COMPUTERS & SECURITY 83 (2019) 1–13

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Autonomously detecting sensors in fully distributed botnets

Leon Böck[a,*], *Emmanouil Vasilomanolakis*[b], *Jan Helge Wolf*[a], *Max Mühlhäuser*[a]

[a] *Telecooperation Lab, Technische Universität Darmstadt, Darmstadt, Germany*
[b] *Aalborg University Copenhagen, Center for Communication, Media and Information Technologies (CMI) Copenhagen, Denmark*

ARTICLE INFO

Article history:
Received 25 October 2018
Accepted 11 January 2019
Available online 19 January 2019

Keywords:
Sensor evasion
Botnet monitoring
Fully distributed botnets
P2P botnets
Computational trust

ABSTRACT

Botnet attacks have devastating effects on public and private infrastructures. The botmasters controlling these networks aim to prevent takedown attempts by using highly resilient P2P overlays to commandeer their botnets, and even harden them with countermeasures against intelligence gathering attempts. In fact, recent research indicates that advanced countermeasures can hamper the ability to gather the necessary intelligence for taking down botnets. In this article, we take the perspective of the botmaster to eventually anticipate their behavior. That said, we present a novel mechanism, namely Trust Based Botnet Monitoring Countermeasure (TrustBotMC), that combines computational trust with specially crafted bot messages to detect the presence of monitoring activity. We study and evaluate different computational trust models, to create a local and autonomous mechanism that ensures the avoidance of common botnet tracking mechanisms, such as sensors. Furthermore, we show, via our experimental results, that our approach can reduce the gathered intelligence by at least 53% compared to techniques that have been seen in botnets to date. Finally, we investigate techniques for mitigating our approach.

1. Introduction

Botnets are networks of infected computing devices, called bots. These bots are remotely controlled and instructed to conduct criminal activities by malicious entities – commonly referred to as botmasters. Botnets are used for a multitude of malicious activities such as Distributed Denial of Service (DDoS) attacks, credential theft, ransom attacks, or spam email distribution. Moreover, botnet activity appears to be on the rise; in fact, recent advances such as the (IoT), increase the botnet attack surface and capabilities (Antonakakis et al., 2017; Kolias et al., 2017).

To cope with these developments, researchers defend by proposing novel botnet detection and prevention methods; for instance, new intrusion detection algorithms and honeypots (Provos and Holz, 2007; Vasilomanolakis et al., 2015). Upon detection of botnet activity, defenders need to take a plethora of actions, e.g., bot enumeration, identification of weaknesses, or preparation of sinkholing attempts, to actually be able to take a botnet down. In many cases such actions are heavily influenced by the network architecture of the botnet.

* Corresponding author.
E-mail address: boeck@tk.tu-darmstadt.de (L. Böck).
https://doi.org/10.1016/j.cose.2019.01.004

Traditionally, many botnets have been based on a centralized architecture consisting of a Command and Control server that relays commands directly to the bots. However, this architecture presents a Single Point of Failure (SPoF) in the centralized server which can be used to seize control of, or dismantle, the botnet (Greengard, 2012). To overcome this weakness, advanced botnets implement a fully distributed C2 channel based on unstructured Peer-to-Peer (P2P) overlays. Such botnets do not inherit the SPoF of centralized approaches and they are very resilient to node churn and node removal attacks (Rossow et al., 2013). Due to their characteristics, P2P botnets are especially relevant nowadays for creating a resilient and stable basis for attackers. In particular, fully distributed botnets are prominent for their ability to provide modular post-infection capabilities (so-called *malware droppers*) to the botmasters (or to entities that hire the botnet for a specific time period). Recent examples include the VPNFilter botnet (Mansfield-Devine, 2018) and the Sality P2P botnet (Falliere, 2011) (that has been active since 2008).

As the absence of a central server prevents straightforward monitoring, researchers have developed various means for gathering intelligence on distributed botnets. This is commonly achieved by reverse engineering the communication protocol of the botnet, and afterwards deploying crawlers and sensors to enumerate the botnet population. Advanced fully distributed botnets such as GameOver Zeus (Andriesse et al., 2014) or Sality (Falliere, 2011) even implement features to impede monitoring attempts.

In this article, we take the perspective of a botmaster, with the goal to better understand and anticipate future monitoring countermeasures. Specifically, we present the Trust Based Botnet Monitoring Countermeasure (TrustBotMC), a novel approach to thwart monitoring attempts by researchers and law-enforcement agencies. The proposed mechanism is based on the utilization of computational trust along with *test-messages*, i.e., specially crafted messages, that the bots exchange to verify the correct behavior of their peers. Our work is one among others published recently that present means to detect monitoring operations in P2P botnets (Andriesse et al., 2015; Böck et al., 2015; Karuppayah et al., 2017; 2016). The multitude of different monitoring prevention mechanisms suggest that the options to harden P2P botnets are numerous and may eventually disrupt successful monitoring. Therefore, we want to highlight the need for developing new mechanisms to effectively gather intelligence on P2P botnets.

1.1. *Contributions*

This article makes the following contributions to the state of the art:

- It proposes the first technique, namely TrustBotMC, that enables bots to *locally* and *autonomously* identify and blacklist sensors in distributed botnets. Furthermore, it examines and evaluates two different techniques for the practical realization of TrustBotMC.
- It discusses and evaluates potential mitigations against such a sensor detection method.

1.2. *Assumptions*

Our work is based on the following fundamental assumptions with regard to botnets and their detection and mitigation methods.

- Botnet infiltration and monitoring techniques introduce characteristics that are distinctive compared to normal bot traffic.
- Researchers are bound to ethical, legal and technical constrains; they cannot assist the botnet into performing illegal activities and/or (unwittingly or knowingly) prevent take-down attempts.

1.3. *Outline*

The remainder of this article is structured as follows. Section 2 provides the reader with some preliminary information and background with regard to botnets, botnet monitoring and computational trust. Furthermore, in Section 3 we present our proposal, *TrustBotMC*. Afterwards, Section 4 describes the evaluation of our proposal; in particular, we discuss the simulation setup, the ability of *TrustBotMC* to blacklist advanced monitoring mechanisms and also some preliminary ideas for mitigating our own approach. Moreover, in Section 5, we discuss the state of the art with an emphasis on botnet monitoring countermeasures. Finally, Section 6 concludes this article.

2. Preliminaries

Within this section we introduce the necessary background on P2P botnets and P2P botnet monitoring. Furthermore, we discuss four trust models that will be utilized in TrustBotMC.

2.1. *P2P botnet basics*

Due to the distributed nature of P2P botnets, each bot is involved in distributing commands and other messages within the network. Therefore, the reliability of the C2 channel is based on the availability of the bots. This is usually addressed by connecting to multiple bots at the same time, to ensure that redundant connections are available. In fact, this is necessary as diurnal patterns and other user behavior causes bots to frequently disconnect from and reconnect to the botnet. This process of nodes leaving and joining is called *churn*.

To ensure that the botnet remains connected in the presence of churn, a Membership Management MM mechanism is used to frequently update connection information. Each bot in a P2P network maintains a list of other bots. This list is commonly referred to as NL and the bots stored within the NL are called neighbors.

Each bot regularly contacts its neighbors to check their responsiveness as well as to receive updated commands. If all neighbors are unavailable, a bot is isolated from the botnet and will not be able to receive any updates or commands. Hence, it is important to update the NL frequently by replacing inactive neighbors with others (active bots). This is accomplished by sending probing messages to all bots in the NL in

reoccurring intervals, the so-called MM cycles. These probing messages are commonly referred to as *hello* messages.

If a node remains unresponsive for a prolonged period of time, it will be replaced by a "fresh" entry of another online bot. To do so, bots send *NL-request* messages to their neighbors asking for additional bots. Upon such a request a bot replies with a *NL-reply* containing a subset of the entries of its own NL. Furthermore, botnets also use the MM cycle to exchange information about the *ID* of the latest instruction set. If one bot does not have the most current update, it will query a neighbor to forward the latest instruction set. In the case of the Sality botnet[1], the *ID* is directly embedded in the *hello* and *hello-reply* messages.

In this article we make use of a formal graph model to describe the connectivity of P2P botnets. This model is based on similar models previously published in (Karuppayah et al., 2017; Rossow et al., 2013). We model a P2P botnet as a graph $G = (E, V)$, where V represents the set of super-peers (i.e., the actively "routable" bots) in the botnet and the connectivity between the bots is represented by the set of directed edges $E \in V \times V$. Each edge $e(u, v) \in E$ with $u, v \in V$, $u \neq v$ represents a directed connection from bot u to v. The NL of a bot u is defined as a set of edges $e(u, v) \in NL_u$. The *out-degree* of a bot u is defined as the number of outgoing edges of u : $deg^+(u) = |NL_u|$ Furthermore, we define the *popularity* (or *in-degree*) of a bot (or sensor) u as the number of incoming edges, i.e., the number of bots that contain u in their NL: $deg^-(u) = |(v, u) \in E|$.

2.2. P2P botnet monitoring

The lack of a SPoF makes it difficult to take down P2P botnets. In order to take any actions against P2P botnets, it is required to obtain reconnaissance information. This process of gathering intelligence is commonly referred to as *monitoring*. Monitoring is usually achieved by reverse engineering the (botnet) malware and re-implementing its communication protocol in various so-called *monitoring mechanisms*. Generally two types of monitoring mechanisms are used: *crawlers* and *sensors* (Rossow et al., 2013).

2.2.1. Crawlers

Crawlers provide an aggressive approach to obtain intelligence about a botnet. Starting with a set of seed-nodes[2], the crawler consecutively sends *NL-requests* to all known bots. The entries contained in the *NL-replies* sent by the bots are added to the queue of the crawler. With this approach, one can quickly collect information about actively "routable" bots (*super-peers*), and their inter-connectivity. However, a major drawback of crawlers is the inability to contact bots that are behind Network Address Translation (NAT) devices or firewalls. Therefore, crawlers often underestimate the overall population of a botnet by up to two orders of magnitude (Rossow et al., 2013).

Another drawback of crawlers is that they behave very differently from normal bots and introduce a lot of noise to the botnet. Specifically, crawlers exhibit an unusually large *out-degree*. This makes them prone to detection and mitigation by monitoring countermeasures (Andriesse et al., 2015; Karuppayah et al., 2016).

2.2.2. Sensors

Contrary to crawlers, sensors follow a less aggressive approach to monitoring. They are implemented such that they imitate the behavior of a regular bot. They join a botnet based on the specified bootstrap mechanism and wait for incoming connections from other bots. To eventually infiltrate the NL of other bots, a sensor has to reply to the *hello messages* of other bots. Over time, the likelihood of other bots replacing some of their inactive neighbors with the sensor increases. If a connection is initiated by a bot, NAT devices will route the traffic accordingly and allow sensors to enumerate bots behind NAT. However, this comes at the cost of lacking connectivity information that can be obtained by crawlers. Such information may be crucial to successfully conduct attacks against the botnet. Therefore, sensors and crawlers are commonly used in combination to obtain both connectivity information and more accurate enumerations.

As sensors are a more passive monitoring mechanism they are harder to detect than crawlers. While their aim is to obtain a very high *in-degree*, which could be used as a distinguishing factor, this behavior is also observed for regular bots (Andriesse et al., 2015). Nevertheless, different approaches exist to detect sensor nodes; these are discussed in greater detail in Section 5.

2.3. Computational trust

Computational trust offers the functionality that supports an entity to make informed decisions within digital environments. This is important, as such decisions are often accompanied by risk and uncertainty. On a theoretical level, trust is commonly described as a contextual relationship between a *trustor* and *trustee*. The trustor engages in an action that requires trusting the target entity referred to as trustee.

Mathematically, trust is a measure for the inherent quality or trustworthiness of the trustee. This measure is estimated based on evidence of previous behavior of the trustee. For the use-case of TrustBotMC, evidence consists of binary experiences of positive or negative interactions with the trustee.

Different approaches exist to calculate a trust score based on evidence. Within our work, we focus on four prominent trust models, capable of working with binary evidence. Namely, these are the *eBay user rating* (Jøsang et al., 2007), *Beta distribution* (Commerce et al., 2002), *CertainTrust* (Ries, 2009b), and *Subjective logic* (Jøsang, 2001). These are introduced in the following alongside a reputation mechanism called *goodcount* which is used by the Sality botnet. We want to point out, that the Beta distribution, CertainTrust and Subjective Logic are isomorphic to each other, even though they can outwardly provide different functionalities through their parameters.

eBay User Rating eBay[3] provides a commercial reputation system that aggregates positive, neutral and negative experiences denoted as: $(e_+, e_0, e_-) \in E$. The ratings from different

[1] The Sality P2P botnet will be the reference botnet example throughout this article.

[2] Seed-nodes are commonly found during the reverse engineering of a malware, or by scanning the Internet.

[3] https://www.ebay.com.

trustors are combined to provide an overall rating for a trustee that is presented to all users of the platform. Specifically, users are provided with the absolute ratings and a trust score based on the positive and negative experiences:

$$T_{ebay}(E) = \frac{|e_+|}{|e_+| + |e_-|} \tag{1}$$

Furthermore, the eBay trust model provides absolute values for all ratings in the last month, last six months and last year to provide insights on the development of the trustee.

Beta distribution The Beta distribution family can be used as a trust model, with a trust score based on the expectation value of the distribution (Commerce et al., 2002). In particular, the Beta distribution $f(p|\alpha, \beta)$ can be used to present posteriori probabilities for binary events. In the context of computational trust, the expectation value of the Beta distribution:

$$E(p) = E^{Beta}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta} \tag{2}$$

is used to predict the future behavior of a trustee. For that, the parameters are set to $\alpha = r + 1$ and $\beta = s + 1$ where r and s represent the number of positive and negative experiences, respectively. The trust score is equal to the expectation value of the beta distribution and is calculated as:

$$T_{Beta} = \frac{r + 1}{s + r + 2} \tag{3}$$

Moreover, the beta reputation system can be extended with two additional parameters s_0 and r_0 to present a *base rate* (Jøsang et al., 2003) or *prior knowledge* (Ries, 2009a). Within this article, we will use this extended model with $\alpha = r + r_0$ and $\beta = s + s_0$.

Subjective logic Subjective logic (SL) (Jøsang, 2001) is a trust model based on the Dempster–Shafer belief theory (Shafer, 1976). SL uses an opinion quad-tuple $\omega_x^A = (b, d, u, a)$ to describe the belief b an entity A has in the truth of a statement x. At its core, SL uses a beta distribution with $\alpha = r + 2a$ and $\beta = s + 2 - 2a$. The trust score is calculated as the expectation value:

$$T_{SL} = E(\omega_x^A) = b + ua \tag{4}$$

Here, the expectation value of the regular beta distribution $b = \frac{r}{r+s+2}$ is extended by considering the uncertainty $u = \frac{2}{r+s+2}$ and the *atomicity* a, which applies a weight to the uncertainty of the trust score.

CertainTrust CertainTrust (CT) (Ries, 2009b) is a trust model expressing trust in uncertain environments with a trust score based on the expectation value:

$$T_{CT} = E^{Beta}_{f,w,N}(r, s) = E^{Beta}(r, s, r_0, s_0) \tag{5}$$

A distinguishing factor from other trust models is that the initial trust s_0 and r_0 can explicitly be changed through the parameter f. Moreover, CT provides *linear fade out* for the initial trust. This is accomplished by dynamically recalculating s_0 and r_0 upon collection of new experience. The speed at which prior knowledge is replaced can be set as parameter w. Furthermore, N denotes the number of experiences required at which prior knowledge is no longer considered, as enough real experiences have been collected. In addition to the context dependent parameters f, w and N, CT also provides an *aging factor* $a \in [0, 1[$. This allows to prioritize newer experiences by adding a negative weight to the influence of older experiences.

Goodcount. *goodcount* is a reputation mechanism implemented by the Sality botnet (Falliere, 2011). While strictly speaking it is not a trust model, it is analogous and also the only implementation of such a countermeasure by any known botnet to date. A bot's *goodcount* is represented as an integer value that represents the availability or reliability of a bot to answer to *hello-messages*. Each bot maintains a *goodcount* value for each of its neighbors. Upon receipt of a *hello-reply* the *goodcount* is increased by one, whereas it is decreased by one if no answer is received in response to a *hello-message*.

This *goodcount* value is used during a bot's MM cycle to decide if a bot is to be removed from the NL or not. This mechanism is supposed to prevent preemptive deletion of highly responsive and reliable neighbors. Moreover, it is in place to prevent sensors from easily invading and replacing legitimate entries in a bot's NL.

3. TrustBotMC: a trust based monitoring countermeasure

Within this section we propose the Trust Based Botnet Monitoring Countermeasure (TrustBotMC), a novel monitoring countermeasure based on computational trust. *TrustBotMC* allows bots in fully distributed botnets to locally compute the trustworthiness of their neighbors and take automated countermeasures in the form of blacklisting against sensor nodes infiltrating their NL. In the following, we introduce the ideas and assumptions behind TrustBotMC. Afterwards, we discuss in detail the three core components of TrustBotMC: the *test message* concept, the *computational trust* models and the *blacklisting* mechanism.

Our proposed trust management approach builds on the fundamental assumption that sensors and crawlers can be distinguished from regular bots. We argue that this is the case for two reasons: (i) the goal and therefore the behavior of monitoring mechanisms is different from bots (cf. Section 2.2), (ii) researchers and law enforcement officials are bound by legal, ethical and technical constraints. Some of these constrains are that sensors should not disseminate commands from the botmaster, or aid the overlay maintenance by sending valid *NL-reply* messages (Karuppayah et al., 2017). If a sensor violates these constraints, it not only raises questions regarding the legal and ethical consequences of actively participating in the maintenance of the botnet, but it could also prevent sinkholing attempts against the botnet. The reason for this is that sensors are re-implementations of the botnet protocol but they are unlikely to share the same vulnerabilities as the malware itself. Therefore, exploiting a bug in the malware to sinkhole the botnet will not affect the sensors. Hence, if these sensors actively share commands or NL-entries with the

sinkholed bots, they will allow sinkholed bots to re-connect to other bots and effectively break the sinkholing attempt.

To leverage the distinctive behavior of monitoring mechanisms, we adapt the concept of test messages (Fung et al., 2009). At a glance, *TrustBotMC* regularly sends test messages to bots and collects experiences based on the validity of the reply. The collected experiences are fed into a trust model to compute a continuously updated trust score for each neighbor in a bot's NL. If the trust value for any of the neighbors falls below a predefined threshold t_{min}, this neighbor will be permanently removed from the NL and added to the blacklist. All messages originating from bots on the blacklist will be ignored indefinitely.

With this approach bots can locally detect and remove sensor nodes from their NL. This will decrease the popularity of sensors and therefore their ability to enumerate the botnet. This greatly limits the capabilities of researchers and law enforcement officials to obtain the necessary information required to take down the botnet.

To summarize, the proposed mechanism can be described as a three-tuple $TBMC = (M, T, t_{min})$, with M being a non-empty set of test messages, a trust model T, and a minimum trust threshold t_{min}. Each of these three components will be described in greater detail in the following subsections.

3.1. *Test messages*

To make decisions about the trustworthiness of a neighbor, bots need to obtain historical data. To obtain this data, TrustBotMC uses the concept of test messages (Fung et al., 2009). Bots frequently send special messages to their neighbors for which the (correct) answer is clearly defined/known. If a neighbor does not reply to the message as expected, a negative experience is recorded. Similarly upon receipt of a correct answer a positive experience is recorded.

While the specific implementations of test messages can differ, they have to fulfill three conditions: (i) they are constructed in a way such that the correct answer is predefined/known, (ii) a sensor should not (ethical/legal/technical reasons), or cannot (e.g., due to the encrypted nature of messages) answer such that the response is considered a positive experience, (iii) regular bots do not, or only rarely (e.g., due to churn) answer in a way that is considered a negative experience.

We propose two different types of test messages: *Bogus Neighborlist Request (BNLM)* messages and *BCM* messages. Each of them assumes the aforementioned legal, ethical and technical limitations of monitoring mechanisms. The reason for proposing these two test messages is that the type of messages are common for all known P2P botnets. This has the benefit that a sensor cannot distinguish them from their regular counterparts. We want to highlight, that a test-message does not need a non test message counterpart. Botmasters can introduce entirely novel test messages to their botnet if they fulfill the three requirements stated earlier. As an example, such a message could be the command to conduct a short-timed (DDoS) attack that can be observed by other bots. Complying with such a test-message would require active participation in attacks. Therefore, due to the aforementioned legal and ethical reasons, sensors would be distinguishable as they cannot carry out the attack.

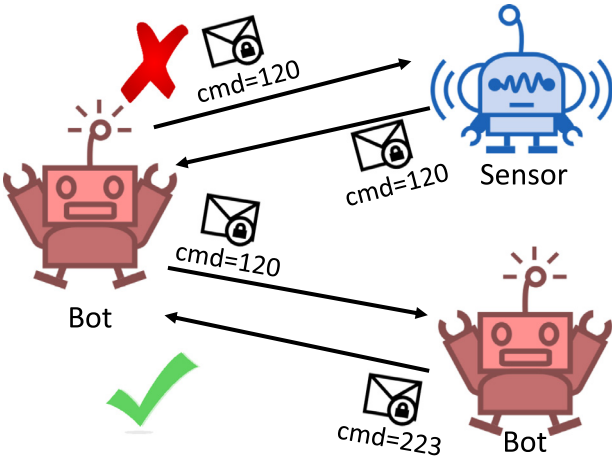


Fig. 1 – TrustBotMC example: two cases of success and failure using the BCM technique.

3.1.1. *Bogus command sequence message (BCM)*

A BCM is a specifically crafted *hello-message* that contains a very old command sequence number, i.e., significantly lower than the current command sequence number. Upon receipt of such a message a bot will reply with its current (higher) command sequence number and the corresponding command set. However, contrary to the behavior of a bot, a sensor will not behave in the same way and instead send a *hello – reply* message that matches the command sequence number of the BCM. Sensors must behave this way, as they would otherwise have to attach an updated command set in their reply. As such command sets are commonly encrypted, the sensor does not know its contents. Therefore, it might contain content that is either legally or ethically problematic, or it is a command used by botmasters to break out of a sinkholing.

Consequently, a bot will consider the receipt of *hello – reply* messages with a (recent) higher command sequence number as a positive experience, whereas a reply with the same or outdated command sequence number is considered a negative experience. This behavior is also depicted in Fig. 1. Note that depending on the specifics of the botnet, a botmaster has to define what is considered a recent command sequence. Otherwise, sensors could cheat by sending higher, yet outdated, command sequences.

We want to point out, that BCMs are unlikely to incur a great amount of false positive (FP) classifications. Even if a bot replies with a highly out-dated command sequence number, it will subsequently be updated and reply correctly to future test-messages.

3.1.2. *Bogus neighborlist request message (BNLM)*

BNLMs are test messages that target sensors by forcing them to share legitimate bot entries. However, sharing of legitimate entries helps the botnet maintain its overlay. Therefore, due to legal and ethical limitations sensors cannot share bot entries. Instead they can send empty *NL-replies* or duplicate addresses of other sensors (Andriesse et al., 2015). Moreover, sharing

legitimate entries could prevent so called sinkholing attacks against the botnet (Karuppayah et al., 2017).

To cover different behavioral patterns of sensors, a negative experience will be recorded for all following scenarios: (*i*) empty *NL-reply*, (*ii*) only unresponsive or non-existing bots, (*iii*) only blacklisted bots, and (*iv*) duplicate entries.

It is important to mention that many of these replies can also occur in *NL-replies* of legitimate bots. Therefore, contrary to the previously discussed BCM, BNLMs are likely to incur FP classifications. Specifically for botnets with low NL-reply sizes (e.g., Sality) it is likely that an unresponsive or blacklisted bot is returned. Therefore, we expect BNLMs to be more likely to incur FPs than BCMs.

Theoretically sensors could avoid the *test message* detection strategy by not replying (to the test messages). This behavior would reflect packet losses or a bot going offline during the message exchange. Therefore, we consider not receiving a reply in a timely manner as a negative experience. While this introduces the possibility of additional FPs, we consider this insignificant with regard to the overall number of test messages sent. Moreover, the approach is designed such that test messages are only sent to bots that are considered online, i.e., they replied to a regular hello message beforehand. This avoids collecting negative experiences for legitimate bots that are currently offline.

3.2. *Leveraging computational trust*

The test message approach allows a bot to collect positive and negative experiences for each of its neighbors. To decide whether to trust or distrust a neighbor, we need to calculate the trustworthiness based on the collected evidence. For this, we leverage the trust models introduced in Section 2.3. By feeding the experiences collected for each neighbor to the trust models, we receive an individual trust score for all NL entries.

3.2.1. *Trust threshold*

While the calculated trust scores provide us with an indicator of the trustworthiness of a bot, we need a method for interpreting the results. As our goal is to make a binary decision about the trustworthiness of a bot, we propose the utilization of a trust threshold.

In this context, the minimum trust threshold t_{min} represents a lower boundary for trust scores. Upon each new experience, negative or positive, the trust score is updated according to the respective trust model. If the newly calculated score falls below the threshold of t_{min}, the bot associated with that trust score is considered untrustworthy. Once a neighbor is considered to be untrustworthy, it will be blacklisted and removed from a bot's NL. Effectively, the trust threshold can be used to set the strictness of TrustBotMC. A high value will lead to quicker blacklistings, i.e., few negative experiences will quickly lead to a trust score lower than t_{min}. Contrary, a low threshold will allow a greater number of negative experiences to be collected before a blacklisting occurs. This also allows legitimate bots to restore their trust score with positive experiences. Therefore, it depends on the *test-messages* used and the preferences of the botmaster to choose a proper trust threshold. For our evaluations we used a parameter study to pick a trust threshold that minimizes FPs while maintaining high TPs.

3.2.2. *Trust models*

In the context of *TrustBotMC*, an adapted version of the *eBay* user rating system is represented as $T = T_{eBay}(n)$. Here, n denotes the number of experiences that will be used by the trust model. More specifically, to compute the trust score, the n most recently collected experiences will be taken into account. Moreover, instead of a combined rating for each bot, every bot maintains their own ratings for each bot on their NL. This not only allows local computation of trust scores, but also prevents abuse of the rating system by researchers, which could falsely down-rate legitimate bots.

TrustBotMC uses an extended *beta distribution*, which is represented as $T = T_{Beta}(r_0, s_0)$. Here, r_0 and s_0 denote the initial trust.

Similarly to the beta distribution, *Subjective Logic* does not consider aging of evidence but takes into account an initial assumption of trust via its base rate parameter $a \in [0, 1]$. In *TrustBotMC*, Subjective Logic is represented as $T = T_{SL}(a)$.

CertainTrust is the only trust model introduced that considers both aging and other environmental factors. In *TrustBotMC*, CertainTrust is represented as $T = T_{CT}(f, w, a)$. Here, the aging factor $a \in [0, 1[$ determines the expected number of evidence $N \in R^+$ via the formula $N = 1/(1 - a)$.

3.3. *Blacklisting*

To prevent a blacklisted bot (i.e., sensor) from causing more harm to the botnet, several steps need to be taken to avoid further communication or manipulation by the potential sensor node. To achieve this, TrustBotMC follows a three-step approach of: *i*) removing the suspected sensor from the NL, *ii*) adding it on the blacklist, and *iii*) blocking any future communication.

Removing the suspected sensor from the NL will prevent it from being shared through *NL-request* messages or contacted during a bot's MM-cycle. Therefore, the sensor will not be able to observe the activities of the bot or increase its view of the botnet. Furthermore, adding the sensor to the blacklist will prevent re-adding the bot to the NL and allow a bot to filter incoming messages from previously blocked sensors. Lastly, by ignoring all incoming messages from nodes on a bot's blacklist, sensors cannot obtain any further intelligence about the availability or connectivity of a bot.

4. Evaluation

Within this section, we evaluate TrustBotMC and compare the effectiveness of the trust mechanisms used in it. Specifically, we compare the trust mechanisms against the *goodcount* approach (the reputation-based approach used by the Sality botnet). Furthermore, we evaluate how much information can be gathered by sensors in the presence of TrustBotMC. Lastly, we investigate how collaborative monitoring can mitigate the loss of monitoring intelligence introduced by TrustBotMC.

4.1. Setup

For our evaluation, we use the OMNeT++ (Varga and Hornig, 2008) discrete simulation framework. Within OMNeT++, we utilize a framework (as presented in our previous work Böck et al., 2018) that implements the Sality botnet and extends it, such that it either uses its original reputation mechanism (i.e., *goodcount*) or the test message based approach of TrustBotMC (along with any of the trust management mechanisms discussed in Section 2.3). The decision to use a simulation framework over a real world testbed, allows to evaluate the proposed mechanism on significantly larger networks.

To ensure a realistic churn behavior of the bots, we use the churn model presented by Karuppayah (2016) and Böck et al. (2018). Based on the churn model, we simulated a botnet with a population of 5500 and an average active population of 1422. We picked this number as it fits the latest reports on the size of the super-peer population of the Sality botnet (Haas et al., 2016). While botnets can be of significantly larger size, we argue that the size of the botnet has no immediate effect on our mechanism. In TrustBotMC each bot locally computes the trust score and maintains their own blacklists. Therefore, the size of the botnet has no immediate effects on the effectiveness of TrustBotMC.

In addition to the bot population, we introduced up to 100 sensor nodes to the botnet. As it requires time to develop sensor nodes, it is unrealistic to assume that they are present from the zero hour of the botnet. Therefore, we use a warm up period before the sensors join the network.

As the churn model requires approximately 41 days to stabilize itself (Böck et al., 2018), we decided upon a warm up period of 50 days before the sensors join the botnet. Once the sensors join the network, we observe their popularity for a period of 14 days. Overall, this results in a simulation time of 64 days. During that period, we recorded the state of the botnet overlay with a granularity of six hours. To evaluate the effectiveness of each proposed test message, we conducted three separate experiments: (1) only BCMs, (2) only BNLMs, and (3) a combination of BNLMs and BCMs with an equal ratio of both message types. To account for statistical variances we repeated each of our experiments 16 times.

For the evaluation of the different trust models, we had to pick a common threshold $t_{min} \in [0, 1]$ and set the parameters for each trust model. For this, we conducted a parameter study to identify which parameter combinations perform best. As the result of this study, we decided upon a minimum threshold of $t_{min} = 0.4$ and the trust parameters specified in the following subsection. Based on the identified threshold, we later compare the trust model based on the sensor popularity, (TP), (FP) and (UD). UD denotes sensors that have not been discovered yet by a bot and therefore are not classified yet. Therefore, the sum of sensor popularity and TP does not always match the total botnet population as the sensor has not yet been discovered and classified by all bots.

4.2. Results

This section presents the evaluation results for BCM and BNLMs for each of the trust models in comparison against the Sality *goodcount* mechanism. Afterwards, a potential mitigation based on increased sensor deployment is evaluated and discussed.

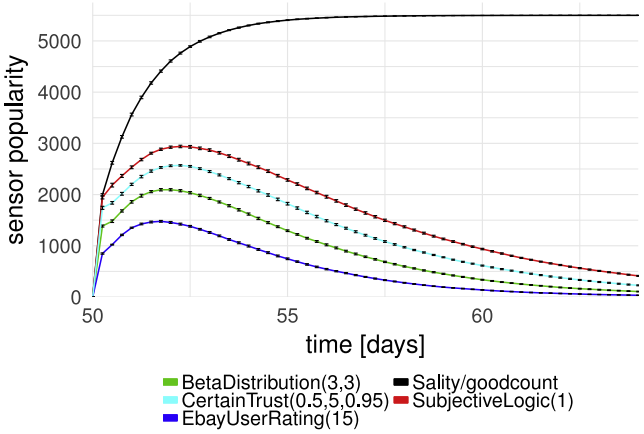


Fig. 2 – Development of the popularity of a single sensor over time with Sality's *goodcount* mechanism and different trust models compared ($t_{min} = 0.4$), $M = \{BCSM\}$.

4.2.1. Bogus commands sequence messages

To recapitulate, BCMs target the legal, ethical and technical issues of sensor sharing botmaster commands with other bots. Fig. 2 presents the popularity of a single sensor throughout the observation period of 14 days for each trust model and the Sality *goodcount*.

The plot highlights that TrustBotMC significantly impedes the intelligence gathering of the sensor. For Sality, a single sensor, in our simulation, is capable of enumerating the entire botnet after ten days of deployment in the Sality botnet. This is not possible in TrustBotMC, where at most 53.436% of the botnet could be enumerated by a single sensor at the peak of its popularity. Moreover, this peak occurs after about two days and then only decreases due to blacklisting. At the end of the simulation, i.e., after 14 days of monitoring with the sensor, only 7.49% of the bots have the sensors in their NL. This is not the case for the unmodified Sality, where the popularity of the sensor steadily increases.

Fig. 2 also highlights the differences between the evaluated trust models. The results show that TrustBotMC indeed mitigates the intelligence gathering by the deployed sensor. Out of the observed trust models, the eBay user rating performed the best with a peak popularity of the sensor of only 1477. In comparison, the sensor peaked at a popularity of 2939 if SubjectiveLogic is used as the trust model. Nevertheless, the popularity of the sensor drops below 500 ($< 10\%$) for all trust models after the observation period of 14 days. This poses a significant problem (for researchers and law enforcement), as node churn and dynamic IP addresses require monitoring knowledge to be as recent as possible. However, due to TrustBotMC, sensors quickly lose the ability to collect the latest monitoring intelligence.

Furthermore, we investigated the FPs incurred by each trust model, i.e., falsely blacklisted regular bots. The results of this are presented in Table 1 alongside with detailed results on the popularity of the sensors after 14 days of monitoring.

Table 1 – Performance overview of different trust models against a single sensor at its peak popularity and after the monitoring period of 14 days ($t_{min} = 0.4$). The TN are omitted from the table due to space reasons and can be calculated as $5500^2 - UD - FP$.

Trust model	Peak popularity	End of simulation			
		Popularity (FN)	TP	FP	UD
$T_{eBay}(15)$	**26.855%** (1477)	**0.6%** (33)	5466	0	1
$T_{Beta}(3, 3)$	**38.073%** (2094)	**2.909%** (106)	5394	0	0
$T_{CT}(.5, 5, .95)$	**46.727%** (2570)	**4.127%** (227)	5272	0	1
$T_{SL}(1)$	**53.436%** (2939)	**7.490%** (412)	5088	0	0

Table 2 – Performance overview of different trust models against a single sensor at its peak popularity and after the monitoring period of 14 days ($t_{min} = 0.4$), M = {BNLM}. The TN are omitted from the table due to space reasons and can be calculated as $5500^2 - UD - FP$.

Trust model	Peak popularity	End of simulation			
		Popularity (FN)	TP	FP	UD
$T_{eBay}(15)$	**27.127%** (1492)	**0.745%** (41)	5458	11,556,538	1
$T_{Beta}(3, 3)$	**38.255%** (2104)	**1.873%** (103)	5391	37,250	6
$T_{CT}(.5, 5, .95)$	**46.764%** (2572)	**4.327%** (238)	5262	748	0
$T_{SL}(1)$	**53.582%** (2947)	**7.636%** (420)	5080	13	0

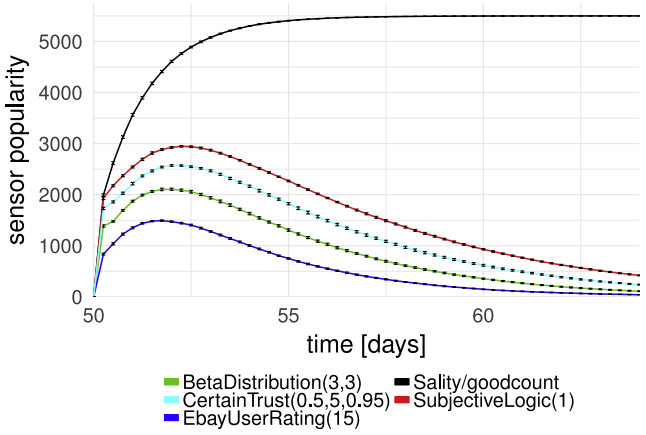


Fig. 3 – Development of the popularity of a single sensor over time with Sality's *goodcount* mechanism and different trust models compared ($t_{min} = 0.4$), $M = \{BNLM\}$.

It is interesting to note that none of the trust models incurred any FPs. This indeed suggests that TrustBotMC is an effective and robust countermeasure against monitoring attempts.

This is likely the case because TrustBotMC, by design, avoids collecting FP experiences for normal bots and BCMs being robust towards false negative experiences (c.f. Section 3.1.1). Hence, eBay performs best as it makes its decisions faster than the other models, which are designed to avoid decisions that can lead to FPs.

4.2.2. *Bogus neighborlist messages*

Next we want to repeat the experiment using BNLMs. BNLMs leverage that sensors are restrained from sharing legitimate NL entries with other bots. Contrary to BCMs, bots may answer incorrectly to BNLMs. Therefore, FP classifications are more likely to occur when using BNLM (c.f. Section 3.1.2). Fig. 3 depicts the popularity of a single sensor throughout the observation period of 14 days for each trust model and the standard Sality *goodcount*.

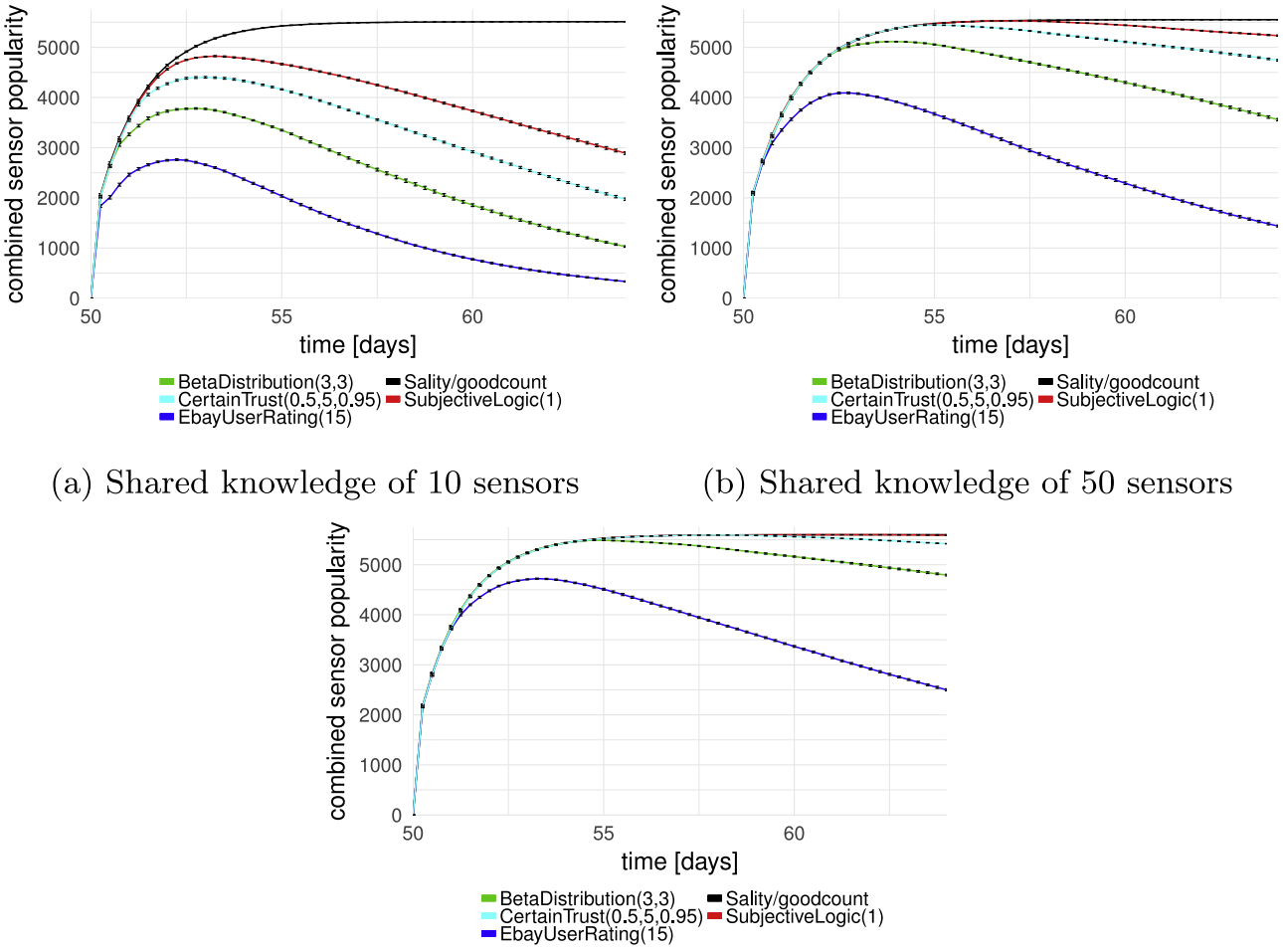
In comparison to the experiment with BCMs, the two types of test messages show only slight deviations with respect to the ability of preventing sensors from infiltrating the botnet. However, as Table 2 highlights, all four trust models incur FPs which is not the case for BCMs.

Even though the eBay user rating still performs the best, the results clearly indicate that the eBay user rating incurs a significantly larger amount of FP blacklistings compared to the other three trust models. In fact, 38% of all possible edges within the network are removed based on blacklisting. This impacts the resilience of the graph structure of the botnet and is undesirable from a botmaster's perspective. Contrary, while SL performed worst at preventing sensors from infiltrating a bot's NL it incurs the least FPs (13) throughout our observation period. CT similarly only incurs 748 FPs, whereas Beta distributions incur 37,250 FPs. Considering, that the overall simulation time of 64 days is fairly short in comparison to real world botnets with lifespans of several years, the results indicate that eBay user ratings and the Beta distribution based trust model are not suited for deployment in conjunction with test messages that are likely to incur negative experiences for legitimate bots. In such a case, the lesser performing CT or SL trust models should be used by botmasters, to impede sensor infiltration while mitigating negative effects on the botnets own structure.

4.2.3. *Discussion*

The experiments of running TrustBotMC with BCM and BNLM yielded differing results. The eBay user rating model performed best at impeding monitoring efforts with sensor nodes in combination with both *test messages*. However, in combination with BNLM, it incurred a significant amount of FPs which renders it unusable in such a scenario. Interestingly, no FP were accumulated at all in combination with BCM. The reason for this discrepancy lies in the nature of the *test messages*. While a bot may provide incorrect answers to a BNLM under some circumstances, this is not the case for BCMs. Therefore, by the characteristics of BCMs a FP may only happen if a bot goes offline while being tested.

In summary, the choice of a proper trust model greatly depends on the characteristics of the *test messages* and the possibility of bots replying incorrectly. In a setting, where the *test message* is likely to incur FPs a more robust trust model such as CT or SL should be used. Contrary in a scenario with few possibilities of FP classifications the more aggressive eBay user rating model yields better results.



(a) Shared knowledge of 10 sensors

(b) Shared knowledge of 50 sensors

(c) Shared knowledge of 100 sensors

Fig. 4 – Comparison of the popularity of 10, 50 and 100 sensors (collaborating), with Sality's *goodcount* mechanism and different trust models (t_{min} = 0.4).

We also repeated our experiments with a combination of both BCM and BNLM. The results of this are presented in Appendix A. Lastly, we want to highlight once more that the *test messages* used for these experiments are only two possible options. We chose them for their applicability to a wide range of P2P botnets. However, botmasters theoretically have complete freedom of adding arbitrary *test messages* to aid in identifying sensor nodes.

4.3. *Mitigation*

As a mechanism such as TrustBotMC significantly impedes monitoring operations, we need to identify means to continue monitoring operations. A simple but resource consuming approach is to increase the number of sensors. In the following, we investigate how running the same simulations (with 10, 50 and 100 colluding sensor nodes) influences the intelligence gathered by means of combining the knowledge from all sensors.

Fig. 4a–c depict the combined popularity of 10, 50 and 100 sensors respectively. As expected, the combined intelligence gathered by the sensors increases with an increasing number of sensors. Our results even indicate that 50 sensors are sufficient to fully enumerate TrustBotMC for a short period, if SubjectiveLogic is used as the trust model. However, with the eBay trust model at an average of 4088.5 bots (74.44%) were enumerated at the peak using 50 sensors. Throughout the monitoring period of 14 days this dropped down to 1438.25 bots (26.15%). Furthermore, even increasing the number of sensors to 100 is insufficient to fully enumerate the botnet if eBay user rating is used as the trust model.

To identify why increasing the number of sensors yielded only small gains in the overall monitoring knowledge, we also investigated the popularity of the individual sensors. We found out, that injecting too many sensors at the same time leads to a situation where the sensors compete against each other. The reasons for this are limitations in NL-size, Sality's neighbor addition strategy, and the order and time of sensors joining the botnet. We discuss these findings in more detail in Appendix B. In summary, this effect is likely to occur in other botnets as well. Therefore, increasing the number of sensors might be more efficient for monitoring other botnets.

Overall the results indicate that increasing the number of sensors can help to mitigate the effect that TrustBotMC has on monitoring operations. However, the limited success and significant resource overhead are clear indicators that we need more advanced monitoring strategies to overcome advanced monitoring countermeasures such as TrustBotMC. One possible approach is to develop smart approaches for collaborative monitoring, as the straightforward method of increasing the number shows odd effects such as sensors competing for popularity.

5. Related work

In this section we discuss the state of the art by emphasizing on (botnet) monitoring countermeasures cas well as on techniques for detecting sensors and crawlers.

5.1. *Botnet monitoring countermeasures*

Countermeasures against monitoring are a common feature of P2P botnets. Among the most typical countermeasures is rate limiting the size of *NL-reply* messages (Andriesse et al., 2014; Falliere, 2011; Neville and Gibb, 2013). This is supposed to prevent crawlers from easily obtaining full information about a bot's neighbors. In addition, the GameOver Zeus botnet also implements an automated blacklisting mechanism that triggers if more than five *NL-request* messages are sent from the same IP address within a sliding window of one minute (Andriesse et al., 2014). As discussed in Section 2.3, the Sality botnet (Falliere, 2011) implements the so-called *goodcount* mechanism to prevent sensor nodes from easily replacing legitimate bots that have been reliable over a prolonged period of time. The *goodcount* mechanism prevents replacing long lived and reliable NL-entries with newer potentially unreliable neighbors. This also affects sensors, as it becomes more difficult to infiltrate and remain in a bot's NL (c.f. Appendix B).

5.2. *Sensor and crawler detection*

In (Böck et al., 2015; Karuppayah et al., 2017) the authors present how sensor nodes can be detected within P2P botnets based on three different graph metrics. In more details, they make use of the local clustering coefficient, PageRank (Page et al., 1999) and strongly connected components to identify sensor nodes in the overlay graph. Their approach is based on the assumption, that sensors will not aid the botnet by returning legitimate bot entries upon a *NL-request*. While these mechanisms effectively identify sensor nodes, they require an aggregated view or global knowledge on the graph connectivity. On the contrary, our approach only requires a bot's local knowledge to autonomously blacklist sensor nodes.

In Andriesse et al. (2015), Andriesse et al. investigate how crawlers and sensors can be detected based on protocol and behavioral anomalies. Specifically for crawlers their approach has proven to be successful. The anomalies used by the authors to identify the crawlers cover both protocol and behavioral anomalies. Furthermore, they show how sensors can be detected based on protocol violation. Nevertheless, they state that their approach is not generally applicable to the detection of sensor nodes, as violations could be avoided in sensor implementations. Karuppayah et al. also use protocol violations to autonomously detect crawlers based on a bot's local view (Karuppayah et al., 2016). Their approach is based on setting traps targeting the behavior of crawlers to locally identify and subsequently blacklist crawlers.

6. Conclusion

Fully distributed botnets exhibit a unique level of resilience against take down attempts and monitoring. Recent work indicates that essential intelligence gathering mechanisms can be detected and repelled by botmasters. In this article, we contribute to this field by presenting TrustBotMC, a method for locally and autonomously identifying sensors in P2P botnets via the utilization of computational trust and special messages that are exchanged between bots. We want to point out, that TrustBotMC assumes that defenders are bound to legal and ethical restrictions. If these restrictions are circumvented (e.g., for reasons of national security) TrustBotMC will not be effective. Nevertheless, we argue that such a scenario is not generally applicable to all botnets. Our evaluation results suggest that the use of TrustBotMC can significantly reduce the benefits of sensor monitoring. In particular, we show that TrustBotMC is much more efficient in detecting and blacklisting sensor nodes than existing mitigation mechanisms in botnets; that is, Sality's *goodcount* mechanism. Moreover, we examine collaborative methods for mitigating the TrustBotMC threat. While collaboration seems to be a promising solution to continue successful monitoring operations, the straightforward approach of simply sharing data shows odd effects of sensors competing against each other. Nevertheless, increasing the number of sensors, while requiring a lot of resources, does increase the overall monitoring intelligence. Moreover, increasing the number of sensors allows to slow down the loss of monitoring information due to blacklistings by TrustBotMC. To further mitigate the effects of monitoring countermeasures such as TrustBotMC, we suggest future work to focus on alternative monitoring approaches as suggested in Böck et al. (2018) as well as to more advanced collaboration strategies that avoid competition among sensors.

Acknowledgements

This work was supported by the Royal Bank of Canada within the project Novel P2P Botnet Detection under Grant Agreement 2761478.4.

Appendix A. BNLM +BCM

We repeated our experiments for TrustBotMC with equally distributed numbers of BCMs and BNLMs. Fig. B.5 indicates, that the results for the popularity of the sensor are similar to using only one of the two test-message types. However, the number of FP incurred by the trust models shows a more interesting pattern. While eBay incurred 35.46% less FPs than only using BNLMs, the Beta distribution incurred 77.22% less

Table A1 – Performance overview of different trust models against a single sensor at its peak popularity and after the monitoring period of 14 days ($t_{min} = 0.4, M = \{BCLM, BCM\}$).

Trust Model	Peak	End of simualtion			
	popularity	Popularity	TP	FP	UD
$T_{eBay}(15)$	**27.309%** (1502)	**0.764%** (42)	5456	7,458,526	2
$T_{Beta}(3, 3)$	**38.036%** (2092)	**1.945%** (107)	5393	8484	0
$T_{CT}(0.5, 5, 0.95)$	**46.309%** (2547)	**4.091%** (225)	5275	88	0
$T_{SL}(1)$	**53.527%** (2944)	**7.436%** (409)	5091	1	0

FPs. This indicates, that the Beta distribution may be usable in combination with test-messages that generate a lower number of false negative experiences than BNLMs. This is interesting, as using the Beta distribution model is more effective at blacklisting sensors than SL or CT. Therefore, botmasters need a deep understanding of the deployed test-messages to chose the trust model that provides high TP classifications with minimal FPs.

Appendix B. Competing sensors

In Section 4.3 we investigated how deploying additional sensors improved the intelligence gathered during monitoring operations. We observed, that the additional information gained decreases with rising numbers of sensors. To investigate why this occurs, we analyzed the popularity of individual sensors. Fig. B.6 presents the average popularity of a sensor in a group of 50 colluding sensors. In comparison to Fig. 2, we observe that the average popularity of the 50 sensors is significantly lower than the popularity of one single sensor.

We investigate whether the aforesaid observation is related to the way sensors are deployed during the simulation. That is, sensors are deployed within a quick succession. This leads to a scenario where a large number of sensors attempt to infiltrate NLs at the same time, effectively competing against each other. Hence, we examined how injecting sensors at a

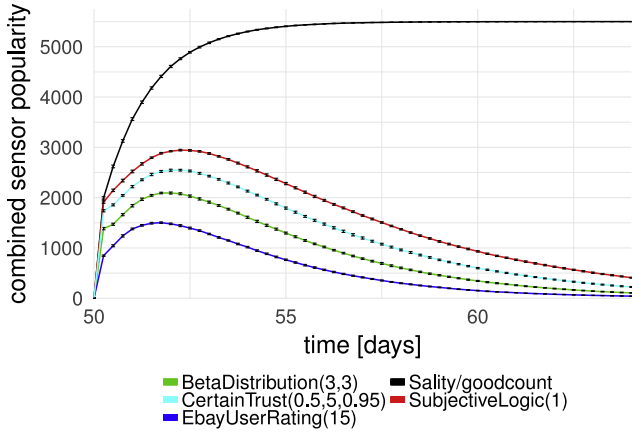


Fig. B1 – Development of the popularity of a single sensor over time with Sality's *goodcount* mechanism and different trust models compared ($t_{min} = 0.4, M = \{BCLM, BCM\}$).

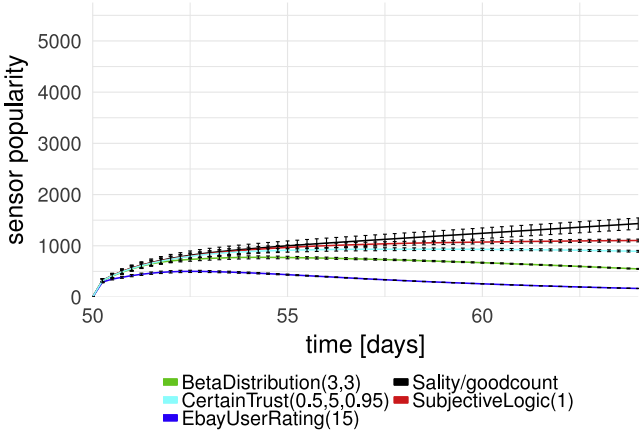


Fig. B2 – Comparison of the average popularity of 50 (non collaborating) sensors over time with Sality's *goodcount* mechanism and different trust models ($t_{min} = 0.4$).

slower pace influences their popularity. However, even 40 min intervals between injecting the sensors did not solve the problem. While even longer intervals might reduce the competition among sensors, it will also spread out the monitoring intelligence across a longer period of time. Therefore, the peak popularity of all sensors will remain high for a longer period of time, but the overall peak will be lower. In fact, deploying 100 sensors with 40 min intervals, causes the last sensor to join only once the first sensors are starting to be blacklisted and dropping in popularity. Therefore, we attempt to optimize the peak popularity of the sensors by injecting them in quick succession.

The specific effects of this competition among sensors depends on the design of the MM-protocol. In the following, we discuss the specific effects of the Sality MM-protocol and discuss why similar effects are expected for other MM-designs.

Sality MM. Sensors can infiltrate a bot's NL by sending a server announcement message. The receiving bot will then either add the sensor to an empty slot in its NL or replace the newest entry. Due to the design of the Sality MM at most 20 empty slots exist in a Sality NL before a bot actively asks for additional entries to populate its NL. Therefore, if sensors join in quick succession, they must compete for at most 20 empty slots within a bot's NL. Once the free slots are occupied, a sensor will most likely replace another sensor in the bot's NL. This leads to a situation, where the first 19 sensors may obtain an empty slot in a bot's NL and all following sensors actively replace other sensors. Fig. B.7 depicts the popularity of individual sensors out of 100 in the order they joined the botnet. The first 20 have a higher popularity as they had a better chance of occupying empty slots. Consequently, a large group of sensors have a similar and comparatively low popularity. This changes again for the last sensors injected to the botnet. As there are less or no sensors joining the botnet after them, they are less likely to be replaced and maintain a higher popularity.

This indicates, that injecting large amounts (i.e., ≥ 20) of sensors into the Sality botnet at the same time only yields marginal improvements in monitoring knowledge. To overcome this drawback, we suggest two possible approaches:

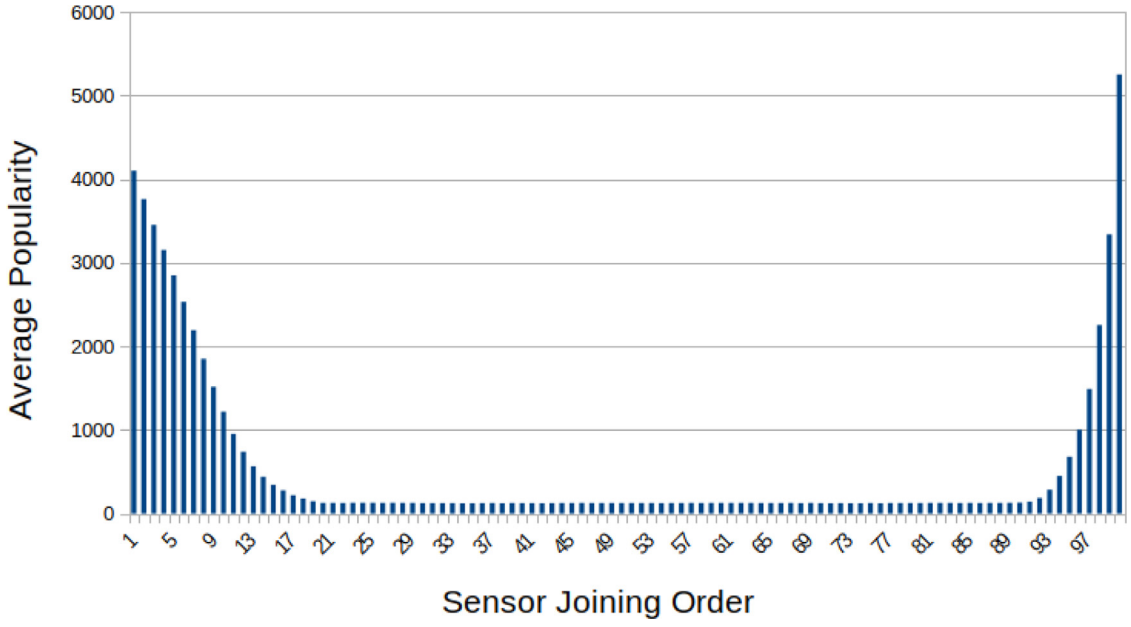


Fig. B3 – Popularity after four days for individual (non collaborating) sensors in the order of joining the botnet.

(i) inject sensors with greater inter-arrival times or (ii) develop more balanced approaches to inject large groups of sensors.

Other-MM-protocols While the discussed effect of competing sensors is specific for the Sality botnet, it is likely to occur in other botnets as well. If a MM-protocol allows large sets of new bots to enter a NL this will make the botnet susceptible to NL-poisoning attacks (Rossow et al., 2013), i.e., a defender could replace all existing entries with sensor to sinkhole all botnet traffic. Therefore, most botnets will prevent a large number of bots or sensors to join a NL in quick succession.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cose.2019.01.004.

REFERENCES

Andriesse D, Rossow C, Bos H. Reliable recon in adversarial peer-to-peer botnets. In: Proceedings of the 2015 Internet Measurement Conference, IMC '15. New York, NY, USA: ACM; 2015. p. 129–40. doi:10.1145/2815675.2815682.

Andriesse D, Rossow C, Stone-Gross B, Plohmann D, Bos H. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. 2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE) 2014;00:116–23. doi:10.1109/MALWARE.2013.6703693.

Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, Durumeric Z, Halderman JA, Invernizzi L, Kallitsis M, et al. Understanding the mirai botnet. In: USENIX Security Symposium; 2017. p. 1092–110.

Böck L, Vasilomanolakis E, Mühlhäuser M, Karuppayah S. Next generation p2p botnets: Monitoring under adverse conditions. In: Research in Attacks, Intrusions, and Defenses (RAID). Springer International Publishing; 2018. p. 511–31. doi:10.1007/978-3-030-00470-5_24.

Böck L, Karuppayah S, Grube T, Mühlhäuser M, Fischer M. Hide and seek: Detecting sensors in p2p botnets. In: 2015 IEEE Conference on Communications and Network Security (CNS); 2015. p. 731–2. doi:10.1109/CNS.2015.7346908.

Commerce BE, Jøsang A, Ismail R. The beta reputation system. In: Proceedings of the 15th Bled Electronic Commerce Conference. Citeseer; 2002. p. 1–14.

Falliere N. In: Technical report. Sality: Story of a peer-to-peer viral network. Symantec Corporation; 2011.

Fung CJ, Zhang J, Aib I, Boutaba R. Robust and scalable trust management for collaborative intrusion detection. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management; 2009. p. 33–40. doi:10.1109/INM.2009.5188784.

Greengard S. The war against botnets. Communications of the ACM 2012;55(2):16. doi:10.1145/2076450.2076456.

Haas S, Karuppayah S, Manickam S, Mühlhäuser M, Fischer M. On the resilience of p2p-based botnet graphs. In: 2016 IEEE Conference on Communications and Network Security (CNS); 2016. p. 225–33. doi:10.1109/CNS.2016.7860489.

Jøsang A. A logic for uncertain probabilities. International Journal of Uncertainty Fuzziness and Knowledge-Based Systems 2001;9(03):279–311. doi:10.1142/S0218488501000831Cited.

Jøsang A, Hird S, Faccer E. Simulating the effect of reputation systems on e-markets. In: Nixon P, Terzis S, editors. In: Trust Management. Berlin, Heidelberg: Springer Berlin Heidelberg; 2003. p. 179–94. doi:10.1007/3-540-44875-6_13.

Jøsang A, Ismail R, Boyd C. A survey of trust and reputation systems for online service provision. Decision Support Systems 2007;43(2):618–44. Emerging Issues in Collaborative Commerce doi: 10.1016/j.dss.2005.05.019.

Karuppayah S. Advanced monitoring in P2P botnets. Technische Universität Darmstadt; 2016. Ph.D. thesis.

Karuppayah S, Böck L, Grube T, Manickam S, Mühlhäuser M, Fischer M. Sensorbuster: On identifying sensor nodes in p2p botnets. In: Proceedings of the 12th International Conference on Availability, Reliability and Security, ARES '17. New York, NY, USA: ACM; 2017. p. 34:1–34:6. doi:10.1145/3098954.3098991.

Karuppayah S, Vasilomanolakis E, Haas S, Muhlhauser M, Fischer M. BoobyTrap: On autonomously detecting and characterizing crawlers in P2P botnets. IEEE international conference on communications, ICC, 2016.
Kolias C, Kambourakis G, Stavrou A, Voas J. Ddos in the iot: Mirai and other botnets. Computer 2017;50(7):80–4. doi:10.1109/MC.2017.201.
Mansfield-Devine S. Nation-state hacking–a threat to everyone. Comput Fraud Secur 2018;2018(8):17–20. doi:10.1016/S1361-3723(18)30077-0.
Neville A, Gibb R. In: Technical report. ZeroAccess Indepth. Symantec Security; 2013.
Page L, Brin S, Motwani R, Winograd T. In: Technical report, 1999-66. The PageRank citation ranking: bringing order to the web. Stanford InfoLab; 1999.
Provos N, Holz T. Virtual honeypots: from botnet tracking to intrusion detection. Addison-Wesley Professional; 2007.
Ries S. Extending bayesian trust models regarding context-dependence and user friendly representation. In: Proceedings of the 2009 ACM symposium on applied computing, SAC '09. New York, NY, USA: ACM; 2009a. p. 1294–301. doi:10.1145/1529282.1529573.
Ries S. Trust in ubiquitous computing. Technische Universität; 2009b. Ph.D. thesis.
Rossow C, Andriesse D, Werner T, Stone-Gross B, Plohmann D, Dietrich CJ, Bos H. Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets. In: 2013 IEEE symposium on security and privacy; 2013. p. 97–111. doi:10.1109/SP.2013.17.
Shafer G, 42. Princeton University Press; 1976.
Varga A, Hornig R. An overview of the omnet++ simulation environment. In: Proceedings of the 1st international conference on simulation tools and techniques for communications, networks and systems & workshops, Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering); 2008. p. 60:1–60:10.
Vasilomanolakis E, Karuppayah S, Mühlhäuser M, Fischer M. Taxonomy and survey of collaborative intrusion detection. ACM Comput Surv 2015;47(4):55:1–55:33. doi: 10.1145/2716260.

Leon Böck is a Ph.D. student at the Telecooperation Labs at Technical University of Darmstadt. His research focus is on detection, monitoring and prevention of Peer-to-Peer botnets. In addition to the technical aspects of his research, he is also interested in the legal and privacy concerning issues related to fighting botnets. Leon received his M. Sc. in computer science from TU Darmstadt in 2017 with a masters thesis on the topic "On P2P botnet monitoring in adverse conditions".

Dr. Emmanouil Vasilomanolakis is a senior researcher and area head at Technische Universitt Darmstadt. His research interests include collaborative intrusion detection, honeypots, botnet monitoring and alert data correlation. He received a Ph.D. from the Technische Universitt Darmstadt in 2016 for his dissertation "On Collaborative Intrusion Detection". Heretofore, he received his diploma (Dipl.-Inform.) and M.Sc. from the University of the Aegean (Greece) in 2008 and 2011 respectively. His master thesis, in the area of honeypots, was conducted in cooperation with the National Center of Scientific Research "Demokritos". Lastly, he worked as a researcher for AGT International, on the field of IoT security, from 2014–2015.

Jan Helge Wolf is an information security consultant focusing on penetration testing and overall technical security. His work and research interests center around cybercrime, darknet markets, and the intersection of (offensive) information security and business in general. Jan graduated from University of Trento, Italy, and Technical University of Darmstadt, Germany with an M.Sc. in IT Security after obtaining his B.Sc. from University of Mnster, Germany.

Prof. Dr. Max Mühlhäuser is head of the Telecooperation Lab at Technische Universitt Darmstadt. His Lab conducts research on smart ubiquitous computing environments in three research fields: middleware and large network infrastructures, novel multimodal interaction concepts, and human protection in ubiquitous computing (privacy, trust, and network security). Max regularly publishes in Ubiquitous and Distributed Computing, HCI, Multimedia, E-Learning, and Privacy&Trust conferences and journals and authored or co-authored more than 400 publications. He is active in numerous conference program committees, as organizer of several annual conferences, and as member of editorial boards or guest editor for journals like Pervasive Computing, ACM Multimedia, Pervasive and Mobile Computing, Web Engineering, and Distance Learning Technology.