Metasoma: Decentralized and Collaborative Early-Stage Detection of IoT Botnets

Lodovico Giaretta, Ahmed Lekssays, Barbara Carminati, *Senior Member, IEEE,* Elena Ferrari, *Fellow, IEEE,* and Šarūnas Girdzijauskas

***Abstract*—Early-stage detection of botnets during their spreading phase, before any attack, is fundamental to IoT security. Recently introduced lightweight memory networks represent the state of the art in this domain. However, they require a central system to capture and analyze all traffic in the network, which may not always be feasible in real-world scenarios.**

In this paper, we introduce a decentralized and collaborative alternative, in which the IoT devices themselves are responsible for this task without any central observer or coordinator. Our results show that the performance of this novel approach is competitive with similar centralized solutions, despite the lack of a global view of the network at any participating device.

We also provide an extensive analysis of the security limitations of our fully-decentralized detection system. We identify the potential exploits that an attacker may attempt to perform, assess their impact on the IoT network as well as propose and evaluate effective countermeasures.

***Index Terms*—Security and Privacy, Botnet Detection, Industrial IoT (IIoT), Device-to-Device Communication, Deep Learning.**

I. Introduction

IoT devices have been growing exponentially in the last few years thanks to their usefulness and the growth of Industry 4.0. As a result of this growth, the number of IoT connections is expected to reach 83 billion by 2024[1].

This growth brought a spotlight on IoT security. These devices are an interesting target for malicious actors due to their large number and weak security measures. Most IoT devices are constantly connected to the Internet yet use weak default configurations, including weak passwords and unencrypted communications.[2] Moreover, their low computational resources limit the use of heavy security solutions that are employed on more powerful computers.

Due to these issues, attackers have been successfully infecting unsecured IoT devices and using them to perform Distributed Denial of Service (DDoS) on a scale not possible before. A notable example is the DDoS attack on DNS provider Dyn, one of the largest attacks known, which reached a bandwidth of 1.2 Tbps [1]. This attack was performed using the Mirai botnet, which infects IoT devices (turning them into "bots") and spreads across local and Internet connections to form a network of compromised devices, i.e. the botnet.

Given this threat, a lot of research has been put into detecting botnet-initiated attacks at the network level to filter the malicious traffic and identify the infected IoT networks. However, this kind of solution can only *mitigate* ongoing attacks. Very few works have instead focused on *preventing* these attacks by identifying and isolating the botnets during their spreading phase when they are still recruiting new bots and before they initiate any attack.

Most of the approaches focusing on this early-stage detection challenge are based on shallow ML models [2]. More recently, some deep learning approaches have been suggested [3], [4], the most recent and effective being LiMNet [4]. LiMNet introduces long-term device memories to preserve information about previous packets exchanged by each device, not only to better classify malicious packet flows but also to perform device-level classification tasks. However, this approach requires a central system to capture and sequentially analyze each individual packet exchanged in the network, as shown in Figure 1a. Unfortunately, this may be impossible in highly-decentralized networks, where communications happen in a peer-to-peer fashion or where no single entity is responsible for the networking infrastructure. Furthermore, even when technically possible, centralized monitoring may be infeasible due to the sheer volume of traffic generated by IoT devices.

To tackle these limitations, in this paper, we propose Metasoma[3], a supervised deep learning system for early-stage botnet detection that employs decentralized, collaborative inference. Metasoma extends LiMNet and adapts the underlying techniques to a decentralized environment. Instead of a central monitor and memories database, each device runs a local instance of Metasoma, which monitors the local traffic and constructs partial memories based on the behavior of each peer in its visible subset of the network, as shown in Figure 1b.

Metasoma is unique in its novel approach to decentralized *collaborative* inference: these partial, local memories are shared between devices according to a peer-to-peer communication protocol, and then merged with the local ones by the receivers, thus boosting their knowledge and improving

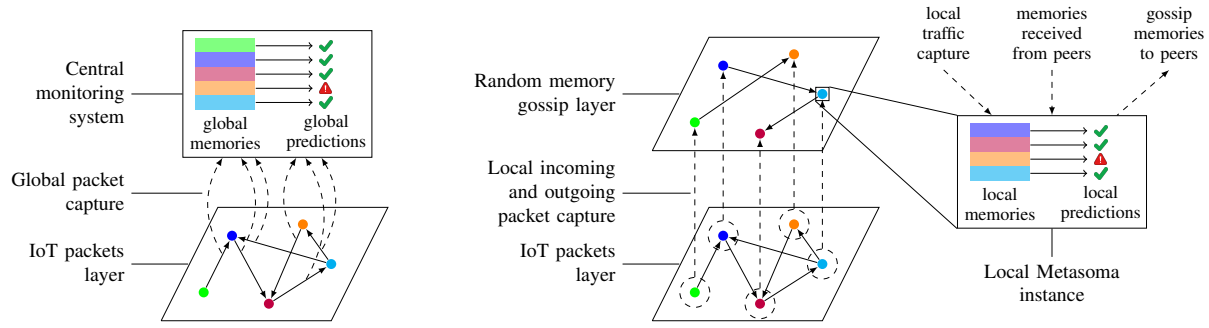
This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813162: RAIS – Real-time Analytics for the Internet of Sports. The content of this paper reflects the views only of their author(s). The European Commission/Research Executive Agency are not responsible for any use that may be made of the information it contains.

Lodovico Giaretta and Šarūnas Girdzijauskas were with KTH Royal Institute of Technology, Stockholm, Sweden. Ahmed Lekssays, Barbara Carminati and Elena Ferrari were with Università degli Studi dell'Insubria, Varese, Italy.
Corresponding authors: Lodovico Giaretta (lodovico@kth.se) and Ahmed Lekssays (ahmed@lekssays.com).

[1]https://www.juniperresearch.com/press/iot-connections-to-reach-83-bn-by-2024

[2]https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot

[3]*Metasoma* is the upper body part of ants, used for several cooperative tasks like transporting food and defending from outsiders.



(a) Centralized (LiMNet) (b) Decentralized (Metasoma)

Fig. 1. Comparison of centralized and decentralized botnet detection systems. LiMNet employs a central monitoring system that captures all traffic and builds complete device memories. In Metasoma each device builds partial local memories that are gossiped with other devices to achieve collaborative detection.

the detection accuracy. These memories are produced, merged and then classified by machine learning components that are jointly trained on a single objective. Thus, to the best of our knowledge, Metasoma is the first deep learning model that achieves decentralized inference by sharing internal model representations across a peer-to-peer network.

Contributions. The contributions of this paper can be summarized as follows:

- We develop Metasoma, which to the best of our knowledge is the first supervised early-stage botnet detector based on a fully-decentralized deep learning inference model.
- We perform an extensive security analysis of Metasoma, identifying its potential weaknesses and implementing suitable mitigations.
- We experimentally evaluate the performance of Metasoma and the efficacy of its security measures, with the results demonstrating the potential of these novel techniques to effectively protect IoT networks.

Outline. The rest of the paper is organized as follows. Section II introduces key concepts used by Metasoma. Section III presents the building blocks of the system, while Section IV analyses the security of Metasoma, discussing potential attacks and mitigations. Section V presents extensive experimental analysis of the system, including security mitigations. Section VI presents related work. Finally, Section VII provides some concluding remarks.

II. Background

Before delving into the architecture of Metasoma, we review certain concepts from the fields of deep learning and distributed systems that play crucial roles in its design: memory networks, normalizing flows, and gossip protocols.

A. Memory Networks

Memory networks, first presented in [5], have been introduced to the field of botnet detection in [4] and play a key role in the architecture of LiMNet and, by extension, Metasoma.

Memory networks are built around a central long-term memory storage, which encodes all facts known to the machine learning model and makes them available for downstream inference. Different deep learning components can act on this storage, adding and modifying memories to encode new facts or querying it for relevant information to complete a task.

In their original formulation, memory networks are composed of four components: *input feature map*, *generalization layer*, *output feature map*, and *response layer*. The *input feature map* converts new incoming information to an internal representation that is passed to the *generalization layer*, which updates the internal memory of the model. The resulting input representation and the updated memory are combined by the *output feature map* into an output representation. The latter is used by the *response layer* to generate the final model output.

B. Normalizing Flows

From a high-level perspective, a deep learning model can be seen as a function $f_\theta : x \mapsto z$, parameterized by a set of trainable parameters θ, which maps an input distribution X to an output distribution Z. Normalizing flows [6] are a family of deep models that enjoy additional properties, the most important being invertibility, that is, the ability to construct an inverse transformation $f'_\theta : z \mapsto x$. This, together with other properties, ensures that normalizing flows can be efficiently used to map back and forth between the probabilities densities of the input and output distributions (X and Z).

Normalizing flows are often used to map a complex input distribution X to a simpler, typically normal (hence the name) distribution Z. Once trained, this mapping can be used to estimate the probability density of X in a certain point, or in other words the likelihood of a certain input x to belong to the distribution X. The inverse mapping can be used in generative contexts, allowing efficient sampling of values of X with desired likelihoods, by first sampling from the well-known distribution Z and then applying f_θ^{-1}.

In this work, normalizing flows will be used in Section IV to estimate the probability distribution of device memories, thus limiting the ability of malicious actors to tamper with them.

C. Gossip Protocols

Gossip protocols [7] are a family of network communication techniques based on the phenomenon of gossiping in social circles. They are extremely efficient in disseminating and aggregating information over a network in a decentralized fashion and have been used for many different tasks [8], [9].

In broad terms, in any gossip protocol, each device periodically shares its information with a randomly-selected peer. Upon receiving information from a peer, each node combines it with its own local information, thus increasing its own knowledge, which it will then share at the next occasion.

To the best of our knowledge, Metasoma is the first work that employs lightweight gossip protocols to enable decentralized and collaborative inference. However, they have been extensively explored in the context of decentralized training [9]–[11], where they have been shown to be robust to adverse conditions, such as suboptimal network topologies and imbalanced data distributions [12].

III. Architecture

A. Target Scenarios

IoT devices are employed in a wide variety of applications, ranging from wearable devices to smart homes to the "Industry 4.0" [13]. Metasoma targets scenarios such as smart grids, smart cities, smart factories and smart farms. These Industrial IoT (IIoT) settings come with a specific set of challenges and opportunities. On one hand, the large amount of devices deployed, and the volume and velocity of their communications makes centralized real-time analysis infeasible. On the other hand, these deployments typically happen in relatively controlled environments, with well-defined workloads and administered by either a single entity or a small consortium. These characteristics make it possible to create inexpensive overlay networks and deploy models that were trained ahead of time on domain-specific network traces.

Metasoma exploits these properties, as it performs decentralized inference by sharing, in a peer-to-peer fashion, memories that are produced by a deep learning classification model trained ahead of time, in a centralized environment, using network traces obtained from the same environment where it will be deployed.

B. System

Metasoma aims to achieve botnet detection in a decentralized, collaborative fashion. Each IoT device in the network thus needs to participate in the protocol, by analyzing the portion of the network traffic that it is aware of in search of clues, and then sharing and merging its information with other peers to obtain a more complete picture of the network.

Thus, Metasoma contains two detection pathways: a local one, extracting and classifying behaviours based on the network traffic visible to the device, and a collaborative one, based on sharing and combining the information extracted by the former. The following sections will detail these two pathways, which are summarized in Figure 2.

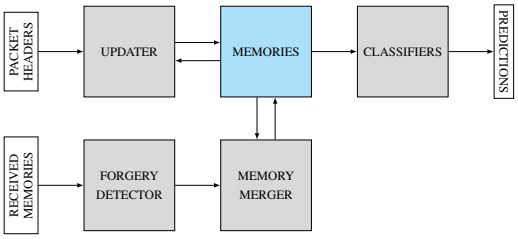


Fig. 2. A high-level view of the Metasoma process within a device. The top row of components represent the local detection flow, while the bottom row represents the collaborative aspect. The forgery detector will be introduced in Section IV.

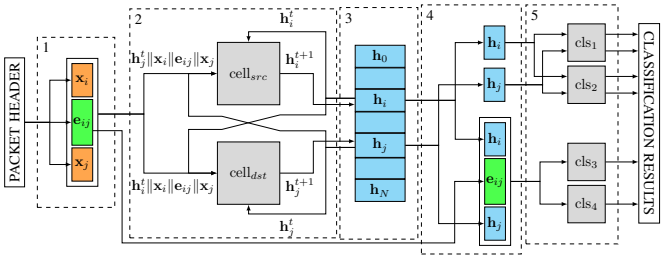


Fig. 3. Structure of LiMNet with two device classifiers and two packet classifiers. Colored boxes are: blue) memory representations; green) packet features; orange) device features; grey) trainable networks. Dashed numbered boxes are: 1) input feature map; 2) generalization layer; 3) memory; 4) output feature map; 5) response layer. Image reproduced from [4]

C. Local Bot Detection based on Packet Headers

For the local detection component, Metasoma employs the LiMNet architecture [4]. It consists of a memory network, which extracts features from the header of each packet and uses them to update the internal representation (i.e., memory) of the source and destination devices. This representation is then used to perform node or packet classification tasks to identify malicious behaviors.

LiMNet is very suitable for our use-case, as it has been shown to be not only very accurate in detecting botnets, but also much lighter than previous state-of-the-art models in terms of RAM and CPU pressure. In fact, it is shown that the entire model can fit in the private cache of a CPU core and can process thousands of packets per second while pinned to an individual core, without resorting to large batch sizes, multi-threading, or specialized AI accelerators. It can thus run effectively on low-power IoT devices without hampering their normal functions or power consumption [4].

Figure 3 presents a high-level view of the LiMNet architecture. The input feature map extracts key source node ($\mathbf{x}_i$), destination node ($\mathbf{x}_j$), and packet information ($\mathbf{e}_i j$) from the packet header. The generalization layer is composed of two recurrent cells that transform the previous memory representation of the source node $\mathbf{h}_i^t$ (resp. target node $\mathbf{h}_j^t$) and turn it into a new representations $\mathbf{h}_i^{t+1}$ (resp. $\mathbf{h}_j^{t+1}$), taking as input the extracted node and packet features and the previous memory representation of the target node (resp. source node).

These updated memory representations can then be directly used by ML classifiers to perform node classification, or they can be concatenated to the input packet features to feed packet classification models.

The original LiMNet is a centralized model that collects all packets in the network and thus builds memory representations based on complete information. In Metasoma, on the other hand, the model is run individually by each node with only a partial, local set of packets as input, and thus cannot provide the same level of accuracy.

D. Collaborative Bot Detection based on Memory Gossiping

To overcome this issue, Metasoma introduces memory sharing using gossip protocols, as introduced in Section II.

More specifically, in Metasoma, at regular intervals, each IoT device chooses a random subset of its local memories and sends them to another randomly-chosen IoT device. By only sending a fixed-size subset of the memories, the network bandwidth required by the protocol is kept constant, even when the number of memories stored by a device increases. While memories are typically small (between 128 and 256 bytes each), further savings can be achieved by compressing them before sending, exploiting the fact that ML models do not typically need the full 23 bits mantissa provided by single-precision floating point numbers [14].

The receiving device will then decompress the memories and merge them with its local ones. As each device is only aware of local traffic, and thus can only directly know the existence of nodes in its neighbourhood, each device stores memories for a different subset of the whole network. As such, the sender must tag each memory vector with the IP address of the node it represents, thus allowing the receiver to correctly match it with local data. If the IP address of a shared vector does not exist in the local memory of the receiver, then it is directly added to it. The receiver thus learns information regarding a device it was not aware of before. If, on the other hand, the IP address matches a local memory, the local and shared information need to be merged, so that both are preserved and the receiver can have a more complete knowledge of the device with that IP, based not only on the packets that were visible to the receiver, but also those that were visible to the sender, or to any other node that than shared its information with the sender. Thus, information is transitively propagated through the network.

Merging the local and shared representations for a device is a non-trivial task, because the two representations were built based on the device behavior during a potentially long period of time. The information stored in the two memories may be partially distinct (such as packets that were visible to the sender but not to the receiver or vice-versa) and partially overlapping (such as information that a third node gossiped to both the sender and the receiver). Furthermore, the time dimension is partially lost, as it is not possible to know when, or how many times, each memory was updated before being shared.

We thus employ a trainable ML component, which can learn the best transformation to perform during the merge operation. In our experiments, we use a single recurrent cell, with the exact same structure as those used in the memory updater, thus enabling the merger to mimic the type of "reasoning" employed by the updater. The recurrent cell transforms the old local memory of the receiver into the new combined memory, by taking as input the memory received via gossip.

E. Training Methodology

1) ML Tasks: While Metasoma is designed to perform inference in a decentralized setting, it still requires a centralized training procedure, in which the model is exposed to labelled network traffic including honest and malicious behavior belonging to all the classes of devices and botnets that the model should handle during inference. This *supervised learning* approach is made possible by the semi-controlled nature of IIoT environments and intuitively can provide better performance than unsupervised approaches.

We jointly train Metasoma on both device classification and packet classification tasks. The latter captures whether a specific communication is honest or not, and allows Metasoma to recognize that even a malicious node may participate in honest communications. On the other hand, the device labels are "sticky": as our datasets do not include recovery scenarios, once a device has participated in a malicious flow, it is marked as malicious for the remainder of the network trace.

This pushes both the local memory updater and the merger component of the model to preserve the memory of previous misbehaviors, thus limiting the ability of a patient attacker to interleave honest and malicious communications in order to bypass detection. Furthermore, this long-term memorization strategy has a substantial effect on which kind of attacks a malicious entity, aware of the implementation of Metasoma, may perform to bypass or weaponize the botnet detection system itself. This will be discussed in detail in Section IV.

2) Training Procedure: At its heart, Metasoma is a sequence-to-sequence recurrent model. Recurrent models are typically applied on datasets containing large numbers of separate sequences, some of which are used for training and others for evaluation. However, botnet detection datasets typically consist of a single long network trace. Naively feeding this to the model would cause issues of: 1) *scalability*, as no hardware parallelism is exploited by training on multiple sequences at the same time, and 2) *vanishing gradients*, as so many inputs are processed in the forward pass that the contributions of the earliest ones are completely lost during backpropagation.

A well-established solution to this problem is truncated backpropagation through time (p-BPTT) [15], a technique that consists of splitting the long input traces into multiple, partially overlapping windows, each of which is treated as a separate, independent input sequence for the model. Thus, multiple sequences can be combined into a batch for efficient training. Furthermore, this procedure allows us to set aside a random subset of these traces, to use as test set.

As already mentioned, Metasoma is unique in the way that representation vectors used internally by the model are randomly gossiped across the network. While this is the key to enable effective decentralized inference, it poses an additional, novel challenge: how to train the memory merger. This component must be trained in conjunction with the classifiers and updater, as its presence can potentially affect how the representations are built by the updater and interpreted

by the classifiers. However, the merger itself is affected by the gossiping procedure, and specifically by the frequency and amount of memories gossiped.

Thus, in order to perform the end-to-end training, each training sequence is augmented by interleaving the original network packets with memory gossiping packets. These are obtained by simulating the gossiping behaviour of the nodes involved in that specific training sequence, considering the hyperparameters controlling the gossiping behavior, and also considering the topology of the overlay network used for memory gossiping (described in Section IV).

Due to the complexity of the training procedure and the limited avenues for parallelization, training Metasoma is a relatively time-consuming task, even if the size of the model itself, in terms of number of parameters, is extremely small. However, this training procedure can be performed on specialized high-performance hardware, and only the small fully-trained model needs to be deployed on the actual IoT devices, which only perform lightweight inference operations.

F. Memory Retention

Having each IoT device store the latest memory of each peer ever known to them (either directly or via gossiping) is not feasible. First, it would not scale to large networks or those were devices are dynamically added or removed over time. Second, it would complicate the recovery process of infected devices. These would be typically taken offline and cleaned, before being reinstated in the network. However, other devices may still hold memories that flag the now-healthy devices as malicious, and may thus refuse to interact with them.

Both issues can be mitigated by having the IoT devices delete memories that have not been updated for a certain period of time, which indicates that the corresponding device is no longer active in the network. This leads to a tradeoff, where a more aggressive deletion strategy can substantially reduce the memory footprint of Metasoma in very dynamic networks and simplify the reintegration of cleaned devices, while at the same time potentially removing useful information for the detector and allowing malicious nodes to attempt slow attacks.

However, as explained in Section III-E, during the training process, the model learns to perform its predictions based on a limited view of the network trace, and therefore the impact of withholding stale information is limited. Furthermore, the reintegration of cleaned devices can be performed very quickly by assigning them a new identity, so that new memories are built for them when they join the network. The memories linked to their previous identities are then deleted at a later point, once the inactivity threshold is reached.

IV. Security Analysis

Metasoma aims to improve the security of existing applications on a decentralized IoT network by identifying botnets as they attempt to spread across the devices. However, due to its decentralized nature, Metasoma is itself an application running on the network, and any vulnerability in its design or implementation could be exploited by malicious entities who are aware of its deployment in the network. In this section, we analyze the security of Metasoma discuss the feasibility of potentially malicious attacks, and design mitigations to render those attacks infeasible.

A. Threat Model

As discussed in Section III-A, Metasoma is targeting IIoT with predefined organizations and network administrators. We assume that each organization has a certificate authority (CA) that validates the identities of devices in its network and issues certificates. We distinguish two types of actors: routers and IoT devices. We assume that the majority of routers and IoT devices are honest in the sense that they follow the Metasoma protocol. In addition, we assume that malicious (i.e., infected) routers can drop and forge network packets. Moreover, we assume that malicious (i.e., infected) IoT devices can:

- simulate arbitrary packets as inputs to their local Metasoma instances, which leads to non-factual local memories based on communications that did not happen;
- withhold (fully or partially) information from the system, such as by not inputting certain packets to their local Metasoma instances (producing incomplete memories), or by preventing the gossiping of certain memories produced by their local Metasoma instances;
- directly forge, store and gossip arbitrary memories in their local Metasoma instances, disregarding the model's normal flow and the trained parameters.

Furthermore, we assume that each honest device can only see its own inbound and outbound traffic and thus can only input to its local Metasoma instance. So, each honest device does not know about the communications that do not involve it directly. This leads to the result that any collusion or communication between infected devices is invisible to honest devices and to the overall Metasoma system, as the local Metasoma instances on the infected devices may be fully compromised. Based on this threat model, we identify three scenarios in which malicious actors that are aware of Metasoma can exploit it in their favor.

B. Scenario 1: Preventing Packets Forgery

a) Attack Scenario: A malicious router with honest devices in its sub-network may try to drop the network packets from such devices. In addition, it may try to forge them by changing packet headers (e.g., destination) which would impersonate other devices. In this section, we focus on forging packet headers. We give a detailed analysis of forging packet contents (i.e., memories) in Section IV-C.

b) Attack Impact: This attack would prevent Metasoma from detecting the ongoing spreading of a botnet, and would thus directly undermine the security of the entire IoT network.

c) Mitigations: To mitigate this attack, all packets must be authenticated and encrypted. In this way, any change to the contents, or any attempt to impersonate a different device, can be immediately detected. Both authentication and encryption can be performed in a relatively lightweight manner by employing asymmetric cryptographic protocols. Each device

is assigned both a public and a private key. By signing all outgoing packets with the private key, impersonation and tampering attempts by malicious intermediaries become impossible. By encrypting the packets with the public key of the receiver, intermediaries are prevented from even looking at the contents to gain information and, for example, choose to drop packets containing incriminating memories. Given the fact that IIoT deployments are typically controlled by either a single administrating entity or a small consortium (as discussed in Section III-A), it is possible for the organizations' certificate authorities to validate devices identities and generate certificates associated with each device public key, thus preventing malicious entities from generating new "throwaway" keys to prevent detection.

C. Scenario 2: Preventing Memory Forgery

a) Attack Scenario: A malicious node, or set of nodes, that are engaging in malicious communications with honest devices in the network, may try to prevent Metasoma from detecting the threat. They may do so by forging and gossiping to their victims specific memories which, once merged with the factual memories stored on those devices, would lead to a misclassification of the malicious node activities, thus preventing detection of the botnet.

b) Attack Impact: This attack would prevent Metasoma from detecting the ongoing spreading of a botnet, and would thus directly undermine the security of the entire IoT network.

c) Mitigations: As discussed in Section III-E, the use of sticky node labels during training should ensure that, if the local memory indicates that a certain device is malicious, this information will be preserved after the memory is merged with a gossiped one, no matter what other information the latter adds to the result. However, as this property is enforced by the training process, it only applies when all the memories involved follow the distribution experienced during training. That is, it only applies when all the memories are obtained through the normal application of the model to the packet flow. Memories that are directly forged by a malicious model are not bound to the same distribution and therefore may be optimized for the specific goal of erasing factual information about a target device stored in the memory of a victim.

Based on these observations, we propose to pair Metasoma with an anomaly detector trained on the memories produced during training. More specifically, we employ the memories produced during the last few iterations of the training procedure, when the weights of the model, and thus the memories it produces, have stabilized. As the training dataset includes both malicious and honest traffic, these memories will encompass both categories. However, as no attacker is forging memories during the training process, all the memories produced in that stage will belong to the distribution produced by the model, and for which the model should be able to guarantee the "stickyness" property. During decentralized inference, the anomaly detector would analyze any gossiped memory before it is processed by the merger, and would discard any anomaly, i.e. any memory which does not appear to belong to that same distribution, and could thus represent a forgery.

We employ a simple normalizing flow model, more specifically a Masked Autoregressive Flow [16]. This type of normalizing flow is particularly effective in density estimation tasks and is thus well suited for our use case. Given a memory as input, the forgery detector will output the density of the training memory distribution in that point of the space of possible memories. Should this density be below a certain threshold, the forgery detector would discard the input memory as too unlikely to appear in normal conditions, and thus potentially forged. By tuning this decision threshold, it is possible to achieve different trade-offs between blocking all forged memories (avoiding false negatives) and ensuring no honest memories are wrongly rejected (avoiding false positives). We evaluate these trade-offs in Section V-E.

D. Scenario 3: Preventing Eclipse Attacks

a) Attack Scenario: A malicious node M can take down an honest node H by forging packets/memories to incriminate it of malicious behavior. It may gossip a memory vector that, once merged with the vectors in honest nodes, will make the classifiers in these honest nodes classify H as malicious. This is possible without any tinkering with the Metasoma memories, by just faking the receipt of one or more malicious packets from the target honest device. Memories produced in this way are indistinguishable from honest memories even for the forgery detector, as they are correctly produced by the memory updater model, although based on a packet that was never actually sent.

So, in order to manipulate the behavior of an honest node H, a malicious node will perform an eclipse attack, possibly by colluding with other malicious nodes. An eclipse attack aims at surrounding an honest node H with malicious nodes as neighbors. So, node H will behave based on the information it receives from the surrounding malicious nodes which allows them to manipulate its actions.

b) Attack Impact: This attack would not directly help a botnet spread further into the network, and could have the opposite effect of alerting network operators of the presence of a threat in the system. However, an attacker whose goal is not to spread its botnet and infect additional devices, but rather to take down said devices, could accuse honest devices of being malicious as a way to deliver targeted Denial of Service attacks against those devices. This would be particularly effective in the presence of automated threat response systems designed to immediately isolate devices that are reported to be misbehaving.

c) Mitigations: In order to mitigate this attack, we need to stop malicious nodes from incriminating honest nodes. For this purpose, we keep a sparse network overlay and only allow neighbors in the overlay to incriminate each other. In this way, an attacker who manages to take control of a device will only be able to take down honest devices in its direct neighborhood, greatly reducing the chances of a denial of service. This method also helps a potential recovery system to identify all potentially infected devices and restore them.

This approach is based on a key assumption that good Metasoma performance can be achieved without gossiping

memories too often, so that each time a gossiping operation is performed, it is feasible to transfer heavier data using a TCP connection, rather than sending a single lightweight UDP message. This assumption is backed up by the experiments shown in Section V-D.

To maintain a very sparse connectivity pattern (and thus a small "blast radius" where an intruder can take down honest devices), we maintain two overlay networks: a "functional" overlay network connecting those IoT devices that need to be connected for maintaining the proper function of the deployment (e.g., a lightbulb must be connected to its relevant switch), and a "gossiping" overlay network, which has a random structure and, therefore, it is optimal to spread memories in the fastest way. Both networks are built via peering operations, and cryptographic proofs are used to ensure that any node is able to prove that it is connected to another node in one of the overlays.

When a neighbor of a node A gossips its memory of node A, it will attach to it the proof (e.g., a signed token) it received from node A during peering, thus proving its right, as a neighbor, to update that memory. When another node, not connected to A, receives from one of its peers the memory of node A, it will store it along with the proof that came with it. If said node later receives a new memory for node A, with a different proof (and thus a different "source"), it will merge those two memories and will then store the merged memory along with the originals and their two proofs. In this way, the node can prove that it merged them in an honest way when gossiping this memory. Thus, with this mechanism, the only way a malicious device can affect the memory of node A to wrongfully accuse it is by being a neighbor of node A.

Thus, in order to target a specific honest node H, a malicious entity needs to take control of one of its neighbors, which could be either: i) a node that is functionally connected to H, or ii) a node that is connected to H in the "gossiping" network. To minimize the chance of the malicious entity succeeding, we need to ensure that the structure of the gossiping network cannot be easily predicted or manipulated by malicious devices. On the other hand, the structure of the functional network cannot be altered, as it is necessary for the devices to fulfill their tasks. It is worth noting that taking control of a neighboring node of the target node is out of our scope as a malicious node could exploit vulnerabilities in the software that the neighbor runs (e.g., web server, firmware, etc.).

Neighbors Selection. The aim of the neighbors' selection protocol is to selectively connect to honest nodes to avoid eclipse attacks. One fundamental observation to defend against eclipse attacks is the randomness of selecting neighbors. In other words, if node M (a malicious node) wants to connect to node H (an honest node), node M will be targeting node H, so the connection request will not be random. So, there is a need to check the randomness of the request. Therefore, each device in Metasoma generates a private salt ϕ^*, which can be any random number. This salt can optionally be dynamic, in the sense that the device can choose to change salt every t seconds, where t is a time window selected by the device itself. Additionally, Metasoma employs a public salt ϕ that changes every $t^{'}$ seconds (e.g., 30 seconds in our implementation). In addition, ϕ needs to be verifiably random [17]. For this purpose, we use drand[4], a distributed randomness beacon. In addition, to make the process challenging, each node will have two lists of neighbors: the accepted ones and the chosen ones. The accepted list consists of the nodes that choose the peer to be a neighbor while the chosen list consists of the nodes that the peer chooses to be neighbors with. For the latter, we use the following distance[5] function:

$$d(nodeM, nodeH, \phi) = hash(nodeM) \oplus hash(nodeH + \phi) \quad (1)$$

We rely on the *exclusive OR* distance since it is a widely adopted metric in structured peer-to-peer systems, such as the networks adopting Kademilia-based distributed hash tables [18] (e.g., BitTorrent). It is worth noting that similar protocols were applied in public distributed ledger technology such as IOTA 2.0 Auto Peering Protocol [19].

When node M wants to connect to node H, first, it keeps a list of potential nodes that it can connect to, sorted in ascending order by their distance. If node H is at the top of the list, node M sends the peering request. When a node H receives a peering request, it measures $d(nodeH, nodeM, \phi_*)$. If the distance is lower than the distance of an existing neighbor, the new neighbor is added and the farthest neighbor is dropped. So, if an attacker wants to perform an eclipse attack on another node, it needs to mine the private salt ϕ_* of the victim node H to be in its neighborhood. It is worth noting that the accepted and the chosen list should be limited in size (to allow dropping the farthest nodes). We study the effect of the neighborhood size in Section V-F.

V. Experimental results

A. Datasets

The original centralized LiMNet [4] was evaluated on two datasets: MedBIoT [2] and Kitsune [20]. These datasets were selected because they gather network traces from the early phases of infection which is the focus of LiMNet. The MedBIoT dataset [2] contains network traces including the communication between the bots and the C&C servers of three botnets: Mirai, Bashlite/Gafgyt, and Torii. The botnets were injected in a medium-size network of 83 devices. Some devices are emulated and others are real including smart locks, fans, switches, and light bulbs. The Kitsune dataset [20] contains network traces of early infection phases by Mirai botnet. The latter was injected in a small-size network that consists of 3 PCs and 9 IoT devices, including a thermostat, a baby monitor, a webcam, low-cost security cameras, and doorbells. The details of the datasets are summarized in Table I.

As explained in Section III-E, each dataset is split into multiple subsequences. 85% of these sequences are employed in the training process, while the remaining 15% are used as a test set to compute the scores reported in this section.

[4]https://drand.love/

[5]It is worth noting that the distance represents a numerical (i.e., XOR) distance between hashes (i.e., $hash(nodeM)$ is the SHA-256 hash of the public key of $nodeM$) and not the geographic distance

TABLE I
DATASETS SUMMARY

Dataset	Devices	Packets	Botnets
MedBIoT	83	17,845,567	Mirai, Bashlite, Torii
Kitsune	12	764,137	Mirai

TABLE II
AUROC SCORES ON THE KITSUNE DATASET

Model	D	AUROC		
		packet mal.	device mal	device atk.
LiMNet	32	99.7	81.2	82.2
Metasoma	32	99.2	80.1	81.6
LiMNet	64	99.8	82.0	82.9
Metasoma	64	99.2	81.4	82.4

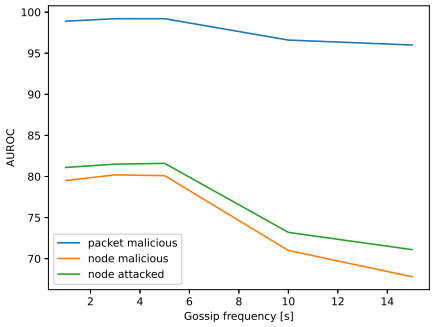


Fig. 4. Effect of memory gossiping frequency on the detection performance of Metasoma.

B. Hyperparameters

In all the experiments, we maintain hyperparameters similar to those presented as the best in [4]. More specifically, we employ the GRU recurrent cell in both the memory updater and merger components, and we test with memories of size $D = 32$ and $D = 64$. For the p-BPTT training process, we use a sequence length of 5000 and a stride of 500, which ensures that every packet is seen in different positions of several sequences. For the gossiping component, unless otherwise noted, the frequency at which each device performs one round of gossip is set to 5 seconds. The maximum number of memories shared in one round is capped at 8, in order to limit the bandwidth required by the protocol.

C. Effectiveness of Metasoma in Botnet Detection

To understand the impact of decentralization on the effectiveness of Metasoma in botnet detection, we compare it with LiMNet[6], which uses an identical architecture to build and classify memories, but does so in a centralized environment. We preserve as much as possible the same conditions as in [4], thus evaluating our system on three binary classification tasks: whether a certain packet belongs to a malicious communication flow, whether a certain node is malicious (i.e. has been infected) and whether it is currently under attack (i.e., a malicious device is trying to spread its botnet to it).

However, our setup presents one major difference with respect to the original LiMNet paper: as explained in Section III-E, we employ "sticky" device labels, which put greater emphasis on longer-term memorization of malicious or potentially malicious behaviors.

Similarly to what reported in [4], we also find that the MedBIoT dataset, despite its wide variety of botnets and large number of devices, represents a too simple challenge for deep learning models such as LiMNet and Metasoma, which all achieve over 99% AUROC in all three tasks. We therefore report our detailed results on the Kitsune dataset in Table II, comparing LiMNet and Metasoma.

LiMNet has access to a complete and ordered stream of all network packets exchanged in the network, and thus the memories it uses for predictions always contain all the available information on the target devices. In Metasoma, on the other hand, the local memories in each device are built from incomplete information. While the gossiping protocol eventually leads to all exchanged packets being included in the memories of all devices, this process discards potentially important information about the global order of the packets and adds noise and non-determinism to the memories. Thus, it is expected for Metasoma to perform worse than LiMNet, being a drop in prediction quality the price to pay to achieve decentralization.

Our experimental results show that this price is relatively small. This validates the feasibility of exchanging internal model representations to perform decentralized inference and means that Metasoma can be deployed in IIoT settings to replace centralized botnet detection solutions, without compromising the security of the devices.

Note that the results for LiMNet in Table II are lower than those originally reported in [4]. This is due to our use of "sticky" device labels, which puts greater emphasis on longer-term memorization of malicious or potentially malicious behaviors, as explained in Section III-E. This makes the task of device classification impossible for models that do not memorize past behavior and makes it more challenging even for LiMNet.

D. Effect of Gossiping Frequency on Detection Accuracy

Metasoma relies heavily on memory gossiping to propagate information between the devices, ensuring that each of them has access to a relatively complete and relatively up-to-date representation of its peers. We therefore investigate how the performance of Metasoma is affected by the frequency at which each node gossips its memories to a random peer.

Figure 4 shows how the AUROC scores of Metasoma on the three tasks presented in Section V-C at different gossiping frequencies (1, 3, 5, 10 and 15 seconds). It can be seen that the performance increases dramatically up to a frequency of 5 seconds, especially for the device-level tasks, which are more sensitive to the staleness of the memories. The packet classification task, on the other hand, has access to the features of the packets, and thus is less dependent on the exact contents of the memories. Further increasing frequency does not lead to any meaningful gains, and actually results in measurable, if minor, losses when approaching 1 second. This may be due to

[6]We used the source code that LiMNet authors made available on Github (https://github.com/lodo1995/LiMNet)

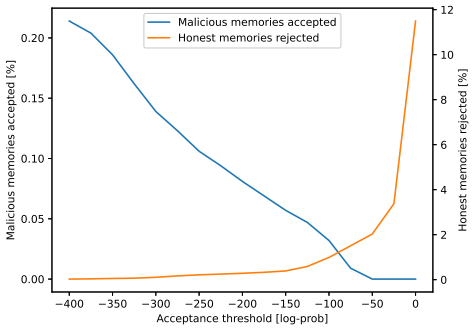


Fig. 5. Probability of the forgery detector accepting forged memories and rejecting honest memories for different choices of the acceptance threshold.

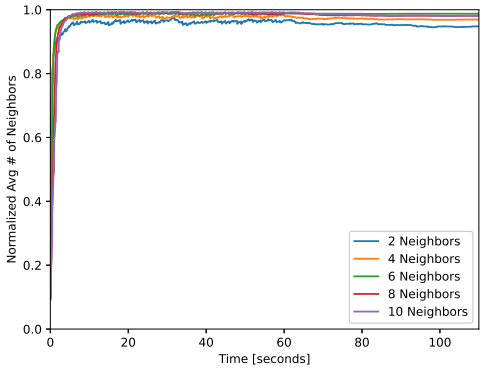


Fig. 6. Neighbourhood sizes: convergence analysis

an increase in the level of noise introduced in the memories by the frequent merging operation, as the model may not be capable of properly handling large amounts of overlapping information received from multiple sources at high frequency.

This result indicates that Metasoma is unlikely to introduce significant network overheads in an IoT deployment, as it does not require a high communication frequency.

E. Effectiveness of Metasoma in Forged Memory Detection

To measure the effectiveness of the Normalizing Flow component of Metasoma in detecting forged malicious memories, we simulate a potential attacker who has access to the trained memory merger component and classifier and has perfect knowledge of the memories on the victim device.

More specifically, given a merger function $merger : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^D$, a classifier function $cls : \mathbb{R}^D \rightarrow [0, 1]$ (where 0 is honest and 1 malicious), and a target memory h, the attacker trains a model $forger : h \mapsto h'$ with the objective of minimizing $cls(merger(h, h'))$, thus ensuring that any malicious behavior that the victim had recorded in the memory h is forgotten after the victim merges it with the memory h' received from the attacker.

In our experiments, we use a simple 3-layers feed-forward neural network as the forger, with all layers having size D, ReLU activations in the hidden layers and hyperbolic tangent activation in the final layer, the latter to match the activation used by the recurrent units in Metasoma. We observe that this model easily achieves a 99% effectiveness in producing false negatives. That is, when fed a memory that Metasoma would classify as representing a malicious device, in 99% of the cases the forger outputs a new memory which, once merged with the input memory, produces a result that is classified as a honest device by Metasoma. This result is expected, as it is very easy to manipulate the output of a machine learning model when its weights are known and the inputs can be freely manipulated.

With this trained attacker, we then proceed to add our Normalizing Flow-based forgery detector to Metasoma and evaluate its effectiveness in identifying forged memories at different probability density thresholds. The results are summarized in Figure 5. Even when setting a very low threshold, which ensures virtually no honest memories are rejected, only a small fraction (around 0.2%) of forged memories are accepted. On the other hand, if the threshold is raised to the point where all forged memories are correctly identified, only around 2% of all honest memories are wrongly discarded. These results indicate that the forgery detector is extremely accurate, allowing the threat of forged memories to be neutralized without severely harming the gossiping protocol.

F. Effectiveness of the Eclipse Attack Mitigations

In order to evaluate the effectiveness of our proposed mitigations to eclipse attacks and their effect on the network, we have simulated our solution in a network of 1000 devices. As explained in Section IV-D, each peer has two types of neighbors: chosen and accepted. We have run our experiment with different neighborhood sizes (i.e., 2, 4, 6, 8, and 10) where each neighborhood size is divided equally between the chosen and accepted neighbors. In addition, our public and private salts change every 30 seconds.

a) Convergence: To analyze the effect of changing neighborhood size on the network, we evaluate the network convergence metric with different neighborhood sizes. In Figure 6, we show the convergence in terms of normalized average number of neighbors over time. We observe that the peers have on average a full neighborhood (i.e., chosen and accepted) after 5 seconds. In other words, the peers quickly build their neighborhoods. So, they stay connected and keep sending and receiving memories regardless of the neighborhood size and the identity of the actual peers in the neighborhood.

b) Link Survival: Since the neighbors selection algorithm was proposed as a possible mitigation for the eclipse attack, we evaluate the link survival in terms of probability over time. In other words, we evaluate the probability of a link surviving (i.e., persisting in a peer's neighborhood) for a period of time. We recall that our public and private salts change every 30 seconds which means that every peer's neighborhood changes every 30 seconds. In Figure 7, we see that after 30 seconds the probability that a link survives is almost 0. So, if a node A is connected to a node B in the first 30 seconds timeframe, the probability that a node A stays in the neighborhood of node B in the following 30 seconds is almost 0. In other words, if a malicious node M wants to perform an eclipse attack on node H, it needs to generate a hash that is closer than the farthest node in node H neighborhood in less than 30 seconds. It is an impractical attack because the search space is large,

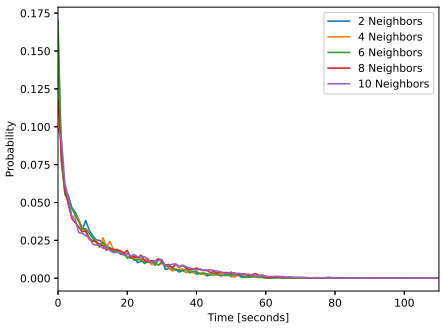


Fig. 7. Link survival probability over time

and the honest nodes could implement a trivial mitigation of brute-forcing attacks by introducing a rate limit.

VI. Related Work

The field of IoT botnet detection has gone through extensive research in recent years. Existing solutions in this area can be categorized based on several aspects: traditional vs machine learning-based detection, centralized and decentralized architecture, and early-stage vs attack-stage detection.

There are two main research directions in centralized IoT botnet detection: traditional and machine learning-based detection. In traditional detection, several approaches [21], [22] exploit graph theory for botnet detection since IoT botnets frequently communicate with each other, thus building communities. Community detection algorithms like the one shown in [22] are proven to be effective in detecting botnets. Similarly, other approaches [23], [24] exploit statistical features of botnets, such as frequency of communication and ports used, to detect botnets. Such approaches identify correlations among bots to infer botnet affiliations.

Similarly, several approaches [4], [25]–[27] exploit network traffic analysis in machine learning-based detection to identify infected nodes. These approaches feed ML models with features of network packets such as source and destination nodes, communication protocol, packet size, etc., to classify packets and/or nodes as benign or malicious. For this purpose, different ML techniques have been employed, including deep autoencoders fed with statistical packet features [28] and recurrent models fed with packet headers [25], [26]. Another emerging direction in IoT botnets detection is the usage of deep learning models in a federated learning setting like the work in [27]. It relies on packet headers to detect IoT botnets attacks by training local models and sharing the model updates with a central server for aggregation. However, few works have focused on early-stage botnet detection. The work in [2] presents a dataset for botnet early-stage detection in IoT. The authors evaluated the performance of simple ML techniques such as k-Nearest Neighbors classifiers, decision trees, and random forests in classifying packets as benign or malicious. In the same line of research, the work in [3] used recurrent models that were shown to be effective for early-stage detection to classify packets based on low-dimensional representations of network packet features.

Another research stream focuses on decentralizing botnet detection solutions. The work in [29] decentralized the PeerHunter community-based botnet detection algorithm [22] by running it as a smart contract in the blockchain. In addition, the work in [30], [31] leveraged blockchain to enhance collaborative threat intelligence in IoT. Furthermore, the authors of [32] proposed a collaborative anomaly detection solution with a focus on botnet detection. They developed their own blockchain to collaboratively exchange threat intelligence and train an extensible Markov model (EMM) based on consensus among IoT devices.

Metasoma differs from the aforementioned approaches on many levels. First, Metasoma focuses on classifying nodes, packets, and attack status. However, it differs from [32] since it is decentralized and it does not require a central entity. Second, Metasoma is a lightweight model that can be deployed in IoT devices with limited computational power, unlike the discussed approaches that are based on heavy models or community detection algorithms. Third, the blockchain-based approaches (i.e., [29]–[32]) require IoT devices to run blockchain clients, which adds communication and computational overhead to devices. Metasoma reduces the communication overhead since it has a limited number of neighbors at each point in time and does not run consensus or other blockchain-related tasks. Fourth, Metasoma does not use any trusted execution environment unlike the work in [32] that uses TrustZone for key management for their blockchain. Finally, Metasoma provides a detailed security analysis of possible attacks and their mitigations.

VII. Conclusion

This paper introduces Metasoma, which, to the best of our knowledge, is the first decentralized, deep-learning based approach for supervised early-stage botnet detection. Metasoma provides virtually the same detection capabilities as previous state-of-the-art approaches while removing the need for expensive and potentially unfeasible centralized monitoring solutions. We also show how a smart attacker may try to exploit the weaknesses of such a decentralized system, and we design and evaluate mitigations to prevent these threats. Overall, Metasoma provides a compelling alternative to existing solutions and will hopefully lead to the exploration of novel research directions in the fight against botnets.

Metasoma opens a number of interesting future research directions. In particular, it employs a relatively shallow architecture, with each component (updater, merger and classifier) consisting of a single recurrent or dense layer. Better prediction accuracy may be achievable by increasing the depth of one or more of these components. This would come at the expense of inference speed and size of the trained model, therefore putting more strain on the limited resources of IIoT devices. The design of deep models that are highly expressive and at the same time resource-efficient is a very active research area.

Related to the previous issue is that of training time, which has partially affected the breadth of experiments that we performed. The use of deeper models would further exacerbate this issue, and thus an interesting direction for future work

consists of exploring more efficient training strategies for models with heterogeneous input sequences, such as Metasoma.

As already discussed in [4], there is a lack of suitable datasets for this task in the academic community. Most botnet detection datasets focus on the attack phase of the malware, rather than on the initial spreading phase. The largest available dataset that focuses on botnet spreading, MedBIoT, is unfortunately not challenging enough to allow meaningful comparisons and to pinpoint the limitations of various deep models. The collection of new datasets for this purpose should be a priority of the research community, in order to foster the development of novel techniques in the area.

REFERENCES

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} security symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.

[2] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nomm, "Medbiot: Generation of an iot botnet dataset in a medium-sized iot network," in *ICISSP*, 2020.

[3] H. Alzahrani, M. Abulkhair, and E. Alkayal, "A multi-class neural network model for rapid detection of iot botnet attacks," *International Journal of Advanced Computer Science and Applications*, vol. 11, 01 2020.

[4] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and Š. Girdzijauskas, "Limnet: Early-stage detection of iot botnets with lightweight memory networks," in *European Symposium on Research in Computer Security*. Springer, 2021, pp. 605–625.

[5] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1410.3916

[6] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964–3979, 2021.

[7] A. Montresor, "Gossip and epidemic protocols," *Wiley encyclopedia of electrical and electronics engineering*, vol. 1, 2017.

[8] G. Mega, A. Montresor, and G. P. Picco, "Efficient dissemination in decentralized social networks," in *2011 IEEE International Conference on Peer-to-Peer Computing*, 2011, pp. 338–347.

[9] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[10] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip learning as a decentralized alternative to federated learning," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2019, pp. 74–90.

[11] A. A. Alkathiri, L. Giaretta, S. Girdzijauskas, and M. Sahlgren, "Decentralized word2vec using gossip learning," in *23rd Nordic Conference on Computational Linguistics (NoDaLiDa 2021)*, 2021.

[12] L. Giaretta and Š. Girdzijauskas, "Gossip learning: Off the beaten path," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1117–1124.

[13] I. Butun, M. Almgren, V. Gulisano, and M. Papatriantafilou, *Industrial IoT*. Springer, 2020.

[14] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[15] H. Jaeger, "Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the" echo state network" approach," 2002.

[16] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," *Advances in neural information processing systems*, vol. 30, 2017.

[17] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[18] P. Maymounkov and D. Mazières, "Kademilia: A peer-to-peer information system based on xor metric, 1st intl," in *Workshop on Peer-to-Peer Systems*, 2002.

[19] S. Popov, H. Moog, D. Camargo, A. Capossele, V. Dimitrov, A. Gal, A. Greve, B. Kusmierz, S. Mueller, A. Penzkofer *et al.*, "The coordicide," *Accessed Jan*, pp. 1–30, 2020.

[20] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *Network and Distributed System Security Symposium (NDSS)*, 2018.

[21] B. Coskun *et al.*, "Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 131–140.

[22] D. Zhuang *et al.*, "Peerhunter: Detecting peer-to-peer botnets through community behavior analysis," in *2017 IEEE Conference on Dependable and Secure Computing*, Taipei, 2017, pp. 493–500.

[23] Z. Yang *et al.*, "P2p botnet detection based on nodes correlation by the mahalanobis distance," *Information*, vol. 10, no. 5, p. 160, 2019.

[24] J. Li *et al.*, "Distributed threat intelligence sharing system: A new sight of p2p botnet detection," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, Riyadh, Saudi Arabia, 2019, pp. 1–6.

[25] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the internet of things using deep learning approaches," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[26] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, "An lstm-based deep learning approach for classifying malicious traffic at the packet level," *Applied Sciences*, vol. 9, no. 16, 2019.

[27] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in iot-edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2021.

[28] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[29] A. Lekssays, L. Landa, B. Carminati, and E. Ferrari, "Pautobotcatcher: A blockchain-based privacy-preserving botnet detector for internet of things," *Computer Networks*, vol. 200, p. 108512, 2021.

[30] P. Kumar, R. Kumar, G. P. Gupta, R. Tripathi, and G. Srivastava, "P2tif: a blockchain and deep learning framework for privacy-preserved threat intelligence in industrial iot," *IEEE Transactions on Industrial Informatics*, 2022.

[31] S. M. Sajjad, M. R. Mufti, M. Yousaf, W. Aslam, R. Alshahrani, N. Nemri, H. Afzal, M. A. Khan, and C.-M. Chen, "Detection and blockchain-based collaborative mitigation of internet of things botnets," *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[32] T. Golomb, Y. Mirsky, and Y. Elovici, "Ciota: Collaborative iot anomaly detection via blockchain," *arXiv preprint arXiv:1803.03807*, 2018.