Reliable Recon in Adversarial Peer-to-Peer Botnets

Dennis Andriesse
VU University Amsterdam
The Netherlands
d.a.andriesse@vu.nl

Christian Rossow
Saarland University, Germany
crossow@mmci.uni-saarland.de

Herbert Bos
VU University Amsterdam
The Netherlands
h.j.bos@vu.nl

ABSTRACT

The decentralized nature of Peer-to-Peer (P2P) botnets precludes traditional takedown strategies, which target dedicated command infrastructure. P2P botnets replace this infrastructure with command channels distributed across the full infected population. Thus, mitigation strongly relies on accurate reconnaissance techniques which map the botnet population.While prior work has studied passive disturbances to reconnaissance accuracy —such as IP churn and NAT gateways—, the same is not true of active anti-reconnaissance attacks. This work shows that active attacks against crawlers and sensors occur frequently in major P2P botnets. Moreover, we show that current crawlers and sensors in the Sality and Zeus botnets produce easily detectable anomalies, making them prone to such attacks. Based on our findings, we categorize and evaluate vectors for stealthier and more reliable P2P botnet reconnaissance.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software*; C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security

Keywords

Peer-to-Peer Botnet, Crawling, Reconnaissance

1. INTRODUCTION

The decentralized nature of Peer-to-Peer (P2P) botnet architectures eliminates the need for centralized Command-and-Control (C2) servers. As a result, P2P botnets are immune to traditional takedown strategies, designed to disrupt centralized C2 channels. Instead, takedown efforts against P2P botnets require large-scale distributed attacks, such as sinkholing, which target all bots in the network [28]. Due to the high resilience of P2P botnet architectures, many high-profile botnets have migrated from centralized to P2P networks. Among the most resilient of these are Sality [10] and P2P (GameOver) Zeus [2].

The recent takedown of GameOver Zeus underscores the resilience of botnets like these. This notorious banking trojan has been active since 2011, and has survived despite multiple intensive sinkholing efforts [28, 4], until finally being crippled in May 2014, following a massive collaboration between FBI, Europol, and several universities and malware analysis labs [5, 3].

Attacks against botnets like these are fundamentally based on knowledge about the composition of the botnet [28]. For instance, sinkholing attacks disrupt C2 connections between bots, and thus require a view of the botnet connectivity graph. Similarly, banks use P2P botnet mappings to identify infected customers, and several nations are currently setting up centralized repositories for reporting infected IPs [31].

Knowledge about the nodes in a botnet and the connections between them is obtained using intelligence gathering (reconnaissance/recon) techniques. Reliable recon techniques for P2P botnets are thus crucial for both invasive attacks (e.g., sinkholing) and non-invasive countermeasures (e.g., infection notifications). *All* recent P2P botnet takedowns required accurate recon, including the GameOver Zeus attack [5, 3, 22, 35, 19]. Note that connectivity graph-agnostic Sybil attacks, which could disrupt early DHT-based P2P botnets [6], are not effective against modern unstructured networks like Sality and Zeus, which use dense, dynamic connectivity graphs to propagate signed commands between bots.

To maximize the reliability of reconnaissance results, prior work has studied a myriad of passive disruptive factors, such as bots behind NAT gateways and IP address aliasing [28, 17]. In contrast, very little attention has been devoted in the literature to the hardening of recon against active disruption by botnet operators.

We believe this issue calls for closer analysis, as botmasters increasingly implement active anti-recon strategies in P2P botnets, largely in response to recent successful takedowns. These upcoming anti-recon strategies are aimed at compromising the intelligence gathered by malware analysts in P2P botnets, using attacks such as

(automatic) blacklisting [2], reputation schemes [10], and even DDoS against recon nodes [4]. Botmasters use anti-recon strategies to augment previously disabled botnets, and redeploy hardened versions of them, which are more difficult to take down. For instance, the Hlux botnet has already respawned three times, each time with additional hardening strategies [35, 19, 11]. These patterns suggest that next-generation P2P botnets will incorporate stronger anti-recon techniques. Because takedown and disinfection efforts against P2P botnets hinge on reliable reconnaissance, our work is aimed at proactively investigating the full extent of anti-recon implemented by P2P botmasters, and the range of possible defenses to safeguard reconnaissance in next-generation botnets.

We show that recon tools used by malware analysts to monitor high-profile botnets suffer from serious protocol deficiencies, making them easily detectable. Specifically, we infiltrate the Sality v3 and GameOver Zeus botnets by inserting passive sensors, which scan incoming traffic for anomalies and deviations from the botnet protocol. Additionally, we use active peer list requests to locate sensor nodes which do not actively initiate communication. (At the time of our analysis, the attack against GameOver Zeus had not yet been launched.) We identify 21 Zeus crawlers, 10 distinct Zeus sensors, and 11 Sality crawlers belonging to well-known malware analysis laboratories, network security companies, and CERTs. We find that all of these have shortcomings which make them easy to identify among the bot populations (200.000 bots for Zeus, and 900.000 for Sality) [28].

At first glance, it may seem that the situation could be remedied by eliminating syntactic protocol deficiencies in crawlers and sensors. We show that this is not so; even syntactically sound recon tools can be detected by anomalous in- or out-degrees. This is commonly true especially for crawlers, as they strive to contact as many bots in as short a time span as possible. To evaluate the magnitude of this problem, we design and implement a novel distributed crawler detection model, and show that it can automatically detect all crawlers in GameOver Zeus without false positives. We illustrate that the algorithm is applicable not only to Zeus, but also to other P2P botnets, including Sality. Based on these results, we propose and evaluate techniques to evade out-degree-based detection, such as rate limiting and distributed crawling, and we measure the impact of these techniques on crawling accuracy and completeness.

We also discuss an alternative reconnaissance strategy which has been widely proposed for use in P2P botnets, namely Internet-wide scanning [9]. We show that it is applicable to some P2P botnets, but is unfortunately not compatible with all P2P botnet protocols.

Summarizing, our main contributions are:

1. We identify and classify anti-recon attacks taking place in current P2P botnets, and generalize to potential attacks which could be implemented in future botnets.
2. We categorize recon strategies and their susceptibility to passive and active disruption.
3. We analyze the quality of crawlers and sensors used in Sality and Zeus, and classify major shortcomings.
4. We design and implement a syntax-agnostic crawler detection algorithm, and use it to analyze tradeoffs between reconnaissance stealthiness versus speed, accuracy, and completeness.
5. We discuss strategies for covert P2P botnet recon based on our results.

2. RECON IN P2P BOTNETS

Reconnaissance methods for P2P botnets can be divided into two classes, namely crawler-based and sensor-based methods (though hybrids are feasible [28]). This section introduces both of these classes, and compares their tradeoffs and resilience to passive disruption. We discuss active recon disruption in Section 3.

Throughout this paper, we consider a botnet to be a digraph $G = (V, E)$, where V is the set of vertices (e.g. bots), and E is the set of edges (e.g. is-neighbor connections between bots). We refer to the number of outgoing edges from a bot v as its out-degree $deg^+(v)$, and the number of incoming edges as in-degree $deg^-(v)$.

2.1 Crawlers

Crawler-based reconnaissance relies upon the peer list exchange mechanism available in all P2P botnets [28, 2, 10, 13, 33, 35]. In P2P botnets, every bot maintains a list of addresses of other peers it has been contacted by or has learned about from other bots. To maintain connectivity with the network despite the constant churn of bots that join and leave the botnet, peers regularly exchange (parts of) their peer list with their neighbors. Support for peer list exchanges is typically implemented through special request messages included in the botnet protocol, called peer list requests, which each bot can use to request a set of peer addresses from another. Crawlers abuse this mechanism by recursively requesting peer lists from all bots they learn about, starting from an initial bootstrap peer list (usually obtained from a bot sample).

Due to its simplicity and ability to rapidly request peers from many bots, crawling is a highly popular intelligence gathering method used by malware analysts [15, 28]. Nevertheless, crawlers have been reported to suffer from a myriad of inaccuracy problems. For instance, address aliasing can occur with bots that use dynamic IP addresses, leading to significant botnet size overestimations if the crawling period is too long [17]. In addition, crawlers cannot contact and verify the liveness of non-externally reachable (non-routable) bots, such as

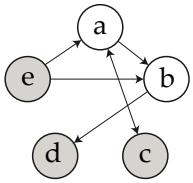


Figure 1: An example botnet connectivity graph. An arrow from node a to node b indicates that a knows b. Non-routable nodes are shaded.

bots protected by a firewall or NAT gateway [17, 28]. This is a significant shortcoming, since the percentage of non-routable bots can range up to 60–87% [28].

Figure 1 shows the inaccuracy that may result from the inability of crawlers to verify non-routable bots. The figure depicts a connectivity graph containing bots $a, \ldots, e$, of which bots $c, \ldots, e$ are non-routable (shaded). A crawler can only directly contact the externally reachable (routable) bots a and b; non-routable bots are only discoverable if they actively advertise themselves.

The only way for a crawler to learn about bots $c, \ldots, e$ is by relying on the peer lists returned by a and b. The addresses returned in these peer lists may be outdated, and the lists may include multiple aliases for bots with dynamic IPs. The crawler cannot be sure of this, and cannot actively establish contact with bots $c, \ldots, e$ to verify their existence. On the other extreme of the scale, the botnet protocol may not allow non-routable bots to push themselves into the peer lists of other bots at all, making it impossible for crawlers to find non-routable bots. As an example, node e in Figure 1 has no incoming edges from any routable bot, making it undetectable to crawlers regardless of the protocol.

2.2 Sensors

In contrast to crawlers, sensors are largely passive. When injecting a new sensor node into a botnet, its address is actively announced (as is done for new bots joining the network) until a sufficient number of bots learn about the existence of the sensor. After that, the active announcement is ceased, and the sensor relies upon peer list exchanges between bots to propagate its address. Sensors map the network by waiting for bots to contact them, instead of actively contacting bots.

Compared to crawlers, sensors achieve better coverage of routable bots [28]. In addition, sensors can find and verify all non-routable bots which contact them after learning the sensor addresses from other bots. This allows sensors to more precisely and reliably map the nodes in the botnet than crawlers [28].

Sensors are currently not as widely used by malware analysts as crawlers are. In part, this is due to the larger implementation and reverse engineering effort required to build and maintain a sensor. Since most botnets have mechanisms to evict unresponsive or incorrectly responding peers from peer lists, sensors must implement most or all of the botnet protocol to avoid being evicted. In contrast, crawlers only need support for peer list requests and responses. Furthermore, sensor injection can be complicated by botnet resilience measures like reputation schemes [10] and non-persistent peer list entries [36].

In contrast to crawlers, sensors map only the nodes in a botnet, but not the connectivity information. However, since sinkholing attacks overwrite peer list entries, they require connectivity information to determine which entries to target. To obtain this information, sensors must be expanded to request peer lists from bots which contact them. Compared to crawlers, sensors which are augmented with active peer list requests have the advantage that they can even reach the majority of non-routable bots through NAT punch holes.

3. ANTI-RECON ATTACKS

Recon tools like crawlers and sensors are open to a range of attacks which impair intelligence gathering. This section categorizes the anti-recon measures which we have observed in the wild in all major P2P botnets since 2007 [28]. We summarize our findings in Table 1, and categorize anti-recon into four categories: (1) deterrence, (2) passive attacks (i.e., blacklisting), (3) active disinformation attacks, and (4) retaliation attacks.

3.1 Deterrence

Deterrence encompasses P2P botnet protocol characteristics designed to complicate recon. This includes measures meant to deter crawling or sensor injection.

As shown in Table 1, deterrence is currently the most common class of anti-recon: all major P2P botnets include some form of recon deterrence. Specifically, all botnets except Storm include an IP filter which prevents the injection of multiple sensors on a single IP [10, 25, 35, 19, 6, 13, 34, 33]. Zeus goes further than this, disallowing more than a single peer list entry per /20 subnet [2].

All botnets also include a form of information limiting designed to slow crawling, usually by returning only a small set of peer list entries at once. This is most extreme in Hlux, where at any moment there is only a small list of around 500 relay nodes (externally reachable bots) circulating in the network [19]. Another form of information limiting is clustering, observed in Zeus, Storm, and Hlux [2, 13, 35]. Zeus and Storm restrict the returned peer list entries based on an XOR metric, returning only entries "close" to the requester according to this metric. Hlux clusters the bot population around a small core of relay bots shared in peer lists.

Finally, some bots also feature more specialized recon deterrence. For instance, Sality actively tries to prevent sensor injection by using a reputation scheme based on a goodcount, which reflects how well-behaved peers

	Deterrence					Blacklisting		Disinformation	Retaliation
	IP filter	**Reputation**	**Info limit**	**Flux**	**Clustering**	**Auto**	**Static**	**Spurious IPs**	**DDoS**
Zeus	By /20	-	Peer list	-	XOR metric	By rate	Manual	-	After attack
Sality	By IP	Goodcount	Peer list	-	-	-	-	-	-
ZeroAccess	By IP	-	Peer list	Peer push	-	-	Manual	Junk	-
Kelihos/Hlux	By IP	-	Relay list	-	Relay core	-	-	-	-
Waledac	By IP	-	Relay list	-	-	-	-	-	-
Storm	-	-	Proximity	-	XOR metric	-	-	Rogue	After attack

Table 1: Anti-recon measures observed in P2P botnets.

have been in the past [10]. Sensors are only propagated to other bots if they achieve a high goodcount first. Additionally, ZeroAccess prevents injection of persistent links to sensors by pushing a continuous flux of peer list updates, constantly overwriting the full peer list of each routable bot [25].

3.2 Blacklisting

Blacklisting is a passive attack, designed to starve crawlers and sensors of information by prohibiting bots from communicating with them. It is already used in ZeroAccess, and in particular also in GameOver Zeus, which features two distinct blacklisting mechanisms [2]. (1) Each bot binary is shipped and periodically updated with a hardcoded blacklist of IPs which the botmasters identified on the network due to anomalous behavior. (2) In recent versions of Zeus, an automatic blacklisting mechanism was introduced, which is a rudimentary approach to identify and block crawlers and nodes attempting to poison the network. Each bot tracks the request frequencies of its peers, and blocks peers sending many requests in quick succession. To prevent blocking of multiple NATed bots using the same IP, the maximum frequency is high enough to be circumventable by relatively efficient crawlers. Nevertheless, this mechanism is sufficient to thwart naive crawlers. Blocked IPs become unusable for malware analysis not only on the botnet in question, but also on other botnets, as (hardcoded) blacklists are often publicly visible.

3.3 Disinformation

Disinformation attacks actively supply crawlers and sensors with spurious peer list entries containing forged addresses, in order to divert them from the main bot population. This is problematic especially for crawlers, as they lack the ability to verify non-routable bot addresses. The attack can be expanded to thwart sensors by leading them into a "shadow botnet", containing a set of responsive peers which are isolated from the main bot population. Disinformation attacks can cause collateral damage by polluting lists of infected addresses reported to ISPs, and even rerouting sinkholing attacks to shadow nodes or uninfected hosts.

We have observed (possibly unintended) disinformation in ZeroAccess, where peer lists frequently include junk addresses, such as reserved or unused IP addresses. In addition, Storm is known to have contained many rogue nodes, which also served to pollute exchanged peer lists [6]. Overall, current P2P botnets do not yet engage in disinformation on a large scale, but there is a real risk that future P2P botnets may engage in this strategy, especially as it has already been explored in detail in prior work [37]. We believe recon tools should implement measures to prevent this kind of attack, as it is quite difficult to detect once deployed in full-scale.

3.4 Retaliation

Retaliation is another category of active attack, in which botmasters take action to disable or compromise hosts used for reconnaissance. Retaliatory actions include denial-of-service attacks, as used by the GameOver Zeus botmasters in response to sinkholing attempts [4], and active infiltration or exploitation of hosts used for recon. Repeated denial-of-service attacks have also been observed in the Storm botnet, in response to takedown/infiltration attempts targeted at it [6].

4. RECON ANOMALIES

This section analyzes the stealthiness of crawlers and sensors used in GameOver Zeus and Sality v3. Unstealthy reconnaissance tools expose themselves to an increased risk of the attacks discussed in Section 3. Section 4.1 discusses protocol anomalies found in crawlers, while Section 4.2 discusses sensor anomalies. In Section 4.3, we design a syntax-agnostic algorithm to detect crawlers based only on network coverage.

4.1 Crawler Protocol Anomalies

We base our crawler analysis on data gathered by 512 sensor nodes in the Zeus network (before it was sinkholed), and 64 sensors in Sality (the number is limited by Sality's peer management scheme and our IP range). We announced these sensors for two days, and then ran them passively for three weeks, until the combined sensor in-degrees matched botnet size estimates [28], ensuring reachability by all crawlers. Our sensor implementations included only protocol logic, and no malicious logic, and we verified the legality of our tactics with law enforcement officials.

We identified crawlers using protocol-specific anomaly detection on our sensor data, looking for peers with deviations from normal bot behavior. To define nor-

	c1	c2	c3	c4	c5	c6, c11
LOP range	✓	✓	✓	✓	✓	✓
Port range		✓	✓	✓	✓	✓
Random ID Version		✓	✓	✓		✓
Hard hitter	✓	✓	✓	✓	✓	✓
Protocol logic	✓	✓			✓	✓
Encryption						
Coverage (%)	69	100	100	100	100	100

Table 2: Defects found in Sality crawlers.

mal behavior, we reverse engineered Zeus and Sality to establish a ground truth. Next, we cross-verified our list of detected crawlers to ensure that crawlers from our anti-malware industry contacts were all correctly detected. While it is impossible to obtain a complete ground truth of crawlers in a live botnet, we established the significance of our results by attributing analyzed crawlers to large malware analysis companies, network security companies, CERTs, and (academic) researchers. We attributed crawlers using WHOIS data, protocol-specific information (such as bot IDs revealing company names), and inquiries with contacts in the community. We informed affected parties via closed mailing lists.

We study only well-functioning crawlers which cover at least 1% of the bot population ($\geq$ 50 messages to our sensors), with the addition of one open-source Zeus crawler belonging to a large network security company. The resulting list of crawlers found by our anomaly detection includes 21 Zeus crawlers and 11 Sality crawlers. We summarize the defects found in these crawlers in Table 2 for Sality, and Table 3 for Zeus (anonymized to avoid revealing IPs used by malware analysts). Note that 6 of the 11 Sality crawlers are grouped together into a single column, as these were all running in the same subnet and exhibited the same characteristics (i.e., multiple instances of the same crawler).

The rest of this section discusses the results from Tables 2 and 3 in detail. We organize the results into several classes of common defects, each of which undermines crawler stealthiness, and increases attack vulnerability.

4.1.1 Range Anomalies

Crawlers with range anomalies exhibit static or constrained values for message fields that should be randomized. Additionally, range anomalies can occur when crawlers use random values for non-random fields. Our results show that range anomalies are the most common class of defects in Zeus crawlers. We found at least one range anomaly in 20 of the 21 analyzed crawlers.

The Zeus message header contains several fields which are normally randomized. These include a random byte at the beginning of each message, the Time to Live (TTL) field which is randomized when unused, and the Length of Padding (LOP) field, which indicates the length of the random padding at the end of the message. Furthermore, a random session ID is generated for each request-response pair. In 14 crawlers, the padding length was constrained, possibly to reduce bandwidth usage by limiting the number of padding bytes at the end of each message. Additionally, static or constrained random bytes and TTL values each occurred in 10 crawlers, and 11 crawlers used static session IDs, or rotated between a small number of session IDs.

On the other hand, we found that 3 Zeus crawlers used random values for the source ID field on each message, which indicates the unique identifier of the bot that sent the message. Although a small variation in source ID per IP address is normal, and can indicate multiple bots behind a NAT gateway, these crawlers used over 1000 different source IDs, making them highly detectable.

Just as in Zeus, normal Sality messages end in a random amount of padding. Nevertheless, in all but one of the analyzed Sality crawlers, the padding length was set to a fixed value, and in the remaining crawler the padding length was constrained to reduce bandwidth usage. Additionally, 10 of the 11 analyzed crawlers sent messages from a fixed port, while ordinary Sality bots use a randomized port per message exchange.

Some types of Sality messages include a bot identifier, which normally does not change while the bot remains up. All of the crawlers we analyzed adhered to these semantics, and did not change their identifiers between messages exchanged with our sensors.

4.1.2 Entropy Anomalies

Entropy anomalies occur when multi-byte fields that normally contain high-entropy content are set to non-random values. In the Zeus protocol, such fields include the source ID and session ID, which are SHA-1 hashes, and the random padding bytes at the end of a message. We found 3 crawlers with low-entropy session IDs, 5 crawlers with non-random padding bytes, and 5 crawlers with low-entropy source IDs. The crawlers with low-entropy source IDs occasionally used identifiers with ASCII bytes representing the name of the company or individual running the crawler.

In contrast to Zeus, Sality uses randomly chosen integers instead of hashes as (non-global) bot identifiers. While Zeus crawlers often use low-entropy identifier strings, all Sality crawlers use seemingly random integer IDs, without any entropy anomalies.

4.1.3 Invalid Encryption

In 7 of the analyzed Zeus crawlers, we encountered corrupted messages that contained irrational data for all message fields. These corrupted messages were interspersed with correctly encoded messages. It appears that these crawlers occasionally select incorrect keys

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	c21
RND range	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓											
TTL range	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓											
LOP range	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓				
Session range	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓									
Session entropy	✓	✓								✓											
Random source																		✓	✓	✓	
Source entropy		✓									✓	✓	✓				✓				
Padding entropy	✓	✓	✓	✓	✓																
Abnormal lookup	✓	✓	✓	✓	✓			✓			✓	✓				✓	✓				
Hard hitter	✓							✓			✓	✓	✓	✓	✓			✓	✓		
Protocol logic	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓						✓
Encryption	✓										✓	✓	✓	✓	✓	✓					
Coverage (%)	90	82	85	75	84	20	53	62	1	8	92	44	85	92	92	88	54	87	86	2	27

Table 3: Defects found in GameOver Zeus crawlers.

to encrypt outgoing messages, preventing our sensors from decrypting these messages. In GameOver Zeus, the unique identifier of each bot is used as the key to encrypt messages towards that bot. Thus, we suspect that the crawlers in question do not correctly keep track of the identifier of each bot they find. We did not encounter any invalid encryption in Sality crawlers.

4.1.4 Protocol Logic Anomalies

As mentioned in Section 2, an advantage of crawlers is that they avoid the need to implement the full P2P protocol. However, taking this simplification too far results in crawlers that manifest incorrect protocol logic. This was the case for 17 of the analyzed Zeus crawlers. In most cases, these crawlers simply sent large amounts of peer list requests, without regard for any other message types used by normal bots, such as messages used to exchange commands and updates.

Additionally, many crawlers used abnormal values for the lookup key field used in Zeus peer list requests. This field includes a Zeus identifier which is used by the receiving peer as a metric to determine which peers are sent back in the peer list response. Normal bots always set this field to the identifier of the remote peer. In contrast, many crawlers randomize the lookup key to increase the range of crawled peers.

Similarly to Zeus, most Sality crawlers (9 out of 11) used incorrect message sequences. Typically, these crawlers sent repeated peer list requests to the same bot, without interspersing any of the other normal message types, such as URL pack exchanges. Another common defect is the use of incorrect version numbers in the Sality message headers sent by crawlers. While all of the crawlers used a correct major version number, only 2 of them also used a valid minor version number.

4.1.5 Request Frequency Anomalies

In an effort to rapidly gather as much intelligence as possible about the botnet connectivity graph, 9 of the analyzed Zeus crawlers sent repeated peer list requests at high frequencies. These hard-hitting crawlers clearly contrast with real bots, which exchange only one peer list request per neighbor before returning to a suspended mode. Such request-suspend behavior is seen in many botnets [28], with a suspend period of 30 minutes for Zeus and 40 minutes for Sality, making hard-hitting crawlers highly anomalous.

In Sality, request frequency anomalies are even more common than in Zeus, occurring in all of the analyzed crawlers. This is because Sality bots only support the exchange of a single peer list entry at once, while Zeus bots return up to 10 entries in a single peer list response. Additionally, Sality peer lists are much larger than Zeus peer lists, typically containing around 1000 entries, while Zeus peer lists are limited to 150 entries (and rarely contain over 50 entries). To obtain a reasonable coverage of a bot's peer list, Sality crawlers are therefore obliged to send this bot multiple peer list requests. Sending these in quick succession, as all of the analyzed crawlers do, results in clear request frequency anomalies.

4.2 Sensor Protocol Anomalies

In addition to our crawler analysis from Section 4.1, we also analyzed the sensors active in GameOver Zeus and Sality during our monitoring. We found Zeus sensors belonging to 10 organizations (grouped by subnet) by analyzing the GameOver Zeus connectivity graph, and looking for nodes with high in-degrees. Since sensors aim to attract as many bots as possible, they are expected to have high in-degrees. To create a view of the connectivity graph and node in-degrees, our sensors sent active peer list requests to every bot that contacted them. After identifying high in-degree nodes, we probed each of these for anomalous responses to identify (defective) sensors. This is necessary as, unlike high out-degrees, our data shows that high in-degrees occur in hundreds of apparently legitimate bots. As in our crawler analysis, we cross-verified the identified sensors with a list of sensors known from industry contacts.

Our results show that the Zeus sensors suffer from the same shortcomings as current crawlers, including

range anomalies, entropy anomalies, and protocol logic anomalies. In addition to these anomalies, we identified several sensor-specific anomalies. Specifically, all of the analyzed sensors failed to return the protocol-mandated list of proxy bots (used as data drops in GameOver Zeus) when requested. In addition, all but 3 of the sensors responded to peer list requests with empty peer lists, which is highly unusual behavior. Furthermore, all of the sensors which returned non-empty peer lists served duplicate peers in order to promote sinkholes or sensors, a behavior never displayed by legitimate bots. Finally, none of the sensors provided a correct implementation of the Zeus update mechanism; only 3 sensors reported valid and recent version numbers, and none of the sensors responded to update requests used to exchange binary and config files.

Although these anomalies allow current sensors to be detected, sensors are not inherently detectable. Specifically, although sensors generally have high in-degrees, our data shows that equally high in-degrees also occur for well-reachable legitimate bots, making high in-degree alone an insufficient metric for reliable sensor detection. Furthermore, P2P botnet protocols cannot be easily designed to expose sensors by limiting the in-degree of bots. This is because, by the degree sum formula[1], limiting the in-degree of bots implies limiting the out-degree as well, thus impairing the connectivity of the bots. In our analysis, we were unable to identify any sensors in Sality, precisely because no nodes with unusually high in-degree were present, and all high in-degree nodes responded correctly to probes for all packet types, including URL pack exchanges and peer exchanges. Thus, if any sensors are present in the Sality network, these cannot be detected in any straightforward way. In contrast, efficient crawlers tend to have unusually high out-degrees, as we show in Section 4.3, requiring special measures to avoid detection. To further investigate the detectability of crawlers by out-degree, we design a distributed crawler detection model in Section 4.3. Subsequently, we categorize crawling techniques to evade this model in Section 5, and implement and evaluate our detection model and evasive techniques in Section 6.

4.3 Network Coverage Anomalies

Even syntactically sound crawlers are detectable due to their tendency to contact many peers in a short timespan, in an attempt to quickly map the network. This behavior is visible in Tables 2 and 3. Sality and Zeus crawlers cover up to 100% and 92% of our sensors, respectively. Furthermore, nearly all crawlers cover at least 20% of our sensors, and most crawlers cover 50% or more. In contrast, ordinary bots cover only a small fraction of the botnet, due to their limited peer list sizes and evolution rates [10, 2, 28].

To evaluate the extent of this problem, and the effectiveness of our proposed countermeasures, we design and implement a syntax-agnostic crawler detection algorithm that identifies crawlers based on their network coverage. We chose to implement a distributed version of the algorithm to show that crawler detection is possible even without requiring centralized components in P2P botnets. The algorithm is scalable and Byzantine-tolerant, to allow crawler detection even in the presence of adversarial nodes (which malware analysts may inject to subvert the algorithm). To prevent impersonation attacks, the algorithm assumes a non-spoofable transport layer, such as TCP. This section provides an overview of our algorithm, while Sections 5 and 6 discuss improved crawling techniques and their effectiveness in evading our detection algorithm, respectively.

Note that this is just one from a range of out-degree-based crawler detection methods. Alternative implementations, even centralized ones, can equally well detect crawlers with anomalous out-degrees, and completely circumvent the risk of subversion by Sybils. Centralized implementations fit well with the hybrid structure of many current P2P botnets, such as Zeus and Kelihos, where bots periodically report to a higher layer of centralized servers [2, 35, 19]. Therefore, this paper focuses on our findings regarding the efficacy of stealthy crawling techniques, rather than technical details of our particular detection algorithm. Due to space limitations, we defer a discussion of such details, and the Byzantine-tolerance of our algorithm, to a technical report [1].

Our algorithm is based on periodic crawler detection rounds, with the period depending on the time needed to comprehensively crawl the botnet. This time depends on the botnet architecture, diurnal pattern, and protocol, and is 24 hours for Zeus and Sality, as shown in Section 6 and prior work [28]. Crawling for less than 24 hours misses part of the population, while crawling longer pollutes results due to infection churn and IP churn [28]. Our tests in Section 6 use hourly detection rounds to detect crawlers well before they cover a full diurnal period. The algorithm assumes that each bot has a random identifier, generated at infection time. Each round consists of the following operations.

Detection round announcement The botmaster pushes a round announcement (signed and timestamped to prevent replays) to a random bot, which then propagates to all bots using gossiping (a technique also used in Zeus and ZeroAccess [28]). We use push-based gossiping to reach only routable peers, excluding non-routable bots (never reached by crawlers) for scalability. The bots partition themselves into 2^g groups by sampling g bit positions (specified in the announcement) from their

[1]In a directed graph $G = (V, E)$, with V the set of nodes and E the set of edges, the degree sum formula states that $\sum_{v \in V} deg^+(v) = \sum_{v \in V} deg^-(v) = |E|$, where $deg^+(v)$ and $deg^-(v)$ are the out-degree and in-degree of node v.

identifiers. Each group contains the bots with identical bit values at these g positions, and a per-group leader is assigned in the announcement, creating 2^g tree-shaped overlay networks rooted at the group leaders.

Hardhitter aggregation Every bot contacts its leader and reports all IPs that requested its peer list within a configurable history interval. This interval must cover multiple detection rounds, or else crawlers can avoid detection by contacting only a limited set of bots per round (see Section 6). Leaders aggregate the IPs (including their own history), and flag IPs reported by at least a configurable threshold fraction of the group as suspicious. Section 6 discusses how to set the threshold to minimize false positives/negatives.

Crawler voting Leaders collectively vote for suspected crawler IPs, and classify those IPs which receive a majority vote as crawlers. Majority voting makes the algorithm tolerant of adversarial leaders which unjustly blacklist non-crawler IPs, or whitelist true crawlers [1]. As leader selection is random, adversarial nodes are unlikely to dominate the leader population unless a large fraction of the bot population consists of Sybils (anti-Sybil strategies are discussed in prior work [29, 7]).

Crawler propagation All bots retrieve the list of crawlers from n randomly chosen leaders, and filter crawlers reported by a majority of the leaders. This limits the scope of faulty results reported by adversarial leaders. Bots can expect reliable results if $|A| < n \times m$, where $|A|$ denotes the number of adversarial leaders and m the fraction of leaders required in a majority.

5. STEALTHY CRAWLING TECHNIQUES

In this section, we propose methods to improve the reliability and stealthiness of P2P botnet crawling. These strategies are specifically aimed at evading out-degree- or request-frequency-based detection, as described in Sections 4 and 4.3. We measure the efficacy of these strategies, and their impact on crawling efficiency, in Section 6. The methods proposed in this section apply to all variants (distributed and centralized) of out-degree-based crawler detection algorithms.

5.1 Contact Ratio Limiting

A straightforward way to limit the out-degree of crawlers is to limit the set of bot IPs they contact. For instance, crawlers which contact only half of the bots (a contact ratio of 1/2) are expected to still obtain a reasonably complete view of the network, as the addresses of the excluded bots are returned in the peer list responses of the remaining bots. A limitation of this approach is that it further exaggerates the node verification problem described in Section 2. Contact ratio-limited crawlers are not only unable to verify the authenticity of non-routable nodes, but also cannot verify the excluded routable bots. Furthermore, we show in Section 6 that to evade detection using contact ratio limiting, crawlers must use very low contact ratios of 1/16, or even 1/32. As we show, this reduces the completeness of the gathered intelligence.

5.2 Request Frequency Limiting

As described in Section 4.1, many crawlers send multiple peer list requests in quick succession to maximize their coverage of each bot's peer list. Stealthy crawlers should limit their request frequencies to avoid being identified as hard hitters (see Section 4.1.5). We measure the impact of request frequency limiting in Section 6, and show that the crawling efficiency obtained by frequency-limited crawlers depends on protocol-specific factors, like the selection strategy for returned peers, and the number of entries per peer list response.

5.3 Distributed Crawling

By distributing their egress traffic over multiple source addresses, crawlers can reduce the risk of detection by IP-centric sensors. This strategy encompasses several techniques. (1) Crawlers can be (virtually) distributed over multiple addresses, crawling only a limited subset of bots per source address. (2) Addresses used for crawling can be rotated periodically, with a rotation frequency such that a new address is selected before exceeding the per-address detection threshold.

As we show in Section 6, distributed crawlers must use IPs from distinct subnets to reliably avoid detection. The crawler detection algorithm in our experiments can reliably detect crawlers with IPs distributed over a /20 subnet, or with per-address crawling traffic limited by a factor of up to 1/32. Thus, distributed Zeus or Sality crawlers require addresses from at least 32 distinct /20 subnets, or a single /16 network block.

5.4 Anonymizing Proxies

Crawlers could attempt to evade detection by using proxy servers or an anonymizing network such as Tor [8] to obtain rapidly changing IP addresses. This approach can be effective, but unfortunately does not blend well with current anonymizing services, due to several issues. (1) Effective crawlers simultaneously open thousands of connections, causing connection tracking and scalability problems with non-dedicated proxy services. Furthermore, the bandwidth and latency performance of the Tor network is known to be constrained, resulting in slowly converging, and thus inaccurate, crawling results [8, 26]. (2) The IPs used by anonymizing proxies and Tor exit nodes are often publicly known, and can thus be easily blocked by botmasters [12, 23]. A more effective option might be to set up a dedicated anonymizing proxy service, though this would require one or more large dedicated network address blocks as well as strong secrecy measures to avoid these from leaking out.

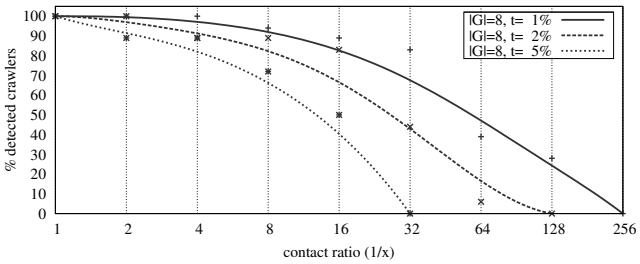


Figure 2: Crawlers detected in 24 hours for $|G| = 8$ and $t \in \{1\%, 2\%, 5\%\}$. The contact ratio simulated on the crawler traffic varies over x.

6. STEALTHY CRAWLING EVALUATION

In this section, we evaluate the detectability of in-the-wild Zeus crawlers by network coverage, using the crawler detection algorithm introduced in Section 4.3. Furthermore, we measure the effectiveness of the evasive crawling techniques introduced in Section 5, and their tradeoffs in crawling speed and completeness. We do this by simulating (to ensure repeatability) the effects of these evasive strategies on traffic we logged from real-world Sality and Zeus crawlers.

6.1 Crawler Detection Accuracy

A full-scale crawler detection algorithm would run distributed over all routable nodes in a botnet. Since we cannot deploy such an experimental setup, we instead ran our crawler detection experiments on the 512 sensor nodes which we injected into the GameOver Zeus botnet. As mentioned in Section 4.1, none of the sensors contained any malicious logic. We performed all our experiments before the recent GameOver Zeus takedown. Our sensor nodes logged all received requests over a total test period of 24 hours. We chose a 24-hour period to account for a full diurnal cycle without suffering from duplicate results due to address aliasing caused by dynamic IP addresses. We then performed our experiments with varying configuration parameters on the logged traffic, to ensure that any detection differences were a result of the configuration parameters rather than churn in the bot and crawler populations. During our test period, 18 of the crawlers from Table 3 (Section 4.1) were active, which we used as a ground truth. The rest of this section describes our measurements of the effectiveness of the crawling techniques proposed in Section 5 to evade out-degree-based detection: (1) contact ratio limiting, and (2) address distribution. (We do not evaluate frequency limiting here as it does not pertain to the out-degree of crawlers.)

6.1.1 Contact Ratio Limiting

The detection results of our GameOver Zeus tests are shown in Figure 2 (detected crawlers) and Table 4 (detection rate vs. false positives). We used a per-bot request history of 24 hours, meaning that crawlers are detected if they cover a significant fraction of the bot population in 24 hours. In our experiments, we randomly partitioned our sensors into 8 groups per detection interval. In a full-scale implementation, this arrangement would split the Zeus population of 200.000 bots [2] into groups of 25.000 bots (including non-routable bots) per leader. We denote the number of groups as $|G|$ (the magnitude of the set of groups G) in Figure 2. We vary the per-group detection threshold, denoted as t in the figure, to illustrate the tradeoffs between detection accuracy and false positives. This threshold is the percentage of bots per group that a crawler must cover to receive a vote from that group. We simulated the detection performance for contact ratio-limited crawlers by excluding crawler requests to a varying subset of our sensors (the x-axis in Figure 2).

As shown in Figure 2, a per-group detection threshold of 1% allows 28% of crawlers to be detected even if these limit their contact ratio to only 1/128 bots. However, this comes at a cost of 119 falsely identified crawlers (denoted #FP in Table 4), most of which are actually sets of NATed bots sharing a single IP. Increasing the threshold to 2% reduces the number of false positives to 13, but also limits the detection accuracy to 44% for a 1/32 contact ratio, and 6% for a 1/64 contact ratio. In our tests, the ideal threshold is 5%, allowing 100% of crawlers to be detected without false positives if no contact ratio limiting is used in the crawlers. The accuracy degrades gracefully to 50% for crawlers with a 1/16 contact ratio, meaning that stealthy crawlers may contact at most 1 in every 16 bots to evade detection with 50% probability. When increasing the threshold further to 10%, the detection rate degrades to 89% (2 false negatives) if no contact ratio limiting is used, and to 50% when a contact ratio of 1/8 is applied.

6.1.2 Address Distribution

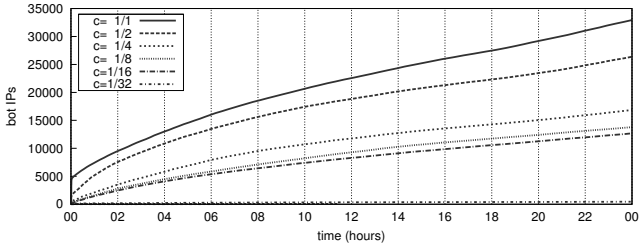
To determine the range of addresses needed by distributed crawlers, we also measured the effectiveness of crawler detection using subnet aggregation. Using the ideal detection threshold of 5%, our experiments show that crawlers can be detected with 100% accuracy (no false positives) using subnet aggregation ranging from /32 (per-IP detection) to /20 (aggregation per subnet of 4096 hosts). Starting from a /19 filter, the crawler detector reports 110 false positives, caused by multiple infections within the same subnet. This indicates that to evade detection, address-distributed crawlers for Zeus must take each address from at least a distinct /20 subnet. As shown by our contact ratio experiments, a coverage reduction of a factor 32 is required for Zeus crawlers to avoid detection, so that the address-distributed crawlers must take their addresses from 32 distinct /20 subnets, or alternatively from a single /16.

t	$\|G\|$	#FP	$D_{1/1}$	$D_{1/2}$	$D_{1/4}$	$D_{1/8}$	$D_{1/16}$	$D_{1/32}$	$D_{1/64}$	$D_{1/128}$	$D_{1/256}$
1	8	119	100	100	100	94	89	83	39	28	0
2	8	13	100	100	89	89	83	44	6	0	0
5	8	0	100	89	89	72	50	0	0	0	0
C_{Zeus}	N/A	N/A	100	80	52	42	38	2			
C_{Sality}	N/A	N/A	100	90	74	44	27	16			

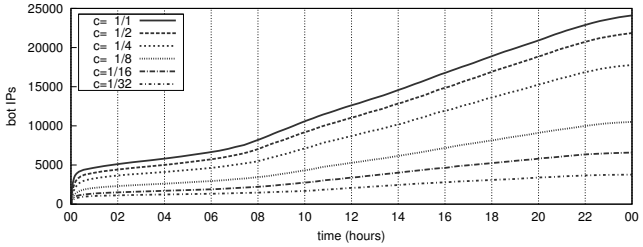
Table 4: False positives vs. detected crawlers for $|G| = 8$ and $t \in \{1\%, 2\%, 5\%\}$.
D_c = % detected crawlers for contact ratio c. C = % bots covered by crawler using contact-ratio limiting (relative).

6.1.3 Sality Crawler Detection

To complement our Zeus experiments, we attempted to repeat our crawler detection tests in the Sality network. Unfortunately, the peer list size and peer exchange properties of Sality prohibit small-scale tests of our algorithm. In Sality, bots maintain a peer list of 1000 entries, and exchange peers based on a responsiveness-based reputation metric [10]. To obtain meaningful evaluation results, it is necessary to run significantly more sensors than fit in a Sality peer list. Otherwise, sensors are highly likely to be present in the peer lists of many legitimate bots, making these bots indistinguishable from crawlers. While our 512 sensor nodes were sufficient to prototype our algorithm for Zeus, where typical bots only maintain around 50 peer list entries [2], a Sality prototype would require several thousand IP addresses, which we were unable to obtain for our experiments. Nevertheless, as Sality's reputation mechanism and its limit of a single entry per peer exchange restrict the out-degree of Sality bots, we expect that Sality crawlers are highly susceptible to full-scale crawler detection (as could be implemented by the Sality botmasters).



(a) Zeus.



(b) Sality.

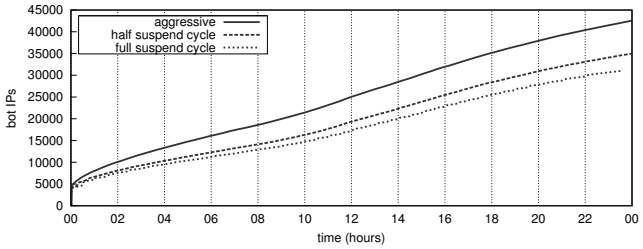
Figure 3: Bots crawled in 24 hours for varying contact ratio.

6.2 Stealthy Crawling Performance

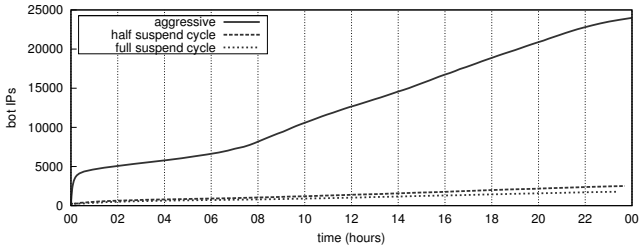
To evaluate the reconnaissance performance of the stealthy crawling techniques proposed in Section 5, we implemented contact ratio and frequency-limited crawlers for both GameOver Zeus and Sality. (We do not evaluate address-distributed crawlers, as these should suffer no degradation in crawling performance.) For both Zeus and Sality, we ran all of the crawling tests in parallel for 24 hours, to ensure that performance differences did not result from churn in the bot population. The contact ratio-limited crawlers only contacted a deterministically restricted fraction of bots, based on the bot identifier. Figure 3 graphs the number of bots detected over time by contact ratio-limited crawlers, while Figure 4 shows the results for frequency-limited crawlers.

6.2.1 Contact Ratio Limiting

As shown in Figure 3 and Table 4, the number of peers found steadily drops as the crawler contact ratio decreases. At a contact ratio of 1/2, crawling performance is still good, finding 80% of the Zeus peers found in a full crawl, and 90% of the Sality peers. However, as can be seen in Figure 2, this contact ratio still allows



(a) Zeus.



(b) Sality.

Figure 4: Bots crawled in 24 hours for varying request frequency.

89% of the crawlers to be detected. Reducing the contact ratio further causes a rapid decline in completeness. At a contact ratio of 1/4, our crawler finds 74% of Sality peers, and only 52% of Zeus peers. When reducing the contact ratio to 1/16, only 38% of the Zeus bots, and 27% of the Sality bots are found by our crawler, while this contact ratio still leads to a probability of 50% that crawlers are detected. These results show that contact ratio limiting achieves crawler stealthiness only at a high cost in crawling completeness. Note that it is not relevant whether or not our initial crawls reached the full bot population. Rather, the results serve only to show the relative coverage degradation which results from contact ratio limiting.

6.2.2 *Frequency Limiting*

Figure 4 shows crawling results for aggressive crawling, where the suspend-request cycle used by normal bots is not respected, as well as for crawls adhering to the suspend period between requests. We show results for crawlers using a full suspend cycle (30 minutes for Zeus and 40 minutes for Sality), as well as a half suspend cycle. The results are highly dependent on the protocol of the crawled botnet. For Zeus, even crawlers adhering to a full suspend cycle achieve reasonable efficiency, finding 74% of the bots found by the aggressive crawler. There are two main reasons for this. (1) Zeus returns 10 peers per peer list response, and the peer lists of typical bots contain only around 50 entries. This makes it possible to cover a larger fraction of each bot's peer list in a single suspend-request cycle than for Sality, which features more stringent constraints. (2) Due to the frequency-based automatic blacklisting mechanism used in Zeus, even our aggressive crawler is rate limited to avoid being blacklisted. This reduces the gap between the aggressive crawling results and the suspend cycle-adherent results. For Sality, the impact of frequency limiting is severe. Only 11% of the bots are found using a half suspend cycle, and only 7% when adhering to a full suspend cycle. This is because the peer lists of Sality bots contain up to 1000 entries, while only one entry is returned per peer list response. Note that even frequency-limited crawlers are prone to out-degree based detection. Thus, frequency limiting must always be used in unison with out-degree limiting.

7. INTERNET-WIDE SCANNING

Recent advances have enabled fast Internet-wide scanning in practical tools such as ZMap [9]. These tools work by probing large subsets of the public IP address space, in order to find hosts that have a particular property (i.e., a security vulnerability, OS version, etc.), as evidenced by their response to the probes. Internet-wide scanning has been proposed as an alternative reconnaissance method for finding botnet servers and bots [24].

	Fixed port	Probe msg	Susceptible
Zeus	✗	✗	✗
Sality	✗	✓	✗
ZeroAccess	✓	✓	✓
Kelihos/Hlux	✓	✓	✓
Waledac	✗	✓	✗
Storm	✗	✓	✗

Table 5: Susceptibility of P2P botnets to Internet-wide scanning.

Additionally, it has been used in practice to discover bots in the ZeroAccess P2P botnet [20].

Internet-wide scanning can be an efficient approach to node discovery in P2P botnets, but unfortunately it does not generalize well to all protocols. This is due to several factors. (1) Many P2P botnets use a large port range, where each bot listens on only a single port from this range. For instance, Zeus bots choose ports in the range 1024-10000 [2]. This makes Internet-wide scanning intrusive and inefficient, as thousands of ports must be scanned per host. Internet-wide scanning is a feasible reconnaissance method for ZeroAccess only because it runs on a single fixed port (depending on the version) [25, 36]. (2) Some botnets, like Zeus, use a different encryption key for packets destined to each bot, based on the node ID of the receiving bot. This makes it impossible to probe bots without a-priori knowledge of their node ID. Consequently, Internet-wide scanning is inherently incompatible with botnets like Gameover Zeus which use this tactic. (3) Like crawling, Internet-wide scanning cannot learn about non-routable nodes that remain hidden behind a firewall or NAT gateway.

Table 5 summarizes the susceptibility of all major P2P botnets active since 2007 to Internet-wide scanning [28]. We consider two prerequisites for Internet-wide scanning in P2P botnet reconnaissance: (1) the bot protocol must run on a known port (or small port range), and (2) it must be possible to construct a probe message to determine if a host is infected or not. As shown in Table 5, only two of the analyzed botnets run on a sufficiently small port range, namely ZeroAccess and Kelihos. Probe construction is possible for all botnets, except Zeus, which requires a-priori knowledge of the node ID in order to contact a bot. These results show that Internet-wide scanning is not a full-fledged alternative to crawling or sensor injection. Moreover, there are serious scalability issues in very large address spaces like IPv6. Still, for susceptible P2P botnets in IPv4 networks, Internet-wide scanning is a valid recon alternative if no bootstrap peer list is available for crawling.

8. DISCUSSION

This section discusses tradeoffs of the various P2P botnet reconnaissance methods, given the stealthiness and coverage of the various alternatives as studied in this paper. We summarize tradeoffs and recon characteristics in Table 6.

Method	Generic	Mapping	Attacks	Stealth strategies	Advantages	Disadvantages
Crawling	✓	Edges	I, P, S	• Protocol adherence • Address distribution • Rate limiting	• Find edges • Fast deployment	• Cannot find NATed nodes • Need out-degree limiting
Sensor injection	✓	Nodes	I, P, S	• Protocol adherence • Announcement rate limiting	• Find NATed nodes • Support sinkholing • Node verification	• Cannot find edge data • Need announcements
Internet-wide scanning	✗	Nodes	I	• Sound probe syntax • Address distribution • One-time usage	• Fast deployment • No bootstrap list	• Cannot find NATed nodes • Cannot find edge data • Not generic • Only probes

Table 6: Tradeoffs of P2P botnet reconnaissance methods. Attacks: I = Infection reporting, P = Partitioning, S = Sinkholing.

8.1 Automating Protocol Logic Extraction

Current crawlers and sensors have many protocol-specific shortcomings which make them very easy to detect. On the other hand, protocol-adherent recon tools inherently require more implementation effort. To reduce this implementation effort, part of the botnet protocol state machine can be extracted from bot samples using tools like Inspector Gadget [21]. This leaves only particular message types, such as peer list requests and responses, in need of special handling and manual implementation.

8.2 Crawling vs. Sensor Injection

To detect syntactically sound reconnaissance implementations, botmasters must rely on semantic anomalies such as in-degree or out-degree fluctuations. Our results have shown that crawlers are inherently more prone to this than sensors, as they strive to actively contact all or most of the bot population. At the same time, high in-degrees are not uncommon in legitimate super-peers, and are thus not an effective metric for detecting sensors. This means that sensors are a more naturally stealthy reconnaissance method than crawlers. Moreover, properly announced sensors can be used as a launchpad for more invasive botnet takedown efforts, for instance by serving as sinkholes [28]. Data gathered by sensors also lends itself well to mapping infected IPs for subsequent cleanup actions. This applies even more to sensors than crawling and Internet-wide scanning, as sensors can find the expected 60–87% of NATed hosts [28].

On the downside, sensors cannot gather data about the edges between bot nodes. As shown in Table 6, this data can only be gathered by crawlers. In sinkholing attacks, edge data is crucial in determining which peer list entries to poison [28]. While it is possible to augment sensors with active peer list requests to gather edge data, this effectively adds a crawling component to them. Thus, both dedicated crawlers and augmented sensors require defenses against out-degree-based detection.

8.3 Stealthy Crawling Strategies

In Section 6, we have evaluated multiple stealthy crawling strategies in the wild. As we have shown, individual crawlers which limit the contact ratio or restrict the request frequency have a strongly reduced network coverage. A more promising method is distributed crawling. Given a large network address block (at least a /16 for the botnets we studied), or several smaller blocks, it is relatively painless to implement, and has no negative impact on crawling coverage. A possible caveat is that it is still an open problem to determine whether detection techniques for distributed network scanning generalize to distributed crawling [16]. Additionally, to prevent request frequency-based detection, crawlers must limit the per-address request rate, reducing network coverage. Coverage can be improved by running multiple rate-limited crawlers in parallel (each with a different node ID, if applicable), effectively masquerading as a set of bots behind a NAT gateway.

8.4 Internet-Wide Scanning

We have also investigated Internet-wide scanning as a reconnaissance alternative in Section 7, and found it unsuitable as a generic reconnaissance strategy. Whether or not it can be used depends on the port range, bot protocol, and IP address space (i.e., IPv4 vs. IPv6) used by the target botnet. Moreover, Internet-wide scanning from a limited address range is prone to IP-based detection. The nature of Internet-wide scanning is to quickly cover large network blocks by sending standalone lightweight probes. Implementing a full bot protocol makes no sense, as this defeats the simplicity of Internet-wide scanning. This means that scanners inherently do not behave like normal bots. Therefore, Internet-wide scanning should be used only as a one-time measure, to bootstrap conventional crawling if no bootstrap peer list can be obtained. We stress that Internet-wide scanning of botnets should be used with extreme care, as by emulating the botnet protocol it may trigger IDS signatures even in uninfected networks. Unlike crawling, Internet-wide scanning cannot gather edge data, and it also cannot find non-routable bots.

8.5 Backbone-Based Sensors

An approach orthogonal to the recon methods we have studied uses sensors placed on Internet backbone systems. Given access to such systems and a suitable

signature for a particular family of bot traffic, this allows for completely passive botnet recon, which cannot be detected or blocked by botmasters. While potentially highly efficient, an obvious caveat is that this method requires the cooperation of backbone operators. Moreover, completely passive detection may be difficult for botnets like GameOver Zeus, which take active measures to thwart signature-based detection systems and encrypt traffic using destination-based keys.

9. RELATED WORK

To the best of our knowledge, our work is the first systematic study of anti-recon in P2P botnets. Prior work has hardened theoretical P2P botnets against recon, and provided anecdotal evidence of anti-recon in practice. Our work expands on these results to provide a more complete picture of anti-recon, and how to overcome it.

Early experiments with crawler-based node enumeration in P2P botnets were performed by Holz et al. in the Storm botnet [13]. At the same time, passive crawling inaccuracies due to address aliasing, firewalls, NAT gateways and churn were studied by Rajab et al. [27] and Kanich et al. [17]. Recent work has studied the completeness and accuracy of sensors compared to crawlers in P2P botnets, showing that sensors can discover up to two orders of magnitude more peers, while also verifying their authenticity [28, 15]. Additionally, technical reports on specific P2P botnets have presented evidence of anti-recon in the wild [10, 25, 35, 19, 6, 13, 34, 33, 2, 4]. Our work systematizes this anecdotal evidence, and performs the first in-depth study of the susceptibility of current and future recon tools to active disruption.

Theoretical work has studied the design of botnet protocols which inherently complicate crawling. Such designs were proposed by Hund et al. [14], who use proof-of-work schemes to prevent efficient crawling, and by Starnberger et al., who use asymetrically encrypted bot IDs to prevent node discovery [32]. Additionally, Yan et al. propose a botnet design which spreads disinformation to thwart crawlers [37]. These methods cannot be directly applied to current botnets, and require botmasters to implement complex and radically different P2P protocols. Moreover, the approach of Starnberger et al. hides bots within legitimate P2P networks which botmasters cannot control. There are currently no real-world botnets implementing these or similar designs.

Sarat et al. detect sensors with faulty protocol implementations in structured P2P networks [30], but do not investigate crawlers or protocol-agnostic anti-recon. Our work studies protocol-specific weaknesses in both crawlers and sensors, as well as protocol-agnostic crawler detection. Moreover, in contrast to prior work, we evaluate improved recon strategies in practice.

Karuppayah et al. propose to reduce the set of crawled nodes by approximating a minimum vertex cover of the botnet graph [18]. However, their results are based on simulations of Zeus which assume that all bots are simultaneously reachable, and that peer lists can be fully retrieved using only a small number of peer list requests. In contrast, our own experience is that the Zeus peer selection strategy complicates this, and that crawling coverage decreases rapidly as the set of crawled nodes is reduced (as in a minimum vertex cover).

10. CONCLUSION

We have systematically analyzed anti-recon in P2P botnets, showing that current botnets already take active measures to discourage and retaliate against crawlers and sensors, and that future recon tools are at risk of more invasive attacks. Moreover, we have shown that current recon tools suffer from a myriad of shortcomings, which significantly increase their susceptibility to subversion. Crawlers are especially prone to protocol-agnostic detection, due to their tendency for out-degree explosion. We have investigated several stealthier crawling strategies, of which distributed crawling is the most promising — it does not negatively impact crawling coverage, and is straightforward to implement given a large network address block. Alternatively, sensor injection supports node verification and improved network coverage, and can be augmented with graph connectivity information.

Acknowledgements

This work was supported by the European Research Council through project ERC-2010-StG 259108 “Rosetta”.

11. REFERENCES

[1] D. Andriesse, C. Rossow, and H. Bos. Distributed Crawler Detection in Peer-to-Peer Botnets, 2015. `http://www.few.vu.nl/~da.andriesse/papers/imc-2015-addendum.pdf`.

[2] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos. Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus. In *MALWARE'13*, 2013.

[3] Brian Krebs. Operation Tovar Targets GameOver Zeus, CryptoLocker, 2014. `http://krebsonsecurity.com/2014/06/operation-tovar-targets-gameover-zeus-botnet-cryptolocker-scourge/`.

[4] CERT.pl. Zeus P2P Monitoring and Analysis, 2013. Tech report. `http://www.cert.pl/PDF/2013-06-p2p-rap_en.pdf`.

[5] CrowdStrike. GameOver Zeus and CryptoLocker Takedown, 2014. Tech report. `http://www.crowdstrike.com/blog/gameover/index.html`.

[6] C. Davis, J. Fernandez, S. Neville, and J. McHugh. Sybil Attacks as a Mitigation Strategy Against the Storm Botnet. In *MALWARE'08*, 2008.

[7] J. Dinger and H. Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *ARES'06*, 2006.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Sec'04*, 2004.

[9] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *USENIX Sec'13*, 2013.

[10] N. Falliere. Sality: Story of a Peer-to-Peer Viral Network, 2011. Tech report, Symantec.

[11] M. Garnaeva. Kelihos/Hlux Botnet Returns with New Techniques, 2012. Tech report, SecureList. `http://securelist.com/blog/virus-watch/32021/`.

[12] Hide My Ass. List of Proxy Addresses. `https://www.hidemyass.com/proxy-list/`.

[13] 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 *LEET'08*, 2008.

[14] R. Hund, M. Hamann, and T. Holz. Towards Next-Generation Botnets. In *EC2ND'08*, 2008.

[15] B. B. H. 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 *ASIACCS'09*, 2009.

[16] M. G. Kang, J. Caballero, and D. Song. Distributed Evasive Scan Techniques and Countermeasures. In *DIMVA'07*, 2007.

[17] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *LEET'08*, 2008.

[18] S. Karuppayah, M. Fischer, C. Rossow, and M. Mühlhäuser. On Advanced Monitoring in Resilient and Unstructured P2P Botnets. In *ICC'14*, 2014.

[19] Kaspersky Lab. How Kaspersky Lab and CrowdStrike Dismantled the Hlux Botnet: Success Story, 2012. `http://newsroom.kaspersky.eu/en/texts/detail/article/how-kaspersky-lab-and-crowdstrike-dismantled-the-second-hluxkelihos-botnet-success-story/`.

[20] P. Kleissner. Me Puppet Master: Behind the Scenes of Crawling P2P Botnets, 2014. Tech report. `http://blog.kleissner.org/?p=455`.

[21] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In *S&P'10*, 2010.

[22] Microsoft Digital Crimes Unit. Microsoft, the FBI, Europol and industry partners disrupt the notorious ZeroAccess botnet, 2013. `http://www.microsoft.com/en-us/news/press/2013/dec13/12-05zeroaccessbotnetpr.aspx`.

[23] S. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *S&P'05*, 2005.

[24] A. Nappa, Z. Xu, J. Caballero, and G. Gu. CyberProbe: Towards Internet-Scale Active Detection of Malicious Servers. In *NDSS'14*, 2014.

[25] A. Neville and R. Gibb. ZeroAccess In-Depth, 2013. Tech report, Symantec.

[26] A. Panchenko, L. Pimenidis, and J. Renner. Performance Analysis of Anonymous Communication Channels Provided by Tor. In *ARES'08*, 2008.

[27] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My Botnet is Bigger Than Yours (Maybe, Better Than Yours): Why Size Estimates Remain Challenging. In *HotBots'07*, 2007.

[28] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos. P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *S&P'13*, 2013.

[29] H. Rowaihy, W. Enck, P. McDaniel, and T. la Porta. Limiting Sybil Attacks in Structured P2P Networks. In *INFOCOM'07*, 2007.

[30] S. Sarat and A. Terzis. On Tracking Peer-to-Peer Botnets. In *LEET'08*, 2008.

[31] SIDN. AbuseHUB Launched to Tackle Botnets, 2013. `https://www.sidn.nl/en/news/news/article/abusehub-van-start-botnets-aangepakt-1/`.

[32] G. Starnberger, C. Kruegel, and E. Kirda. Overbot: A Botnet Protocol Based on Kademlia. In *SecureComm'08*, 2008.

[33] B. Stock, M. Engelberth, F. C. Freiling, and T. Holz. Walowdac – Analysis of a Peer-to-Peer Botnet. In *EC2ND'09*, 2009.

[34] G. Tenebro. W32.Waledac Threat Analysis, 2009. Tech report, Symantec.

[35] T. Werner. Botnet Shutdown Success Story: How Kaspersky Lab Disabled the Hlux/Kelihos Botnet, 2011. Tech report, Kaspersky Lab. `http://www.securelist.com/en/blog/208193137/`.

[36] J. Wyke. ZeroAccess, 2012. Tech report, SophosLabs.

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