not very resilient to intelligence gathering. However, some botnets do implement simple countermeasures against crawling, like rate limiting of peer list exchanges and automated blacklisting of hard hitters. While none of these measures renders crawling impossible, they can slow down the enumeration process, leading to less accurate results. The use of sensor nodes, when possible, avoids these anti-crawling countermeasures, and typically provides more reliable results.

## VI. P2P Botnet Disruption and Destruction

In this section, we evaluate the resilience of all currently known P2P botnet families against the disruption and destruction attacks introduced in Section IV.

### A. Communication Layer Poisoning Resilience

Depending on the implementation, it may be possible to poison a P2P botnet using its own commands, or to disrupt the C&C channel to prevent legitimate commands from spreading. For example, if commands are not properly authenticated to originate from the botnet operators (e.g., through digital signatures), defenders could issue arbitrary commands, such as a removal instruction or a command to stop spam activities. Storm is an example of a P2P botnet that was vulnerable to both kinds of attacks. Commands were published under periodically changing keys in a DHT. The bots generated a set of time-dependent hashes and queried the DHT for them. The responses encoded the location of a C&C server, which was then contacted to request orders. Holz et al. [9] showed how to perform an Eclipse attack against the botnet by predicting the hash generation algorithm and injecting peers with an ID close to the current hashes. This effectively prevented bots from connecting to their C&C servers. In 2008, a group of researchers also demonstrated how the Storm botnet is vulnerable to command injection, and that anybody can push arbitrary executables and run them on the infected machines[5]. Both attacks were possible because the protocol lacked basic security measures that would have prevented these attacks.

| Botnet | Crypto | Signing | Replay Possible |
|---|---|---|---|
| Kelihos v1 | Blowfish, 3DES | RSA2048 | |
| Kelihos v2 | Blowfish, 3DES | RSA2048 | |
| Kelihos v3 | Blowfish, custom | RSA2048 | |
| Miner | none | none | ✓ (no signing) |
| Nugache | RSA, Rijndael | RSA4096 | ✓ |
| Sality v3 | RC4 | RSA1024 | |
| Sality v4 | RC4 | RSA2048 | |
| Storm | XOR | none | ✓ (no signing) |
| Waledac | RSA1024, AES | none | ✓ (no signing) |
| ZeroAccess v1 | RC4 | RSA512 | ✓ |
| ZeroAccess v2 | XOR | RSA1024 | |
| Zeus | chained XOR[6] | RSA2048 | ✓ |

Table IV: Protocol security properties of P2P botnets.

Table IV summarizes the most important security aspects of the P2P botnets that we analyzed. All bots, except Miner, encrypt their communication. However, the used encryption does not secure the botnets against disruptive attacks, as the keys and algorithms can in all cases be found through reverse engineering. Zeus and Sality also add random padding to their messages, which is meant only to thwart network-based signature detection. Additionally, all active botnets secure their command layers with RSA signatures. This does provide real security by preventing third parties from injecting unauthorized new commands into the botnets. Although Nugache, ZeroAccess v1, and Zeus prevent the forging of new commands through signing, they do not prevent replay attacks because their commands do not carry nonces.

[5]Presentation *Stormfucker: 0wning the Storm Botnet*, Chaos Communication Congress 25C3, Berlin, Germany, December 2008.

[6]Encrypts a message by performing a bytewise XOR operation with the preceding byte, starting at the end of the message.

### B. Sinkholing Resilience

As we discussed in Section IV-C2, sinkholing a P2P botnet involves manipulation of the peer lists for all bots in the botnet such that the bots' peer list entries no longer point to other bots, but instead to sinkholes. It is a very effective way to disrupt the communication within a P2P botnet. When performing a sinkholing attack, we distinguish the following general steps.

1) *Sinkhole announcement*: We bootstrap the sinkholing attack by announcing some sinkholes to as many peers as possible.
2) *Node isolation*: We then try to eliminate all edges in the P2P graph that do not point to a sinkhole. The goal of this step is to isolate the bots from each other as much as possible, rendering them unable to communicate with other peers.
3) *Fallback prevention*: Some P2P botnets have other C&C mechanisms that they fall back to under certain circumstances, e.g., if they have not received a new command in some time. A sinkholing attack must ensure that bots do not activate backup C&C channels to recover. This step requires that either (1) the botnet's backup channel is somehow disabled, or (2) the bots are prevented from entering into their fallback mode.

| Botnet | 1) Sinkhole announcement | 2) Node isolation | 3) Backup C&C |
|---|---|---|---|
| Kelihos | replace less recent | all | fast-flux |
| Miner | append to existing | not possible | central |
| Nugache | replace less recent | 100 per message | central |
| Sality | replace low-rated | 1 per IP address | none |
| Storm | replace any | 20 per message | none |
| Waledac | replace less recent | all | fast-flux |
| ZeroA. v1 | replace less recent | all | none |
| ZeroA. v2 | replace less recent | 16 per message | none |
| Zeus | replace any | 10 per message | DGA |

Table V: Sinkholing resilience properties of P2P botnets.

Table V summarizes the sinkholing resilience properties for all botnets, following the three strategic steps proposed above. The first column shows how a sinkhole can replace peer list entries. In the second column, we denote how many entries can be destroyed in a single P2P exchange. The third column lists fallback command-and-control channels, if any. During our analysis, we evaluated how these properties influence the feasiblity of sinkholing attacks against all current P2P botnets by developing attack prototypes and testing them in the wild. The results are described in the remainder of this section.

*1) Kelihos:*

a) *Sinkhole announcement*: Kelihos has major weaknesses in all of its versions, since they are all based on the same underlying protocol. The bots favor more recent

peers when merging received peer lists into their local list $E_v$. It is possible to actively push new peer lists to other peers, replacing their local entries all at once. As shown in sinkholing attacks against Kelihos v1 and v2, Kelihos can be sinkholed quite effectively by crawling the network and poisoning every visited peer [3]. Although only a few routable peers can be attacked initially, the poisoned entries will then propagate across the network. Since the Kelihos v3 botnet uses effectively the same P2P protocol and architecture as Kelihos v1 and v2, a full scale sinkholing attack analogous to the previous ones would succeed again.

b) *Node isolation*: Node isolation in the Kelihos botnet works very similar to sinkhole announcement. Since it is possible to replace the entire peer list of a peer with arbitrary entries, unlinking it from the P2P graph is trivial.

c) *Fallback prevention*: Kelihos implements a number of backup channels that are activated by a bot if none of its peers can be reached. The most important one is a DNS fast-flux network. Every bot contains a hard-coded domain name that points back to another infected machine. This machine can be used as a seed node to recover connectivity in the P2P network. If the botnet is sinkholed correctly, Kelihos will not resort to the fast-flux domains – as long as the sinkholes are responsive.

*2) Sality:*

a) *Sinkhole announcement*: Sality uses specialized "announce" messages to advertise itself to routable peers in the P2P network. The protocol implements a reputation scheme, in which each bot keeps track of a reputation for all of its neighboring peers. A peer's reputation is increased if it correctly responds to requests, and decreased otherwise [7]. An important detail of this scheme is that bots only share peer list entries with high reputations with each other. So in order to propagate an injected sinkhole entry to non-routable peers we need to ensure that this entry gains a high reputation by conforming strictly to the Sality protocol. In our experiments, we were able to achieve a high reputation, causing our sinkhole addresses to be propagated to many non-routable peers.

b) *Node isolation*: The reputation system significantly hardens the botnet against sinkholing attacks, because Sality only allows peer list entries with low reputations to be overwritten. Thus, while we succeeded in achieving high reputations for our sinkholes, it is significantly more difficult to evict existing high-reputation peers. However, in current Sality implementations, we found a weakness which allows us to poison even high-reputation peer list entries. This requires knowledge of which IP addresses to overwrite, which could for instance be gathered through crawling. The vulnerability lies in the fact that a Sality announcement message can be used to overwrite the port associated with a peer list entry. This requires sending an announcement with a spoofed IP address matching the address of the entry to overwrite, but from a different port. Such a message causes an update of the existing port value, rendering the entry invalid. However, this method can only be used against routable peers, as it is not possible to send unsolicited announcement messages to non-routable nodes. We were unable to find any other ways to poison existing high-reputation Sality peer list entries. Thus, we conclude that Sality's design is extremely resilient to sinkholing.

c) *Fallback prevention*: Sality does not implement a backup C&C mechanism. However, as Sality is a downloader, it could recover from a sinkholing operation with the help of previously dropped malware. Consequently, prior to sinkholing P2P-based downloaders, one must ensure that none of the downloaded files contain functionality (such as the ability to download and execute binaries) that the botmaster could use to recover from the attack.

*3) ZeroAccess:*

a) *Sinkhole announcement*: ZeroAccess uses a constant stream of broadcast messages to announce live peers in the network. These broadcast messages can be leveraged to announce sinkholes, too. However, due to the continuous announcements of other peers, peer list entries are very volatile and are typically overwritten within a few seconds. Therefore, in order to remain in peer lists, it is necessary to keep flooding the botnet with sinkhole announcements. One issue is that the broadcast messages are unable to reach non-routable peers. Nevertheless, the more routable peers that contain sinkhole entries, the more likely they are to propagate these entries to non-routable peers. As soon as this occurs, the fact that these peers are not reachable from the outside turns into an advantage; subsequent peer announcements will not be able to easily replace the sinkhole entries.

b) *Node isolation*: When a ZeroAccess bot receives a new list of peers from another bot, it merges it with its current peer list and keeps the most recent 256 entries, as determined by a timestamp associated with every peer list entry ("future" timestamps are not allowed). Thus, bots can be isolated by sending them peer lists containing invalid entries with very recent timestamps. The two ZeroAccess variants differ in their peer list exchange protocols. In ZeroAccess v1, peer lists are only shared upon request. Thus, poisoning peers in this network requires serving poisoned peer list exchange messages from a sinkhole whenever it is asked for a peer list. In contrast, ZeroAccess v2 accepts unsolicited peer list messages, which makes poisoning almost trivial. In either case, our sinkholing prototype is able to completely overwrite the peer lists of bots for each variant.

c) *Fallback prevention*: ZeroAccess does not implement a fallback command-and-control channel. However, it is potentially able to recover from sinkholing attacks by using its downloaded malware plugins to repair damage and reconnect bots to the P2P network. We did not find such functionality when manually reverse engineering the downloaded plugins (as of Nov 2012), although we found one plugin that tracks all bots via centralized servers.

*4) Zeus:*

a) *Sinkhole announcement*: In the case of Zeus, we are able to announce a sinkhole simply by sending requests to bots. When receiving a request, a Zeus bot adds the source to its peer list if it knows fewer than 50 peers. Because the non-routable peers $V_n$ cannot receive incoming requests, this approach only allows us to announce our sinkholes to routable peers $V_r$. However, our prototype has shown that non-routable peers learn about our sinkholes through peer list exchanges with peers in $V_r$. We refer to this phenomenon as *sinkhole propagation*.

b) *Node isolation*: When a Zeus bot receives a request from a peer whose unique ID is already in the peer list, it updates the respective entry accordingly. This allows us to overwrite peer list entries by sending requests with spoofed source IDs to bots (every Zeus message carries a source ID in its header). In our prototyped attack, we continuously crawl the botnet to find edges between bots, and then redirect as many of these edges as possible to our sinkholes by using the strategy described above. Our sinkhole then poisons non-routable peers by responding to their periodic peer list requests with manipulated peer list replies. Each request allows us to update up to ten entries in the remote peer's peer list (see Table V). Note that we are forced to rate-limit this process because recent versions of Zeus feature an automated IP-based blacklist, which blocks IP addresses with high request rates.

c) *Fallback prevention*: Because Zeus bots periodically verify that their neighboring peers are alive and responsive, our sinkhole is required to implement the full P2P protocol to remain in bots' peer lists. Additionally, Zeus uses a backup strategy which severely complicates long-term sinkholing. If a bot is unable to update itself or its configuration file for seven days, it attempts to obtain a fresh peer list by the following strategy: First, it attempts to contact a set of hardcoded IP addresses to re-establish contact with the P2P network. If this fails, the bot attempts to download a new peer list by triggering a DGA and randomly contacting one of 1,000 weekly generated domains. Because both binaries and configuration files are signed, they cannot be forged. As long as these secondary channels remain intact, sinkholing operations against Zeus are only temporarily successful.

We conclude that sinkholing attacks against Zeus or Sality are difficult. Zeus requires a coordinated effort among multiple domain registries and an attack on the P2P layer. Sality makes it hard to isolate peers, as peers with a high reputation cannot be eliminated from the peer lists of non-routable bots. In the case of ZeroAccess, it is feasible to execute a long-term sinkholing attack against all routable peers. Since routable bots propagate sinkhole entries to non-routable peers, we expect an attack to be successful over time. Kelihos is relatively easy to sinkhole, but requires intensive tracking by researchers due to its frequent encryption and protocol changes.

### C. Partitioning Resilience

Partitioning and sinkholing are closely related attacks. They both destroy existing links in a botnet. Depending on the goals of a botnet takedown operation, partitioning may have advantages over sinkholing. For example, the injected sinkholes are generally easily identified, as they stand out against other bots due to their popularity. Thus, botnet operators may start counter attacks against these sinkholes to regain control over their infrastructure. Partitioning attacks do not expose such attack surfaces and are immune to this type of counter attack.

However, once a botnet has been partitioned, it is next to impossible to regain control over it and perform additional mitigation attempts — this is after all the exact goal of a partitioning attack. There is usually no way to reverse the effects. In contrast, sinkholing adheres control over the botnet and allows for adapted attack strategies at a later point, should that become necessary. More importantly, it generates accurate logs of infected machines and can record the exact time when an infected systems was last active. These logs can be used to coordinate cleanups to ultimately eliminate infections.

We believe that sinkholing is generally the preferred attack, and that partitioning attacks should be seen as a last resort. For this reason, we did not perform and evaluate large-scale partitioning attacks against the live botnets we studied. Instead, we restricted our tests to the smallest possible sub-graph, consisting of only a single peer. We were able to prove that in all cases isolation of nodes is possible. Non-routable peers $v \in V_n$ can remain isolated forever, as there is no way for them to recover unless some backup mechanism re-establishes contact with the botnet. We found routable peers to recover quickly in the botnets we studied, as they are still known to other peers even if their out-degree $deg^+(v) = 0$. Depending on the P2P protocol, these peers may contact the attacked peer and propagate new peer list entries to it. A partitioning attack may not be successful unless it affects the whole P2P network by aggressively eliminating edges until all nodes are isolated.

## VII. Discussion and Future Work

Attacking P2P botnets raises some controversy. In this section, we discuss the most pressing controversial issues, as well as directions for future work.

To begin with, attacks like sinkholing involve unsolicited communication with infected hosts, which could be considered unethical by some. Proponents of such attacks might argue that sinkholing does not cause additional harm, as infected hosts already communicate with other bots.

Another ethical concern is the level of detail our work should reveal to the public about botnet resilience. The concern is that botmasters could use our insights to harden their botnet designs. Therefore, we describe our results such that they cannot be directly applied by botmasters to harden their botnets. On a reassuring note, several detailed strategies for resilient P2P botnet designs have been publicly proposed, but we do not know of any real botnets that make use of these ideas [22, 31, 32, 10]. We believe that the potential of our work to assist security experts justifies its publication. We hope that our insights will help avoid damage to innocent third parties caused by the common takedown problems pointed out by Dittrich et al. [6].

While our evaluation shows that some P2P botnets exhibit a high level of resilience, we also find that all real-world P2P botnets are susceptible to at least one of the mitigation strategies we model. Regardless, implementing mitigation strategies against new P2P botnets remains non-trivial due to the need to understand the peculiarities of each botnet's C&C protocol. Additionally, attacking networks containing millions of peers requires significant resources which may need to remain available over the long term. We believe that a discussion is required concerning alternative mitigation strategies against P2P botnets. Moreover, we think there is a pressing need for debate to establish clear boundaries on how far authorities are allowed to go when disabling P2P botnets. We currently see several alternative mitigation possibilities which we believe are deserving of further analysis.

First, it is sometimes possible to disinfect bots remotely by exploiting vulnerabilities in bot software. This strategy is currently considered unethical because it could cause collateral damage to the hosts being disinfected if executed without great care. Nevertheless, it is a method which may need to be considered if future P2P botnets become immune to more conventional countermeasures.

A second vector for mitigating P2P botnets is to impersonate the botmaster by forging commands for the bots. Section VI-A has shown that this approach is often prohibited by the use of signed commands. However, it may sometimes be possible for law enforcement to capture infrastructure used by the botmaster to create commands, in which case infected hosts could be commanded to clean up.

Finally, some botnets could be mitigated by attacking their monetization models. For example, Zeus gathers banking credentials, and its botmaster relies on the accuracy of the stolen data. Inserting large amounts of invalid banking data could render the botnet unprofitable for the botmaster. For spamming botnets, node enumeration could be used to create spam blacklists, reducing the botnets' revenue.

## VIII. Related Work

In this work, we have discussed the resilience of several current and past P2P botnets. For a full discussion of each of the botnets, we refer the interested reader to malware analysis reports [14, 2, 15, 30, 7, 3, 21]. We have used results from these works to aid our manual code analyses, although in most cases the P2P resilience was yet undocumented.

As noted earlier, a few examples of enumeration and takedown operations against past P2P botnets exist. For instance, Holz et al. performed an early crawl of the Storm botnet, and also discussed some general resilience aspects of structured P2P botnets [9]. The sinkholing results of Stock et al. against Waledac represent the first successful attack against an unstructured P2P botnet [23]. Sinclair et al. have described the vulnerabilities of Waledac in detail [21]. The attacks against previous variants of Kelihos are also examples of recent sinkholing successes against unstructured P2P botnets [27]. Although several works on the resilience of individual botnets exist, our work is the first to systematically compare the resilience of all live P2P botnets.

The problem of crawling P2P botnets was first addressed by Kanich et al., based on lessons learned while crawling the Storm botnet [13]. An alternative concept to enumerate infected hosts (included NATed hosts) in structured P2P botnets was proposed by Kang et al. [11]. Their method involves the introduction of many fake nodes (sensors) into the target structured botnet. These sensors find infected hosts by monitoring search requests from bots looking for commands. We extended this approach and applied it to several unstructured P2P botnets, providing us with much more accurate enumeration results than traditional crawling.

In several previous works, graph models have been used to describe network structures. Holme et al. used graph models to study the response of complex networks to several attacks [8]. The first application of random graphs, small world structures, and scale free networks in the context of botnets was given by Dagon et al. [4]. Davis et al. used graph simulations to analyze the impact of bot disinfections on the communication effectiveness of P2P botnets [5]. Recently Yen and Reiter discussed the role of assortative mixing in P2P botnets and its consequences for network resilience and recovery [33]. However, to the best of our knowledge, our work is the first to introduce formal definitions for the systematization of attacks against P2P botnets.

To explore the threats we may expect from future P2P botnets, several researchers have designed their own theoretical highly resilient P2P botnets [22, 31, 32, 10]. We are not aware of existing P2P botnets based on these proposals.

## IX. Conclusion

We have presented a model which formalizes reconnaissance and disruption attacks to support mitigation efforts against P2P botnets. We have used this model to analyze several live real-world P2P botnets in two ways.

First, we have estimated and compared the population sizes of current P2P botnets using crawlers and sensor nodes. We have shown that sensor nodes reveal large numbers of bots which cannot be found using crawlers. We conclude that combining crawlers and sensor nodes can provide much more accurate population estimates than crawling alone.

Second, we have evaluated the disruption resilience of all four current P2P botnet families through a combination of static analysis and attack prototyping. Our evaluation has shown weaknesses which could be used to disrupt the Kelihos and ZeroAccess botnets. However, we have also shown that the Zeus and Sality botnets are highly resilient to sinkholing attacks, the currently most used class of disruptive attacks against P2P botnets. We believe our findings demonstrate that research on alternative P2P botnet mitigation methods is urgently needed.

## X. Acknowledgements

We sincerely thank The Shadowserver Foundation and SURFnet for their friendly support. We also thank Dave Dittrich for his insights on Nugache, James Wyke and Kevin McNamee for the exchange on ZeroAccess, and Tomasz Bukowski for the collaboration on Zeus. This work was supported by the Federal Ministry of Education and Research of Germany (Grant 16BY1110, MoBE), the European Research Council Starting Grant "Rosetta" and the EU FP7-ICT-257007 SysSec project.

## References

[1] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[2] T. Bukowski. ZeuS v3 P2P Network Monitoring, 2012. Technical Report by CERT.pl.

[3] P.-M. Bureau. Same Botnet, Same Guys, New Code: Win32/Kelihos. In *VirusBulletin*, 2011.

[4] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, 2007.

[5] C. R. Davis, S. Neville, J. M. Fernandez, J.-M. Robert, and J. McHugh. Structured Peer-to-Peer Overlay Networks: Ideal Botnet Command and Control Infrastructures? In *Proceedings of the 13th European Symposium on Research in Computer Security*, 2008.

[6] D. Dittrich. So You Want to Take Over a Botnet. In *Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats*, 2012.

[7] N. Falliere. Sality: Story of a Peer-to-Peer Viral Network, 2011. Technical Report by Symantec Labs.

[8] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han. Attack Vulnerability of Complex Networks. *Physical Review E, vol. 65*, 2002.

[9] 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 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[10] R. Hund, M. Hamann, and T. Holz. Towards Next-Generation Botnets. In *Proceedings of the 2008 European Conference on Computer Network Defense*, 2008.

[11] B. Kang, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim. Towards Complete Node Enumeration in a Peer-to-Peer Botnet. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 2009.

[12] J. Kang and J.-Y. Zhang. Application Entropy Theory to Detect New Peer-to-Peer Botnets with Multi-chart CUSUM. In *Proceedings of the 2nd International Symposium on Electronic Commerce and Security*. IEEE Computer Society, 2009.

[13] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[14] A. Lelli. Zeusbot/Spyeye P2P Updated, Fortifying the Botnet, 2012. Technical Report by Symantec Labs: http://www.symantec.com/connect/node/2135671.

[15] K. McNamee. Malware Analysis Report: ZeroAccess/Sirefef, 2012. Technical Report by Kindsight Security Labs.

[16] J. Nazario and T. Holz. As the Net Churns: Fast-Flux Botnet Observations Tracking Fast-Flux Domains. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software*, 2008.

[17] P. Porras, H. Saidi, and V. Yegneswaran. Conficker C Analysis, 2009. SRI International Technical Report.

[18] D. Plohmann and E. Gerhards-Padilla. Case Study of the Miner Botnet. In *Proceedings of the 4th International Conference on Cyber Conflict*, 2012.

[19] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *ACM EuroSys BADGERS*, 2011.

[20] V. R. Sergey Golovanov. TDSS, 2010. Technical Report: http://www.securelist.com/en/analysis/204792131/.

[21] G. Sinclair, C. Nunnery, and B. Kang. The Waledac Protocol: The How and Why, 2009. Technical Report by Infrastructure Systems Research Lab/University of North Carolina.

[22] G. Starnberger, C. Kruegel, and E. Kirda. Overbot: A Botnet Protocol Based on Kademlia. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, 2008.

[23] B. Stock, M. Engelberth, F. C. Freiling, and T. Holz. Walowdac – Analysis of a Peer-to-Peer Botnet. In *Proceedings of the European Conference on Computer Network Defense*, 2009.

[24] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[25] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. In *USENIX; login, vol. 32, no. 6*, 2007.

[26] G. Urdaneta, G. Pierre, and M. van Steen. A Survey of DHT Security Techniques. *ACM Computing Surveys, vol. 43*, 2011.

[27] T. Werner. Botnet Shutdown Success Story: How Kaspersky Lab Disabled the Hlux/Kelihos Botnet, 2011. Technical Report: http://www.securelist.com/en/blog/208193137/.

[28] T. Werner. The Miner Botnet: Bitcoin Mining Goes Peer-To-Peer, 2011. Blog article by Kaspersky Lab: http://www.securelist.com/en/blog/208193084/.

[29] T. Werner. Kelihos.C: Same Code, New Botnet, 2012. Blog article by CrowdStrike: http://blog.crowdstrike.com/2012/03/kelihosc-same-code-new-botnet.html.

[30] J. Wyke. ZeroAccess, 2012. Technical Report by SophosLabs.

[31] G. Yan, S. Chen, and S. Eidenbenz. RatBot: Anti-enumeration Peer-to-Peer Botnets. In *Lecture Notes in Computer Science, vol. 7001*, 2011.

[32] G. Yan, D. T. Ha, and S. Eidenbenz. AntBot: Anti-Pollution Peer-to-Peer Botnets. In *Journal of Computer Networks, vol. 55*, 2011.

[33] T.-F. Yen and M. K. Reiter. Revisiting Botnet Models and Their Implications for Takedown Strategies. In *Proceedings of the 1st Conference on Principles of Security and Trust*, 2012.