SensorBuster: On Identifying Sensor Nodes in P2P Botnets

Shankar Karuppayah
National Advanced IPv6 Centre
Universiti Sains Malaysia
Malaysia
kshankar@usm.my

Leon Böck
Telecooperation Group
Technische Universität Darmstadt
Germany
boeck@tk.tu-darmstadt.de

Tim Grube
Telecooperation Group
Technische Universität Darmstadt
Germany
grube@tk.tu-darmstadt.de

Selvakumar Manickam
National Advanced IPv6 Centre
Universiti Sains Malaysia
Malaysia
selva@usm.my

Max Mühlhäuser
Telecooperation Group
Technische Universität Darmstadt
Germany
max@tk.tu-darmstadt.de

Mathias Fischer
IT-Security and Security Management
Universität Hamburg
Germany
mathias.fischer@uni-hamburg.de

ABSTRACT

The ever-growing number of cyber attacks originating from botnets has made them one of the biggest threat to the Internet ecosystem. Especially P2P-based botnets like ZeroAccess and Sality require special attention as they have been proven to be very resilient against takedown attempts. To identify weaknesses and to prepare takedowns more carefully, it is thus a necessity to monitor them by crawling and deploying sensor nodes. This in turn provokes botmasters to come up with monitoring countermeasures to protect their assets. Most existing anti-monitoring countermeasures focus mainly on the detection of crawlers and not on the detection of sensors deployed in a botnet. In this paper, we propose two sensor detection mechanisms called *SensorRanker* and *SensorBuster*. We evaluate these mechanisms in two real world botnets, *Sality* and *ZeroAccess*. Our results indicate that *SensorRanker* and *SensorBuster* are able to detect up to 17 sensors deployed in Sality and four within ZeroAccess.

CCS CONCEPTS

•Security and privacy → Intrusion/anomaly detection and malware mitigation; *Distributed systems security;*

KEYWORDS

P2P Botnet; Anti-monitoring; Countermeasure; Sensor; Detection

ACM Reference format:
Shankar Karuppayah, Leon Böck, Tim Grube, Selvakumar Manickam, Max Mühlhäuser, and Mathias Fischer. 2017. SensorBuster: On Identifying Sensor Nodes in P2P Botnets. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 6 pages.
DOI: 10.1145/3098954.3098991

1 INTRODUCTION

Many cyber-crime activities such as Distributed Denial of Service (DDoS) and banking credential thefts are executed using *botnets* that consist of a multitude of malware-infected computers, so-called *bots*. Bots can be automatically updated and receive orders from a *botmaster*. As botnets can generate high income, they represent valuable assets to their botmasters. For the GameOver Zeus botnet, officials estimated a total revenue of about $100 million USD [4]. Early botnets followed a centralized Command and Control (C2) structure and used HTTP or Internet Relay Chat (IRC) servers for their management. However, such a centralized architecture represents a bottleneck and a Single Point of Failure (SPoF), as such, these botnets can be easily taken down [16]. Recent botnets like Sality [5], GameOver Zeus [2], or ZeroAccess (ZA) [18, 20] employ a Peer-to-Peer (P2P) architecture that significantly impedes such takedown attacks.

A takedown attack, e.g., a sinkholing of all bots [15], can only be successful when all participating bots and their interconnectivity are known. As a result, these botnets are constantly crawled [9] and infiltrated via sensors [8] by diverse actors, e.g., researchers and law enforcement agencies. *Crawling* allows to retrieve a graph of all *superpeers* in the botnet, i.e., all nodes that are directly routable, but it cannot identify nodes behind Network Address Translation (NAT) devices. *Sensor nodes*, in contrast, cannot provide an interconnectivity graph, but allow to enumerate all bots in the botnet, including those behind NAT devices. For this reason, they are important for takedown operations, as they can provide a complete list of participating bots. During a sinkholing attack, bots are tricked into believing that all other bots are no longer responsive, except designated sinkholing servers, i.e., a variant of sensor node. If the attack is successful, the botmaster cannot communicate with his bots anymore and thus loses control over the botnet.

To protect their assets, botmasters started to introduce additional defense mechanisms against monitoring activities [1]. For example, bots return a neighborlist of restricted size [10] or they employ automatic blacklisting mechanisms [2], which exploits that current crawlers can be easily identified based on their significantly increased request frequency compared to other bots in the botnet. While crawlers can be detected easily, countermeasures that focus on disclosing and/or blacklisting sensor nodes are rare [3]. As sensor nodes are fundamental to prepare takedown attacks, we anticipate that botmasters will put more attention and efforts to detect and blacklist them in the future.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
ARES '17, Reggio Calabria, Italy
© 2017 ACM. 978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3098991

As our main contribution in this paper, we present two sensor detection mechanisms for botnets from the perspective of a botmaster. We evaluate them by comparing it with our previous work called LCC [3]. For that, we use two real-world datasets gathered from our monitoring of the *Sality* [12] and *ZA* [18, 20] botnets. We do acknowledge the possibility of the proposals in this paper to be misused by future botnets to prevent monitoring. However, our intentions were to explore the possible advancements of future botnets and bring them to the view of researchers so that countermeasures could be discovered before the future botnets start implementing the proposed mechanisms.

The remainder of this paper is structured as follows: we present background and related work on P2P botnets and existing botnet monitoring mechanism in Section 2. Section 3 describes our advanced sensor detection mechanisms and Section 4 summarizes our evaluation results on the efficiency of the proposed mechanisms. Finally, Section 5 summarizes our contribution and lists future work.

2 BACKGROUND AND RELATED WORK

In this section, we first present a general overview of P2P botnets before describing two specific real-world P2P botnets, Sality and ZA. Then, we describe the commonly utilized botnet monitoring mechanisms before summarizing the state of the art in anti-monitoring mechanisms for sensor nodes.

2.1 P2P Botnets

P2P botnets utilize a *Membership Maintenance (MM)* mechanism to ensure a connected botnet overlay to withstand node churn, i.e., bots joining and leaving the botnet. Besides bootstrapping newly infected machines into the overlay, this mechanism also ensures that a bot regularly replaces offline or unresponsive bots in its *Neighbor List (NL)*.

The maintenance of the *NLs* follows the periodic and botnet-specific *MM-cycle*. Typically a cycle ranges within seconds, minutes, and up to an hour. In each cycle, a bot iterates through all entries in its *NL* and probes each neighbor for its responsiveness. Bots are removed if they are unresponsive to several successive probing attempts. If required, bots may also request the *NL* of their existing neighbors to add additional bots to their *NLs*. In the following, we describe the specific MM protocols of *Sality* and *ZA* that we obtained by our own reverse-engineering efforts of the respective malwares.

2.1.1 Sality. Each bot in *Sality* [12] maintains a *NL* with at maximum 1,000 entries and a MM-cycle of 40 minutes. In each MM-cycle, a bot probes all neighbors in its *NL* by sending them a *Hello* message. Entries in the *NL* are marked as *online* if a valid reply is received. If the number of entries in the *NL* is low at the beginning of a MM-cycle, an additional *NL Request* message (NL_{Req}) is sent to all online neighbors to discover additional bots. Bots receiving a NL_{Req} will respond with a *NL Reply* (NL_{Rep}) message containing one randomly selected (*online*) bot from their *NL*.

2.1.2 ZeroAccess (ZA). Bots in *ZA* [18, 20] maintain three *NLs*. The *primary* list contains at maximum 256 entries and the other two can contain more than 16 million entries each. The MM is carried out every 256 seconds. In each MM-cycle, a bot sequentially probes an entry from its primary list every one second with a NL_{Req} message. For each received probe message NL_{Req}, a bot replies with a NL_{Rep} message that includes its 16 most-recent entries from its primary *NL*. Bots that sent a response are shifted to the top of the primary *NL*.

2.2 Botnet Monitoring Mechanisms

Monitoring in P2P botnets is often conducted using specialized mechanisms like *crawlers* and *sensor nodes*. Both mechanisms are described in more detail in the following.

2.2.1 Crawlers. To enumerate the botnet, a crawler pretends to be a bot with very few neighbors and requests *NL* entries of other bots in the botnet. This presumes a *seednode* to start with, which is often found in the malware binary. Starting with the *seednode*, a crawler iteratively sends requests to bots and increases its knowledge about the botnet with each reply. The goal of the crawler is to obtain a near-complete *snapshot* of the botnet overlay by discovering all participating bots as well as their interconnectivities. However, crawling often fails to enumerate bots behind NAT, proxies, or firewalls, which forms the majority of the botnet population ($60 - 90\%$ according to [15]).

2.2.2 Sensor Nodes. In contrast to a crawler, a sensor node is able to enumerate non-superpeers in botnets. Exploiting the bootstrapping process, a sensor can be *announced and popularized* to existing superpeers. Thereafter, whenever a bot requests additional neighbors from one of these superpeers, it is likely that the sensor is returned. Consequently, non-superpeers will include the sensor into their *NLs* and regularly probe the sensor for its responsiveness. Based on the received probing messages, a sensor can enumerate both, superpeers and non-superpeers.

Therefore, sensors are required to be always active and responsive to ensure they remain in the *NLs* of the bots. Moreover, the high availability of a sensor also directly influences its popularity [9], i.e., amount of bots that have the sensor in their *NL*. Superpeers that are able to continuously and successfully verify a sensor's responsiveness, will continue to share the sensor's information to requesting bots, hence, improving the sensor's coverage.

Sensors are also often used as *sinkhole servers* to takedown P2P botnets. In sinkholing attacks, all entries within the *NLs* of bots are invalidated such that only sinkhole servers are reachable to the bots. Ultimately, sinkholing prevents botmasters and bots to contact each other.

2.3 Anti-Monitoring Mechanisms For Sensors

Botnet monitoring activities and sinkholing attacks are a threat for botnets. Thus, botmasters started to deploy anti-monitoring mechanisms to impede monitoring activities. However, these mechanisms focus primarily on impeding crawling activities that can easily be detected based on their contact frequency, as exploited by the blacklisting mechanism of GameOver Zeus [15].

In contrast, sensor nodes are a passive method. They only react or respond when they receive messages from bots. Their behavior cannot be easily distinguished from popular superpeers that are well-known within the overlay as well as having a high uptime [3].

Hence, compared to crawlers, they represent a stealthier monitoring approach [1].

There are also existing countermeasures to prevent sensors in current P2P botnets. Most botnets, including Sality and ZA, have IP-based filtering mechanisms. They prevent multiple sensors on a single IP address being inserted into the *NL* of a bot. GameOver Zeus filters more strictly and allows only one entry for each `/20` subnet [2]. Bots in Sality apply a local reputation mechanism that keeps track of the behavior of their neighbors. This mechanism prevents quick-deployment of sensors, by preferring existing and responsive neighbors over newly discovered ones. As for ZA, its short MM-cycle ensures that all entries in its primary NL are probed and cycled at a high rate. As a consequence, a sensor will loose its popularity if it is not highly responsive.

In [3], we proposed the first sensor detection mechanism that exploits graph-theoretic properties of a botnet to detect sensors. This mechanism can distinguish sensors from bots using the Local Clustering Coefficient (LCC) connectivity-metric. LCC exploits that most botnets have a set of stable backbone nodes [9, 17], i.e., nodes with high uptime. These nodes are well-connected among each other and are also popular among other bots in the botnet. As such, using the interconnectivity information of the bots in the botnet, i.e., the crawl data, the LCC for each node is calculated to measure how interconnected the neighbors of a node are. Based on LCC, it is possible to identify sensors that have: 1) only other sensors in their *NL*, 2) no neighbors, and 3) only non-existent neighbors, i.e, spoofed entries. As this is the only prior work available, we will compare LCC against our proposals in Section 4.

3 SENSOR DETECTION MECHANISMS

In this section, we propose two novel sensor detection mechanisms, called *SensorRanker* and *SensorBuster*. These mechanisms can disclose sensors by analyzing the interconnectivity properties of bots. The assumptions for this work are based on the related work presented in [3]. It is assumed that 1) at least one sensor is present among the bots, 2) sensors aim to be responsive all the time, and 3) sensors do not aid the botnet in any manner, i.e., including sharing valid bots as neighbors or botmaster updates/commands. Next, we introduce our formal botnet model and the proposed mechanisms.

3.1 Formal Botnet Model

A P2P botnet can be modeled as a directed graph $G := (V, E)$, where V is the set of all bots. To maintain a connected overlay, each bot follows a MM mechanism and establishes neighborhood relationships to a subset of peers. These connections $E \subseteq V \times V$, are represented as a set of directed edges (u, v) with $u, v \in V$. The neighborhood relationship, i.e., *NL*, of a peer $v \in V$ contains all successors of v

$$\text{succ}_v = \text{NL}_v = \{u | \forall u \in V : (v, u) \in E, v \neq u\}$$

that is the set of all bots that v has an outgoing connection to. The set of bots that have bot v as their neighbor can be expressed by the set of predecessors

$$\text{pred}_v = \{u | \forall u \in V : (u, v) \in E, v \neq u\}$$

The NL can also be specifically expressed as NL_v^t to reflect the exact view of peer v at time t. A botmaster can retrieve the entire neighborlist NL_v^t of a bot v at time t using a single (or multiple) request message req_v^t. Bots receiving this message respond with a reply message rep_v^t that contains all of their neighbors. Also, the *responsiveness* of a bot v at time t can be expressed as the ratio of the number of received replies to the number of sent requests between t and $t - \delta t$:

$$\text{R}_v^t = \sum_{\tau=t-\delta}^{t} \text{rep}_v^\tau \times \frac{1}{\sum_{\tau=t-\delta}^{t} \text{req}_v^\tau} \quad (1)$$

Next, we introduce our sensor detection mechanisms.

3.2 SensorRanker

Our first sensor detection mechanism uses the *PageRank* algorithm [13] to distinguish sensors from bots. The PageRank algorithm returns the importance, i.e., popularity, of websites based on the number of pages referring to them. Similar to botnets, the relation of websites and hyperlinks can be modeled as directed graphs with the websites as *nodes* and the hyperlinks as *edges*.

The PageRank algorithm assigns values in the range $[0.0, 1.0]$, where a higher value denotes higher *rank* or popularity of a node, e.g., $\text{PR}_v = 1.0$. The values are calculated based on a node's predecessors pred_v and their respective ranks. In each iteration of the algorithm, the rank of a node is distributed equally among all its outgoing edges succ_v. Therefore, the rank distributed over all edges of a node v is expressed as $\text{edge-weight}_v = \frac{\text{PR}_v}{|\text{succ}_v|}$. Thus, the PageRank value of a node is the sum of the edge-weights of all its predecessors.

The concept of PageRank is directly applicable to the popularity of bots in a P2P botnet. A bot that is known by many bots, has a large amount of predecessors and thus a higher rank. Bots become widely known and popular in the botnet when they have been available and responsive for a long period. Sensors can usually be found among the most responsive nodes in the botnet. However, they cannot be easily distinguished from popular bots and popularity alone is not an effective metric to distinguish them [1]. However, when taking PageRank into consideration, the edge-weights on outgoing edges should differ significantly between sensor nodes, which either have none or few outgoing edges compared to benign bots, i.e., not returning other bots as their neighbors. For this reason, PageRank seems as a promising metric to disclose sensors.

However, due to heavy churn in P2P botnets, using the original PageRank algorithm is not effective as it may accidentally assign unpopular bots a high edge-weight. For instance, consider a scenario where a bot has few but coincidentally high-ranked predecessors in combination with very few successors. This node will appear to have an abnormally high edge-weight. For this reason, we normalize the edge-weight of a node v by multiplying it with its *popularity ratio*, i.e., the fraction of predecessors over the total population. We refer to this adapted PageRank algorithm as *SensorRank* and it is defined as follows:

$$\text{SensorRank}_v = \text{edge-weight}_v \times \frac{|\text{pred}_v|}{|V|} \quad (2)$$

SensorRank represents the fraction of a bot's PageRank that is equally distributed among its neighbors. Although the values

for sensors would be higher than for bots, we still need an automatic method for sensor detection. For this, we utilize clustering algorithms to group nodes according to their SensorRank values. Sensors will be represented as outliers due to extreme SensorRank values compared to regular bots (cf. Section 4).

3.3 SensorBuster

SensorBuster is our second proposal that utilizes the *Strongly Connected Component (SCC)* connectivity metric [19] to detect sensors. A SCC of a directed graph G is defined as a maximum set of nodes $C \subseteq V$ with a directed path between each pair of nodes $(u, v) \in C$, i.e., $u \rightarrow v$ and $v \rightarrow u$.

To be robust in the presence of churn, P2P botnets facilitate a large number of connections between their bots to prevent segmentation. For this reason, superpeers $B \subseteq V$ often form a single and large SCCs, we refer to it as the *main* SCC. A sensor would not be part of this main SCC because it will not have any bots as its successors (cf. Section 3), thus there is no path from the sensor back into the main SCC. Even in the case of multiple colluding sensors that are returning each other, they would only establish their own SCCs that is different from the main SCC. As such, all nodes that are not included in the main SCC are most likely sensors.

4 EVALUATION

In this section, we first describe the datasets utilized to evaluate our proposed sensor detection mechanisms in comparison to the LCC-approach [3] on two real world botnets: Sality and ZA. Then, we elaborate the experimental setup of our evaluation and present our findings.

	Sanitized		Selected	
	Sality	ZA	Sality	ZA
Total Bots	7,480	3,376	2,901	526
Hourly Avg. Bots	1,227	152	839	141
Max. Neighbors	544	179	453	142
Min. Neighbors	0	0	0	0
Avg. Neighbors	308	102	297	97
Median Neighbors	356	113	343	109

Table 1: Details of the datasets

4.1 Datasets

We obtained our dataset by continuously crawling the Sality and ZA botnet at high-frequency using `Strobo-Crawler` [6] for a week with a crawling frequency of one request every second for ZA and 12 requests every second for Sality.

Sality was crawled from 21st – 27th April 2016 and ZA was crawled from 22nd – 28th February 2016. From the initial 22,506 bots discovered in the Sality botnet, we pruned 3,228 because they never responded to any *Hello* message. Similarly, out of 84,631 bots discovered in the ZA botnet, we removed 81,255, because they did not respond to any request. The large amount of nodes removed from the ZA dataset is mainly because ZA includes non-superpeers in their NL; they are not reachable by crawlers. We provide details of both sanitized datasets in Table 1.

4.2 Experimental Setup

For the analysis of our detection mechanisms for sensors, we implemented them in several *Python scripts* that use the *NetworkX* [7] and *scikit-learn* [14] modules. The scripts implement our novel mechanisms, SensorRanker and SensorBuster, as well as LCC[3]. The remainder of this subsection will provide a more detailed explanation of our setup.

4.2.1 Selecting and preparing snapshots for evaluation. The *NL*-reply mechanism that is adopted by both botnets prevents us from obtaining the entire connectivity of a bot by a single request. More details on the restriction mechanisms can be found in [5] and [12] for Sality and ZA respectively. To address this, we utilized the *NL* deduction technique introduced in [6] to infer the *near-complete* snapshot of the botnet at a given point in time, i.e., NL_v^t. Using this technique, we derived hourly snapshots of the botnets. Furthermore, based on the assumption that sensors are highly responsive, we pick only the snapshot with the least nodes for each day.

A summary of the selected snapshots for each botnet is presented in Table 1[1]. We use these snapshots as input to the detection mechanisms. From here onward, we refer to these selected snapshots as the respective botnet dataset itself.

4.2.2 Establishing ground truth. To compare the performance of the proposed mechanisms, we need to first identify sensors deployed in a botnet. Since it is almost impossible to identify from real-world botnets, we leverage the fact that the sensors do not aid the botnet in any manner (cf. Section 3) and use it to identify them. Similar to the work of Andriesse et al. [1], we use protocol anomalies to identify sensors that use incomplete/limited (re-)implementation of the botnet protocol. For that, prior to crawling, we request the latest botmaster update or command from all newly discovered bots. While regular bots are expected to return the newest update available at them upon request, a sensor would normallyrefuse to do so. Hence, we can use this observation to approximate the ground truth.

While this method itself can be used as a detection mechanism, it can be easily circumvented by replying with a corrupted message or data. This will make it difficult to distinguish between deliberately corrupted messages and those corrupted in network transit. We acknowledge that this method is not perfect for establishing ground truth, especially if a sensor does not follow our assumptions in Section 3. However, given that to the best of our knowledge, sensors have not yet been targeted by botmasters, this approach allows to approximate the ground truth for the purpose of our evaluation.

4.2.3 Sanitizing churn affected nodes. Due to churn within a botnet, artifacts might be present in the crawl data. This does not only include offline bots that are returned by bots, but also those that joined or left the botnet overlay during a snapshot. Such artifacts may skew the accuracy of any detection mechanism that relies on connectivity-specific metrics of bots. To address this, we apply the *responsiveness* ratio R_R of bots (cf. Equation 1) to handle the artifacts. Therefore, bots that have poor R_R below a certain threshold R_T will remain in the dataset but ignored during

[1]These snapshots are available at : https://github.com/botnet-research/sensor-buster

classification. Sensors would not be flagged as artifacts as they always aim to be responsive.

The responsiveness of bots in Sality is measured as ratio of the number of replies over the number of sent *Hello* messages (cf. Section 2.1.1). For ZA it is measured using the ratio of received replies over the number of sent NL_{Req} messages (cf. Section 2.1.2).

4.3 Results

In this subsection, we first present the results of our evaluation on the baseline of total sensors present in both datasets. Then, we present our parameter study analysis on the clustering algorithm suitable for SensorRanker. Finally, we provide the results from our comparative analysis of all three mechanisms. Please note that due to space constraints, results for ZA are not discussed in detail. Therefore, we summarize and present the results wherever possible.

4.3.1 Baseline information on present sensors. We utilize protocol anomalies to establish the ground truth for our evaluation, e.g., sensors not sharing botmaster updates when requested. We manually verified all flagged nodes and were able to detect 17 unique sensors in the Sality dataset and four in the ZA dataset.

	Minimum Responsiveness Threshold, $R_T \geq$									
Day	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	16	16	16	16	16	16	16	15	15	15
2	17	16	16	16	16	16	16	16	15	15
3	15	15	15	15	15	15	15	15	14	14
4	16	15	15	15	15	15	15	15	14	14
5	16	16	16	16	16	15	15	15	15	14
6	16	16	15	15	15	15	15	15	14	14
7	17	17	17	16	16	16	16	16	15	15

Table 2: Total sensors present in a particular day dependent on R_T in the Sality dataset

The detected number of sensors per day in dependence on different values of R_T for Sality is presented in Table 2. As for ZA, four sensors were detected consistently throughout the seven days regardless of the different values of R_T.

4.3.2 SensorRanker clustering algorithms. We evaluated the effectiveness of different clustering algorithms namely *K-Means*, *Gaussian Mixture Models*, *SpectralClustering*, *DBSCAN* and *Agglomerative Clustering* in classifying sensors based on the *SensorRank* values. The evaluation was conducted on both datasets with the *responsiveness* threshold set to $R_T \geq 0.0$, i.e., bots that responded to at least one message during our monitoring period. Although all algorithms perform equally well on accurately detecting sensor nodes in the Sality botnet, *DBSCAN* generates the highest number of false positives. In contrast, *Agglomerative Clustering* generates the least false positives. We repeated this evaluation on the ZA dataset and found that contrary to Sality, *DBSCAN* is able to detect all sensors with the least false positives. Based on the analysis, *Agglomerative Clustering* and *DBSCAN* is used for Sality and ZA respectively for the SensorRanker mechanism.

4.3.3 Sanitizing artifacts present in datasets. As artifacts could adversely affect the performance of the detection mechanisms, we investigated the influence of various values of R_T, i.e., between 0.0 and 0.9, on all three detection mechanisms using both datasets.

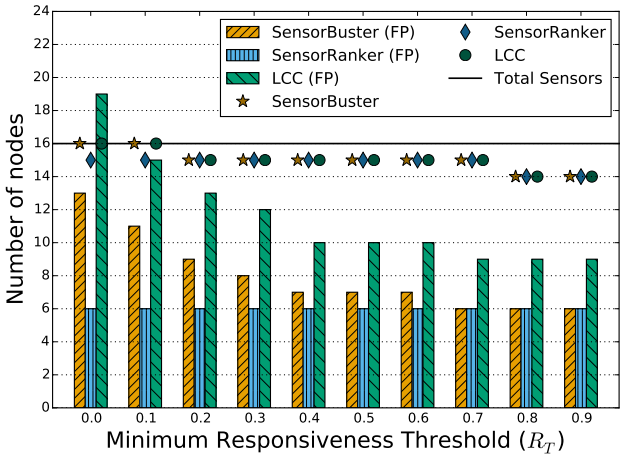

Figure 1: Influence of artifacts to the detection performance of all three mechanisms with varying values of R_T between 0.0 to 0.9 in Sality dataset on Day 6

Figure 1 depicts the number of nodes classified as true positives by the respective detection mechanisms along with the corresponding false positives in dependence on different values of R_T for Day 6 in the Sality dataset. The total number of sensors present in the dataset, i.e., 16, is indicated by a horizontal line in the plot.

The results of this analysis indicate that with increasing R_T, the number of false positives decreases for both datasets. We also found that SensorRanker is minimally affected by different values of R_T and thus is nearly unaffected by the presence of artifacts. In comparison, the influence of artifacts on LCC is more significant. Please note that with a selection of higher R_T, some highly-responsive sensors are being wrongly-flagged as artifacts and ignored by the detection mechanisms, e.g., at $R_T > 0.7$ for Day 6 in the Sality dataset. Therefore, to increase the accuracy of LCC and SensorBuster, we chose a threshold value of $R_T = 0.4$ for Sality. While this threshold value prohibits us from detecting a particular sensor on Days 2,4,6, and 7, we argue that sensors with very low uptimes, i.e., $R_R \leq 0.4$, won't be popular in the botnet and thus are not very useful to the botmaster or the defender anyway. If such a sensor becomes popular over time, it will eventually be picked up by the detection mechanisms.

Further, a threshold of $R_T = 0.2$ is observed to be appropriate for all detection mechanisms on the ZA dataset. We argue that the disparity between the threshold values for Sality and ZA is a direct result of the different MM-intervals of the botnets. A Sality bot has a larger MM-interval compared to ZA, hence having a higher probability of NL artifacts present in the form of stale or unresponsive entries.

4.3.4 Performance comparison of all mechanisms. After obtaining suitable values from the parameter study, we compared the performance of all three detection mechanisms on both datasets with $R_T = 0.4$ on Sality and $R_T = 0.2$ on ZA. Table 3 represents the detection results of all three mechanisms on the Sality dataset throughout the week. The evaluation indicate that both SensorBuster and LCC miss out at most one sensor in any given day. This particular sensor had a poor responsiveness ratio, i.e., $R_R < 0.4$. Meanwhile, even though SensorRanker has one additional false

	SensorBuster			SensorRanker			LCC		
Day	TP	FP	FN	TP	FP	FN	TP	FP	FN
1	16	9	0	15	5	1	16	10	0
2	16	10	1	15	6	2	16	10	1
3	15	8	0	14	4	1	15	11	0
4	15	10	1	14	6	2	15	16	1
5	16	9	0	15	9	1	16	13	0
6	15	7	1	15	6	1	15	10	1
7	16	14	1	16	11	1	16	18	1

Table 3: Performance comparison of all detection mechanism with $R_T = 0.4$ on the Sality dataset

negative compared to the other two, it had fewer false positives throughout the week. This particular false negative was a sensor-variant node called *BoobyTrap (BT)* from our other work [11]. A BT node is designed to have low popularity to detect crawlers that attempt to contact all possible bots. Since SensorRanker relies upon high popularity, this BT node was missed out. However, since the characteristics of a BT node is similar to a sensor (cf. Section 3), it should have been detected by the mechanism as a sensor. In comparison, LCC performs worst among all mechanisms w.r.t. the number of incurred false positives.

When evaluated on ZA, SensorBuster and LCC consistently detected four sensors, whereas SensorRanker detected only three sensors on each day. However, SensorRanker did not incur any false positives, whereas LCC and SensorBuster incurred up to five and four of them, respectively.

5 CONCLUSION

In this work, we proposed SensorRanker and SensorBuster, two novel sensor detection mechanisms that are able to detect sensors deployed in P2P botnets. We evaluated the mechanisms along with the state of the art, LCC, on real world botnets, i.e., Sality and ZA. We found that many sensors that are currently deployed in those botnets are susceptible to the proposed detection mechanisms. In particular, we were able to detect up to 17 sensors deployed in the Sality botnet and four in the ZA botnet.

Our evaluation results in Section 4 indicated that both LCC and SensorBuster mechanism are able to detect all sensors that were flagged using our baseline approach. However, LCC is found to be inferior to SensorBuster as it produces far more false positives. Meanwhile, although SensorRanker was not able to detect all sensors, it still detects most of them with the least number of false positives compared to the other mechanisms. Therefore, a botmaster would decide to deploy the SensorBuster mechanism if he is concerned on detecting all sensors that are present in the botnet and has some level of tolerance for false positives. However, if avoiding false positives is the main concern, a botmaster would most likely utilize SensorRanker instead.

From our evaluation results, we also noticed that contrary to the understanding that deployed sensors would always try to be responsive to probe messages from other bots, some sensors were found to have poor responsiveness. We speculate that this may results from network-specific issues, such as network traffic congestion, or simply from poor implementations of the sensor mechanism itself. However, as seen in the past, botmasters frequently attempt to upgrade their botnets to impede botnet monitoring. As such, it is just a matter of time before the proposed mechanisms are adopted by the botmasters on existing and newer botnets. As future work, we would like to investigate ways to evade the proposed methods to provide an upper hand back to the defenders, i.e., researchers and law enforcement agencies.

6 ACKNOWLEDGEMENTS

This work has been co-funded by Universiti Sains Malaysia (USM) through Short Term Research Grant, No: 304/PNAV/6313332, the DFG as part of project B.2 within the RTG 2050 "Privacy and Trust for Mobile Users" and EU's Horizon 2020 Research and Innovation Programme, Grant Agreement No: 700688 (TAKEDOWN).

REFERENCES

[1] Dennis Andriesse, Christian Rossow, and Herbert Bos. 2015. Reliable Recon in Adversarial Peer-to-Peer Botnets. In *ACM SIGCOMM Internet Measurement Conference (IMC)*.

[2] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. 2013. Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus. In *International Conference on Malicious and Unwanted Software: "The Americas"*.

[3] Leon Böck, Shankar Karuppayah, Tim Grube, Max Mühlhäuser, and Mathias Fischer. 2015. Hide And Seek: Detecting Sensors In P2P Botnets. In *IEEE Conference on Communications and Network Security*. 731–732.

[4] Department of Justice. 2014. U.S. Leads Multi-National Action Against "Gameover Zeus" Botnet and "Cryptolocker" Ransomware, Charges Botnet Administrator. (2014).

[5] N Falliere. 2011. *Sality: Story of a Peer-to-Peer Viral Network.* Technical Report. Symantec.

[6] Steffen Haas, Shankar Karuppayah, Selvakumar Manickam, Max Mühlhäuser, and Mathias Fischer. 2016. On the Resilience of P2P-based Botnet Graphs. In *IEEE Conference on Communications and Network Security (CNS)*.

[7] A Hagberg, P Swart, and DS Chult. 2008. Exploring network structure, dynamics, and function using NetworkX. SciPy (2008).

[8] BBH Kang, E Chan-Tin, and CP Lee. 2009. Towards complete node enumeration in a peer-to-peer botnet. *Proceedings of International Symposium on Information, Computer, and Communications Security (ASIACCS)* (2009).

[9] Shankar Karuppayah, Mathias Fischer, Christian Rossow, and Max Mühlhäuser. 2014. On Advanced Monitoring in Resilient and Unstructured P2P Botnets. In *IEEE International Conference on Communications (ICC)*.

[10] Shankar Karuppayah, Stefanie Roos, Christian Rossow, Max Mühlhäuser, and Mathias Fischer. 2015. ZeusMilker: Circumventing the P2P Zeus Neighbor List Restriction Mechanism. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*.

[11] Shankar Karuppayah, Emmanouil Vasilomanolakis, Steffen Haas, Max Mühlhäuser, and Mathias Fischer. 2016. BoobyTrap: On Autonomously Detecting and Characterizing Crawlers in P2P Botnets. In *IEEE International Conference on Communications (ICC)*.

[12] Alan Neville and Ross Gibb. 2013. ZeroAccess Indepth. *Symantec Security Response* (2013).

[13] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011).

[15] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-gross, Daniel Plohmann, Christian J Dietrich, Herbert Bos, and Dell Secureworks. 2013. P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *IEEE Symposium on Security & Privacy*.

[16] Brett Stone-gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. 2009. Your Botnet is My Botnet : Analysis of a Botnet Takeover. In *ACM CCS*. ACM.

[17] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. 2005. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. *ACM SIGCOMM Internet Measurement Conference (IMC)* (2005).

[18] Symantec. 2013. Grappling with the ZeroAccess Botnet. (2013). http://www.symantec.com/connect/blogs/grappling-zeroaccess-botnet

[19] Robert Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972).

[20] J Wyke. 2012. The ZeroAccess Botnet–Mining and Fraud for Massive Financial Gain. *Sophos Technical Paper* (2012).