

AutoBotCatcher: Blockchain-based P2P Botnet Detection for the Internet of Things

Gokhan Sagirlar, Barbara Carminati, Elena Ferrari

University Of Insubria, Italy {gsagirlar, barbara.carminati, elena.ferrari}@uninsubria.it

Abstract—In general, a *botnet* is a collection of compromised internet computers, controlled by attackers for malicious purposes. To increase attacks' success chance and resilience against defence mechanisms, modern botnets have often a decentralized P2P structure. Here, IoT devices are playing a critical role, becoming one of the major tools for malicious parties to perform attacks. Notable examples are DDoS attacks on Krebs on Security¹ and DYN², which have been performed by IoT devices part of botnets.

We take a first step towards detecting P2P botnets in IoT, by proposing *AutoBotCatcher*, whose design is driven by the consideration that bots of the same botnet frequently communicate with each other and form communities. As such, the purpose of *AutoBotCatcher* is to dynamically analyze communities of IoT devices, formed according to their network traffic flows, to detect botnets. *AutoBotCatcher* exploits a permissioned Byzantine Fault Tolerant (BFT) blockchain, as a state transition machine that allows collaboration of a set of pre-identified parties without trust, in order to perform collaborative and dynamic botnet detection by collecting and auditing IoT devices' network traffic flows as blockchain transactions.

In this paper, we focus on the design of the *AutoBotCatcher* by first defining the blockchain structure underlying *AutoBotCatcher*, then discussing its components.

Index Terms—Blockchain, Internet of Things (IoT), Security, P2P Botnets, Botnet Detection.

I. INTRODUCTION

IoT technology has been growing chaotically on many environments, such as homes and factories [1], connecting exceptionally large number of devices, expected to increase up to 130 billion devices in 2030.³ Yet, this increasing popularity has made IoT devices a powerful amplifying platform for cyberattacks [2]. In fact, IoT devices are often simple products that due to the limits of available constrained-resources do not take security as primary goal. As such, they represent a rather easy target to attackers and the weakest link in the security chain of modern computer networks [2]. As proof of this, a recent study from HP found that more than 70% of IoT devices do not have passwords with sufficient complexity and use unencrypted network services, resulting in being easy targets for attackers.⁴

In such a vulnerable environment, attackers can easily gain access to insecure IoT devices, and inject malicious softwares, *malware*, to control them or to steal confidential information

[3]. Today, one of the most relevant threat posed by malware in IoT is represented by malicious *botnets*.⁵ A botnet is a collection of compromised internet computers being controlled remotely by attackers for malicious and illegal purposes [4]. For example, some recent Distributed Denial of Services (DDoS) attacks on *Krebs on Security* and *DYN* were due to malware named Mirai [5], that uses IoT devices as botnets to generate extensive amount of network traffic, more than 1 Tbps. Additionally, such botnets have been commoditized by malicious parties, known as *booters* [6], that offers DDoS as a service. Booters exploit compromised IoT devices to send attack packets to a target victim, in order to interrupt its service or shut it down. Given that, botnets capable of using tens of thousands of IoT devices pose huge threats to online services' security and privacy.

Botnets. Let us examine in more details the main elements of botnets. Briefly, a typical botnet consists of [4]: i) several *bots*, that is, infected machines running the bot executable; ii) a *Command and Control (C&C) server*, able to control every bot; and iii) a *botmaster*, which is the malicious party controlling the botnet via the C&C server. Early botnets followed a centralized architecture, where the botmaster manages bots via the central C&C server. To increase resilience of their attacks against defence mechanisms, more recent botnet architectures evolved into decentralized P2P architectures. Today, decentralized P2P botnet topologies are able to utilize regular bots as C&C servers [4], thus eliminating the single point of failure problem. On the other hand, this makes P2P botnets harder to being detected and stopped, as botmasters are able to send attack commands through various channels.

AutoBotCatcher. The design of *AutoBotCatcher* is driven by the consideration that bots of a same botnet frequently communicate with each other and form communities [7], [8]. As such, the purpose of *AutoBotCatcher* is to dynamically analyze communities of IoT devices, formed according to their network traffic flow (see problem statement in Section II), to detect botnets. Specifically, it is a blockchain-based P2P botnet detection mechanism for IoT that makes use of two main actors, namely: *agents* and *block generators* (see Section II for more details). Where, agents are entitled to monitor IoT network traffic flows in their subnets, and send collected traffic information as blockchain transactions. In contrast, by using collected network traffic flows, block generators (i.e., trusted big entities in IoT domain) aim at modeling *mutual*

¹krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos

²dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack

³cdn.ihs.com/www/pdf/IoT_ebook.pdf

⁴go.saas.hpe.com/1/28912/2015-07-21/32bhy3/28912/69168/IoT_Report.

pdf

⁵For the rest of paper, we use the term botnet to refer malicious botnets.

contact information of IoT devices (i.e., connections between IoT devices) and generating mutual contacts graph. This graph is then exploited to detect communities (see Section III-A).

In particular, AutoBotCatcher uses Louvain method [9] to perform community detection on mutual contacts graphs. Since mutual contact information of IoT devices evolves over time, new snapshots of the mutual contacts graph are periodically generated. To this end, AutoBotCatcher exploits states of a BFT blockchain in order to store snapshots of the mutual contacts graph (see Section IV). AutoBotCatcher's BFT blockchain is a permissioned blockchain, where a set of pre-identified block generators generate blocks and participate in the consensus process (see Section III-C). Thus, network data stored on the blockchain is only accessible to block generators.

Why blockchain? To enable multiple parties to collaborate for botnet detection, we chose to use blockchain rather than a centralized system given the benefits blockchain might bring. Thanks to its distributed consensus protocol⁶, blockchain platform does not require a central trusted party to validate the correct execution of the collaborative process (aka botnet detection), and ensure transparency on collected snapshots of communities of IoT devices overcoming the possible lack of trust among parties involved in the botnet detection (see Section III-C). Moreover, as a state transition machine, blockchain lets us model the whole botnet detection process as a set of shared application states (aka states of parties collaborating in the botnet detection). This allows AutoBotCatcher to perform dynamic and collaborative botnet detection on large number of IoT devices.

Contributions. The main contributions described in this paper can be summarized as follows:

- ★ a first blockchain-based botnet detection architecture for IoT;
- ★ dynamic and collaborative approach for botnet detection and prevention;
- ★ dynamic community detection with the help of blockchain technology.

Outline. The remainder of this paper is organized as follows. In Section II, we present the considered problem statement and the main entities involved in AutoBotCatcher. We provide background information on blockchain technology, mutual contacts graph, and community detection approaches in Section III. In Section IV, we introduce the blockchain paradigm defined for AutoBotCatcher. We detail the design of AutoBotCatcher in Section V. Section VI discusses related work, whereas Section VII concludes the paper.

II. SYSTEM MODEL

In this section, we introduce the problem statement and the main entities of AutoBotCatcher.

Problem Statement. We assume to have a network of IoT devices, gateways, and some external hosts that communicate with IoT devices (such as device vendors' servers, cloud

services). IoT devices are connected to the internet, they sense and process data, and communicate with other IoT devices or external hosts. Botmasters compromise IoT devices and make them part of their botnets for malicious purposes, such as performing DDoS attacks. On the other hand, gateways, such as Dell Edge Gateways⁷, are located in network boundaries and monitor the internet traffic to/from IoT devices within their networks, referred to as their *subnet*. We assume that gateways are secure and trusted devices, as such, they behave as expected, and cannot be compromised by botmasters.

AutoBotCatcher targets P2P botnets, where all bots potentially can be utilized as C&C servers by botmasters, and performs community analysis on network traffic flows of IoT devices to detect botnet communities. We assume that a botnet community is a group of compromised IoT devices that frequently communicate with each other and with the same set of botmasters. AutoBotCatcher relies on *mutual contacts* information of IoT devices, which refers to shared connections between a pair of IoT devices and/or other hosts. For example, let us assume *Host A*⁸ is connected to *Host C*; given that, if *Host B* is also connected to *Host C* then *Host A* and *Host B* share a mutual contact, that is, *Host C*. As discussed in [7] and [8], bots of a P2P botnet communicate with at least one mutual contact with very high probability, therefore mutual contacts can be exploited for botnet detection [7]. Given that, AutoBotCatcher exploits mutual contact information of IoT devices in performing botnet community analysis (see Section V).

Our assumptions on the threat model are as follows: IoT devices can become part of a botnet anytime; new types of botnets may emerge in the network; botmasters encrypt C&C channels, and therefore DPI techniques are not suitable; botnets tend to hide their operations and botmasters try to stay as stealthy as possible [4], where botnets are able to manipulate characteristics of bot traffic, and thus they are able to make network flow traffic signature based defense approaches ineffective; botnets are in their waiting stage, where bots are joined to the C&C network and wait for commands from the botmaster, thus their malicious activity may not be easily observable. The goal of AutoBotCatcher is to dynamically identify IoT devices and other hosts in the network that are part of botnets.

Entities. AutoBotCatcher consists of two main entity types:

- *Agents*: They are typically gateway devices that are deployed in the network boundaries. AutoBotCatcher's agents monitor network traffic flows of IoT devices in their subnet and take actions, such as: generating network-data transactions (NTs) (see Definition 1) and forcing infected devices to shut down. In AutoBotCatcher agents are considered trusted and honest.
- *Block Generators*: This role is played by big entities in IoT, such as device vendors, internet service providers (ISPs), security/privacy regulators. We assume that such big entities

⁶With the assumption that more than two-thirds of the block generators are honest.

⁷dell.com/us/business/p/edge-gateway

⁸Here, *Host* refers to both IoT devices and other hosts in the network.

devote enough computing and network resources to take block generator role in P2P botnet detection process (see Section V). For effective botnet detection and prevention, collaboration of different device vendors and ISPs is very important, as recent IoT botnets, such as Mirai and Hajime, infected IoT products from various vendors.⁹ In fact, malware behind those botnets are adaptable to the various device architectures, such as ARM and Intel, and to different products, and they were effective in all around the world. Therefore, block generators collaborate to achieve large-scale defense and protection from botnet threat without trusting each other with the help of a permissioned BFT blockchain. We assume that block generators have enough computing, storage and network resources available to devote to P2P botnet detection process operations using blockchain (see Section V).

III. BACKGROUND

In this section, we provide background information needed to understand the rest of the paper. To this end, first, we explain the mutual contacts graph concept; then, we provide an overview of community detection approaches; and, finally, we briefly describe blockchain technology.

A. Mutual Contacts Graph

In AutoBotCatcher, mutual contacts graphs are exploited as a graph based data representation of the mutual contacts of IoT devices and other hosts in the network. We denote a mutual contacts graph as $G = (V, E)$, where IP addresses of IoT devices and other hosts are *vertices* (V). Vertices share an *edge* (E), if they have at least one mutual contact. Edges are bidirectional and weighted, where number of mutual contacts between vertices is the weight of the edge between them. As such, by referring to the example of mutual contacts given in Section II and assuming that there are only three hosts in the network, *Hosts A, B, C* are vertices in the mutual contacts graph, where *Host A* and *Host B* share an edge with weight 1, as they share the mutual contact *Host C*. In AutoBotCatcher, the whole topology of the mutual contacts graph is represented by a 2 dimensional weighted adjacency matrix, referred to as *mutual contacts matrix (MCM)*, whose element MCM_{ij} indicates the number of mutual contacts between vertices i and j .

B. Community Detection

Bots of the same botnet use similar C&C channels and share the same messages [8], [10], as such they are much likely to share many mutual contacts [7] than legitimate P2P hosts. Therefore, P2P bots show community behaviours and form community structures that can be useful for botnet detection. Given that, AutoBotCatcher performs *community analysis* on mutual contacts graphs.

Community detection methodology. In AutoBotCatcher, accurate and fast detection of communities in the mutual contacts graph is of great importance for proper botnet detection.

Literature offers several community detection approaches for systems modeled as graphs (see [11] for more details). The main objective of these methods is to find good partitions on the graphs as possible communities in the network. How to measure goodness of a partition is an important concern in designing community detection methods. In general, a *quality function* is used as a quantitative criterion that assigns a number to each partition of a graph to rank partitions based on their score [11]. Notably, *modularity* is the most popular quality function, where achieving high modularity translates to a better partition of the graph, thus better community structure (for further discussion on the methodology please refer to [12]). Given that, AutoBotCatcher exploits a modularity-based *Louvain* method [9], that is, an hierarchical greedy algorithm trying to improve modularity for community detection.¹⁰ Louvain method has many advantages that makes it a good choice for AutoBotCatcher, such as: it is faster and achieves higher modularity than other methods; and it is able to process large networks in a short time.¹¹

C. Blockchain

Blockchain relies on the concept of a distributed ledger maintained by a peer-to-peer network [13]. Novelty of the blockchain technology lies in its ability to achieve coordination and verification of individual activities carried out by different parties without a centralized authority or trusted third party, that allows decentralization of application execution with concerted and autonomous operations [14]. In blockchain, *transactions* transfer information (i.e., data packets) between peers. They have a unique identifier (transaction-id), input data, and are bundled into data chunks, referred to as *blocks*. Block generator peers of the blockchain broadcast blocks by exploiting public-key cryptography. Blocks are recorded in the blockchain with an exact order. Briefly, a block contains: a set of transactions; a timestamp; a reference to the preceding block that identifies the block's place in the blockchain; an authenticated data structure (e.g., a Merkle tree) to ensure block integrity.¹²

As a state transition machine, different participants in the blockchain have to achieve consensus on the latest state of the ledger in order to achieve coordination on the processes that they perform on the ledger. There are different methodologies to achieve consensus in the blockchain. For example, some blockchains use *Byzantine Fault Tolerant (BFT)* methodologies that depend on state replication between block generators; other use *Proof of Work (PoW)*, where block generators have to use their hardware resources and energy to generate a block by solving a cryptographic puzzle; while others use *Proof of Stake (PoS)*, where block generator selection depends on part of peers' wealth that they have voted as their stake in the

¹⁰Despite initially being designed for unweighted graphs, it can be easily adapted to weighted ones.

¹¹It took 12 minutes for a network containing 39 million vertices and 783 million edges [9] with AMD dual opteron 2.2k, 24 GB of RAM.

¹²Block structure varies in different blockchain protocols, here we list the most common elements.

⁹symantec.com/connect/blogs/hajime-worm-battles-mirai-control-internet-things

block generator selection process. Notably, BFT blockchains can maintain a relative high throughput. For example, BFT Tendermint consensus protocol [15] is able to process thousands of transactions per second. Generally, BFT blockchains can scale to dozens or few hundreds of block generators.

Blockchains can be classified into two groups as *public* and *permissioned* according to their way of regulating peers' participation in blockchain operations. Particularly, in public blockchains, any peer can read and write to the blockchain, meaning that anyone can participate in the consensus process. Whereas, in permissioned blockchains, only a set of previously identified peers can write to the blockchain and participate in the consensus and access to the stored transactions.

In AutoBotCatcher, we exploit a BFT blockchain in permissioned setting for processing network traffic flow data for botnet community detection. Permissioned BFT blockchain is adequate for AutoBotCatcher setting, since its design is based on assigning the block generator role to relatively small set of actors such as device vendors, internet service providers etc. Moreover, AutoBotCatcher exploits states of a BFT blockchain in performing dynamic botnet community detection process, as discussed in the following section. Finally, in permissioned blockchain setting stored network data kept confidential from unknown parties as only a set of pre-identified block generators can access to network data stored in the blockchain.

IV. THE BLOCKCHAIN PARADIGM

We devote this section to explain how blockchain is used in AutoBotCatcher.

Transactions. To detect botnets, AutoBotCatcher employs community detection analysis on the mutual contacts graph. In particular, to generate mutual contacts graphs, AutoBotCatcher needs to collect and analyze meta-data information about network traffic flow of IoT devices (i.e., IP addresses). To this end, we exploit a permissioned blockchain as a shared data store to audit meta-data, which are thus modelled as blockchain transactions sent by agents to the blockchain's *transaction pool*, a shared data store hosted by all block generators, that holds unprocessed transactions. Particularly, each agent is connected to one block generator to send transactions, where block generator that receives a new transaction disseminates new transactions to other block generators.

Meta-data are encoded into *network-data transaction (NT)*, formally defined as follows:

Definition 1: Network-data Transaction (NT). Let *NetFlow* be the network traffic flow defined as $NetFlow = (IP_{src}, IP_{dest})$, where IP_{src} and IP_{dest} are the source's and destination's IP addresses, respectively. Let $Device_{addr}$ be the unique public key of the agent that sends the transaction, and let $Tx - Pool_{addr}$ be the public key of the transaction pool that agent is connected to send the transaction. A network-data transaction is a tuple: $NT = \langle Device_{addr}, IP_{src}, IP_{dest}, Tx - Pool_{addr} \rangle$

Blocks. Transactions are bundled into blocks that are generated by block generators.¹³ In what follows, we symbolize a block as b_m , where m represents the block number. Given that, the block structure is as follows:

$$b_m = \{NT_x, \dots, NT_y\} \quad (1)$$

Where, NT represents a network-data transaction, and x and y are the transaction number. During the execution of AutoBotCatcher, one of the block generators is elected as a leader to generate a block. Block generation interval, symbolized as τ , represents the amount of time between consecutive block generations. Upon generation of a block, at least two-thirds of block generators should send acknowledgements to the block generator regarding their approval on the block.¹⁴ Once acknowledgements have been obtained and τ is expired, block generator of the next block generates the new block.

Rounds. In AutoBotCatcher, P2P botnet analysis is periodically performed upon generation of a set of blocks. We refer to such periods as *rounds*. Each round takes certain amount of time, symbolized as Δ . Value of Δ depends on both the amount of time needed for block generators to perform botnet detection operations on the blocks (e.g., mutual contacts graph extraction, botnet community detection, etc.), network quality (e.g., network connection delays), and blockchain protocol (e.g., transaction throughput etc.). A round is symbolized as Γ_{Δ_t} , where Δ_t is a timestamp specifying the starting time of the round. When the round Γ_{Δ_t} ends, the next round $\Gamma_{\Delta_{t+1}}$ starts. In round Γ_{Δ_t} , botnet detection is performed on NTs sent during the previous round $\Gamma_{\Delta_{t-1}}$. More precisely, NTs sent from timestamp Δ_{t-1} to timestamp Δ_t (which corresponds to execution time of round $\Gamma_{\Delta_{t-1}}$) are processed in round Γ_{Δ_t} .

State. Fundamentally, blockchain consists of set of shared states and performs state transition. In general, the *state* notion can be used to represent financial balances of users (e.g., Bitcoin), or, in a broader setting, it can represent anything that can be modeled as result of the computer programs (i.e., on Ethereum blockchain [16]). In our setting, the state is a mapping between IoT devices' and other hosts' IP addresses (that are subject to botnet detection) and communities (each marked as botnet community or benign community). In AutoBotCatcher, similar to the Ethereum protocol [16], the state is not stored on the blockchain, rather it is maintained on an efficient Merkle tree implementation,¹⁵ such as Patricia Merkle Trees.¹⁶ Merkle trees are hash based data structures, where each node is the hash of its children or hash of the data, if the node is a leaf. Main benefits of using Merkle trees to store states are: they are immutable data structures, thus we are able to secure entire system states with cryptographic dependences;

¹³Number of transactions in a block is regulated according to the block size and transaction size settings of the blockchain protocol.

¹⁴If the block generator does not get approval from at least two-thirds of block generators, that block will not be added to the blockchain, and a new block generator will be selected to generate a block.

¹⁵Exploiting Merkle trees in our blockchain setting requires a simple state database backend.

¹⁶github.com/ethereum/wiki/wiki/Patricia-Tree

and, they allow the blockchain protocol to trivially revert to any old state by simply altering the root hash [16].

In AutoBotCatcher, the state of the blockchain on timestamp Δ_t , symbolized as σ_{Δ_t} , consists of: the snapshot of the latest version of the mutual contacts graph G_{Δ_t} , that is generated from the network-data transactions; the set of all communities and IP addresses of hosts associated with that communities, extracted from the mutual contacts graph by block validators, symbolized as $CommSet_{\Delta_t}$; and, all blocks of the blockchain. Given that, the state of the blockchain on timestamp Δ_t is represented as follows:

$$\sigma_{\Delta_t} = (G_{\Delta_t}, CommSet_{\Delta_t}, [B_0, \dots, B_m]) \quad (2)$$

In Figure 1, we present an example of rounds and states, where each round includes generation of three blocks.

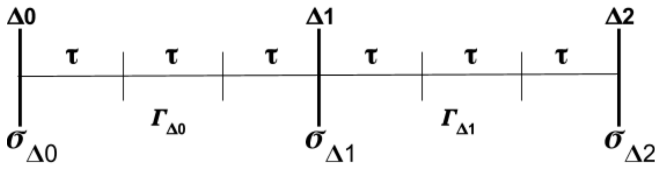


Figure 1: Round and state relation

State transitions. In blockchain, state transition refers to achieving a new valid state after executing a set of transactions on the previous state. AutoBotCatcher uses the community set and snapshot of the mutual contacts graph of the previous blockchain state to dynamically perform botnet community detection. Therefore, in AutoBotCatcher, a state transition occurs upon finishing a round of botnet detection operations (see Section V for more details). Given that, for every round Γ defined above, one state change occurs upon execution of the set of blocks. Particularly, upon execution of round Γ_{Δ_t} , the state changes from σ_{Δ_t} to $\sigma_{\Delta_{t+1}}$ which includes a new set of processed blocks. We define the state transition as a function of rounds, which takes previous state and new blocks as input, as follows:

$$\sigma_{\Delta_{t+1}} = \Gamma(\sigma_{\Delta_t}, [B_m, \dots, B_{m+z}]) \quad (3)$$

Where σ_{Δ_t} is the previous state (see Equation 2), and $[B_m, \dots, B_{m+z}]$ is the set of new blocks, where m and z represent block number (see Equation 1).

Consensus. Blockchain protocol adopted by AutoBotCatcher uses Byzantine Fault Tolerant (BFT) methodology to achieve consensus in a permissioned setting. In BFT blockchain, in order to achieve consensus, at least two-thirds of the pre-identified block generators have to agree on the latest state proposed by one of the block generators. In our setting, this translates to agree on: the same set of blocks processed; same mutual contacts graph; and same community mapping of IoT devices.

V. SYSTEM ARCHITECTURE

AutoBotCatcher exploits a permissioned BFT blockchain as a backend module to perform dynamic and collaborative botnet detection on large scale networks. An overview of the execution flow of AutoBotCatcher is presented in Figure 2 and discussed in what follows.

Network data pre-processing. This task is executed by agents for monitoring network traffic flow of IoT devices and taking actions according to that. In performing such operations, agents maintain two types of IP address lists, namely *blacklist* and *whitelist*. Blacklists contain IP addresses that have been previously detected as part of a botnet. On the other hand, whitelists contain predefined and trusted IP addresses,¹⁷ such as IP addresses of the vendor's servers, for all IoT devices in their subnet.

More precisely, an agent constantly sniffs network traffic flows of IoT devices in its subnet; the agent does not take any action for network traffic of an IoT device with a whitelisted IP address; for all other network traffic flows, the agent forms network flow transactions (NTs) (as given in Definition 1), if a network flow is with a blacklisted IP address, then agent quarantines that IoT device either by forcing it to shut down or by cutting all of its network connections.

Blockchain operations. This task is executed by block generators for the generation and relay of blocks that include NTs. To perform these operations in a round Γ_{Δ_t} , first, one block generator is selected to generate one or more blocks.¹⁸ That block generator takes the subset of NTs from the blockchain transaction pool, it forms a block and broadcasts the block to all block generators. We recall that each block has to receive approval from at least two-thirds of the block generators in order to be valid, and to achieve consensus on its validity in a BFT blockchain setting.

Graph extraction. This task is executed by block generators, aiming at elaborating the network traffic flow data, that is, the NTs processed in a round Γ , to generate/update the mutual contacts graph G . Particularly in round Γ_{Δ_t} , block generators create the mutual contacts matrix $MCM_{\Delta_{t+1}}$, representing $G_{\Delta_{t+1}}$ that results from the state $\sigma_{\Delta_{t+1}}$, by updating MCM_{Δ_t} , i.e., G_{Δ_t} of state σ_{Δ_t} , with the NTs in blocks sent from Δ_{t-1} to Δ_t .

Operations to transit from MCM_{Δ_t} to $MCM_{\Delta_{t+1}}$ are: *updating weights of edges* between vertices that communicated in round Γ_{Δ_t} ; *adding vertices and edges*, if new IoT devices connect to a device network, or a new host IP address communicates with an IoT device; *removing vertices and edges*, if some IoT devices or hosts do not exist anymore.

Community detection. This task, executed by block generators, performs dynamic community discovery (DCD) on the mutual contacts graph G .

¹⁷We assume that hosts corresponding to IP addresses in whitelists are secure, and do not pose any threat to IoT devices.

¹⁸For the sake of brevity, we do not detail the election process, but any leader election process performed in a typical distributed system is suitable for our setting.

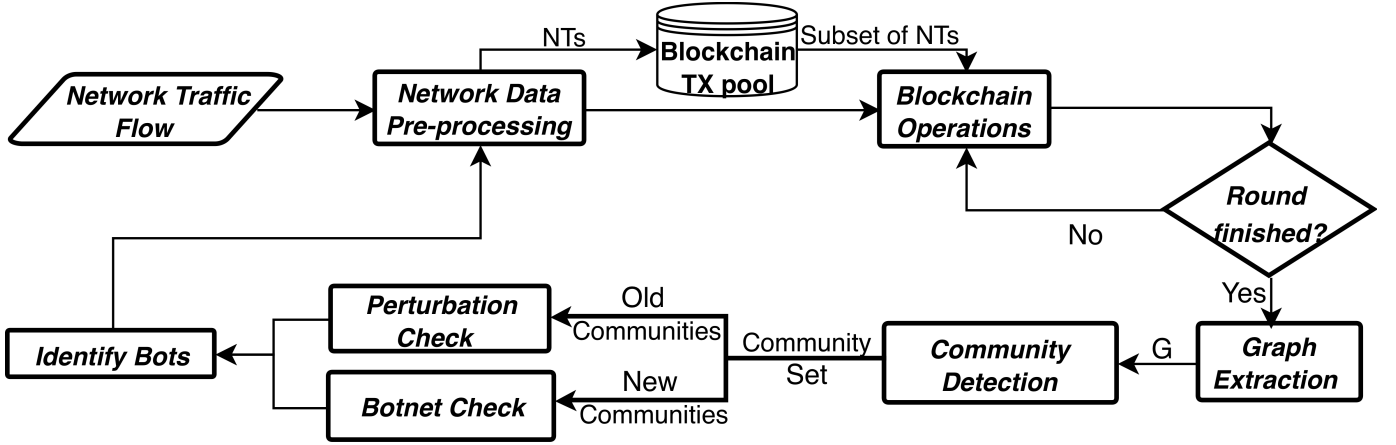


Figure 2: AutoBotCatcher system flow

AutoBotCatcher performs dynamic community detection with Louvain method on snapshots of the mutual contacts graphs from consecutive states of the permissioned BFT blockchain. Yet, as presented in [17], even small changes in consecutive network snapshots may cause Louvain method to generate two alike community structures, which would eventually cause AutoBotCatcher to lose track of the communities and thus degrade its execution. Therefore, to have a more stable community structure between timestamps, AutoBotCatcher initializes the Louvain algorithm for community detection at timestamp Δ_{t+1} with the communities found in Δ_t as proposed by Aynaud *et al.* in [17]. More precisely, in round Γ_{Δ_t} , AutoBotCatcher performs community detection on newly extracted mutual contacts graph $G_{\Delta_{t+1}}$, by feeding community set of IoT devices, $CommSet_{\Delta_t}$ from the last state δ_{Δ_t} to Louvain method.

Perturbation check. This task is performed by block generators to check updates (i.e., IP address additions and removals) on communities that already exist in the previous timestamp.¹⁹ Particularly, for botnet communities, IP addresses of new bots are inserted to the list called *additions to blacklist*, whereas IP addresses of bots that are no longer part of a botnet are inserted to the list called *removals from blacklist*. Upon block generators achieve consensus on the new state, block generators share those lists with agents (see task bot identifier).

Botnet check. Executed by block generators, this task is responsible for classifying new communities as either botnet or benign communities. Botnet detection methodology used by AutoBotCatcher is based on two observations: 1) bots connect with each other for command exchange, thus they have more mutual contacts than benign communities [8]; 2) botmasters or attack targets communicate with many nodes, referred to as pivotal nodes [18], so that they have very high number of mutual contacts. Therefore, according to the first observation, AutoBotCatcher calculates the average number

of mutual contacts per IP address for all communities. If the result is higher than a pre-defined mutual contacts threshold θ , that community is marked as *candidate botnet* community. Secondly, AutoBotCatcher checks for pivotal nodes in candidate botnet communities. Particularly, for all IP addresses in a candidate botnet community, AutoBotCatcher sums their rows in the mutual contacts matrix (MCM). If one or more pivotal nodes exists in the candidate botnet community, it is marked as botnet community by the block generator. Upon block generators achieve consensus on the new state, block generators share IP addresses in botnet communities with next task for bot blacklisting.

Identify bots. The last task is responsible for updating the bot blacklists of agents, and it is executed by block generators after a state change. It updates the blacklists of all IoT devices with the help of smart contract transactions for addition and deletion of IP addresses on agent's local blacklists.²⁰

VI. RELATED WORK

In recent years, vast amounts of work has been devoted to P2P botnet detection. In general, botnet detection methodologies can be categorized into two groups: host-based and network-based approaches. Host-based approaches require the monitoring of all hosts, which is impractical for the IoT domain. Therefore, we focus on network-based approaches, which in turn can be classified into two groups:

- *Network traffic signature based approaches.* Literature offers many works that classify hosts based on their network traffic behaviour. In general, these approaches exploit supervised/unsupervised machine learning techniques to identify whether hosts are benign or malicious [19]–[27]. However, in a dynamic network, botmasters can randomize botnet traffic by changing communication frequency, packet sizes, etc. As such, network traffic signatures learned by machine learning approaches may not be robust enough to identify bots [10], which would eventually make such approaches ineffective.

¹⁹If a community changes more than a threshold ϕ , calculated as the ratio of total changes to number of edges and vertices, that community will be considered as a new community, and next task will be executed on it.

²⁰As it is not the main focus of our work, we do not detail how this mechanism works, such as transaction structure, which is left as future work.

Moreover, some of the proposed approaches, such as [19], [26], rely on deep packet inspection techniques (DPI) to analyze network packet contents. However, these checks can be bypassed through encryption of C&C channels. Given that, we conclude that network traffic signature based approaches are not suitable for dynamic and evolving IoT environments.

– *Group and community behavior based approaches.* Some works use group and community behaviour analysis for botnet detection [7], [8], [10]. As an example, similarly to us, [7] exploits mutual contacts extracted from network traffic flow of hosts, in order to identify bots in a P2P network. Whereas [10] performs group level behavior analysis on network traffic flow with Support Vector Machine (SVM). However, these approaches are able to detect only previously known bot types. Therefore, they are not suitable for IoT, where new botnets emerge frequently [2]. Differently, PeerHunter [8] exploits Louvain method to perform network flow level community behaviour analysis on mutual contacts graph, without relying on previously known bot types. Yet, PeerHunter performs static botnet detection on the collected network traffic flow data, which is inadequate for a dynamic and evolving IoT environment that requires dynamic botnet detection.

AutoBotCatcher differs from the previously discussed approaches in several ways. First, unlike DPI based approaches, to perform botnet detection AutoBotCatcher requires to trace only high level meta-data about network flow traffic (i.e., source and destination addresses), as such it is effective against encrypted C&C channels. Second, unlike network traffic signature based approaches, even if botmasters randomize network traffic by changing packet sizes, communication frequency etc., botnet community structures do not alter, since the same set of commands has to be shared with the same set of bots. Third, unlike other group and community behaviour based approaches, AutoBotCatcher performs dynamic community detection by exploiting blockchain, so it is able to detect emerging and unknown botnets, and to take preventive measures against them.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce a blockchain based dynamic P2P botnet detection and prevention mechanism for IoT, referred to as AutoBotCatcher, that performs community detection on network flows of IoT devices. In AutoBotCatcher, IoT gateway devices become peers of a BFT blockchain, where device vendors and/or security regulators take the block generator role and participate in the consensus process. Blockchain is exploited to perform dynamic network based botnet community detection on snapshots of the mutual contact graph of IoT devices.

Future work includes: implementing AutoBotCatcher by leveraging on BFT based Hyperledger blockchain²¹ and exploiting its smart contracts to generate mutual contacts graphs and to manage state changes; integrating secure multi-party

computation platforms, such as Enigma [28], in order to protect privacy of the collaborating parties, while ensuring correct botnet detection; testing AutoBotCatcher with several botnet examples; designing and implementing a trust management module between agents and block generators, in order to make AutoBotCatcher resilient against insider threats. We also plan to extend the current botnet detection model with methods making use of statistical network traffic features.

REFERENCES

- [1] B. Carminati *et al.*, “Enhancing user control on personal data usage in internet of things ecosystems,” in *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 291–298.
- [2] C. Kolias *et al.*, “Ddos in the iot: Mirai and other botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
- [3] G. Sagirlar *et al.*, “Decentralizing Privacy Enforcement for Internet of Things Smart Objects,” *ArXiv e-prints*, Apr. 2018.
- [4] G. Vormayr *et al.*, “Botnet communication patterns,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2768–2796, 2017.
- [5] M. Antonakakis *et al.*, “Understanding the mirai botnet,” in *USENIX Security Symposium*, 2017.
- [6] M. Karami *et al.*, “Stress testing the booters: Understanding and undermining the business of ddos services,” in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 1033–1043.
- [7] 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*. ACM, 2010, pp. 131–140.
- [8] D. Zhuang *et al.*, “Peerhunter: Detecting peer-to-peer botnets through community behavior analysis,” in *Dependable and Secure Computing, 2017 IEEE Conference on*. IEEE, 2017, pp. 493–500.
- [9] V. D. Blondel *et al.*, “Fast unfolding of communities in large networks,” *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.
- [10] Q. Yan *et al.*, “Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis,” in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 316–324.
- [11] S. Fortunato, “Community detection in graphs,” *Physics reports*, vol. 486, no. 3–5, pp. 75–174, 2010.
- [12] M. E. Newman *et al.*, “Finding and evaluating community structure in networks,” *Physical review E*, vol. 69, no. 2, p. 026113, 2004.
- [13] F. Tschorsch *et al.*, “Bitcoin and beyond: A technical survey on decentralized digital currencies,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.
- [14] G. Sagirlar *et al.*, “Hybrid-IoT: Hybrid Blockchain Architecture for Internet of Things - PoW Sub-blockchains,” *ArXiv e-prints*, Apr. 2018.
- [15] E. Buchman, “Tendermint: Byzantine fault tolerance in the age of blockchains,” Ph.D. dissertation, 2016.
- [16] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, 2016.
- [17] T. Aynaud *et al.*, “Static community detection algorithms for evolving networks,” in *Modeling and optimization in mobile, ad hoc and wireless networks (WiOpt), 2010 proceedings of the 8th international symposium on*. IEEE, 2010, pp. 513–519.
- [18] J. Wang *et al.*, “Botnet detection based on anomaly and community detection,” *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, 2017.
- [19] G. Gu *et al.*, “Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection,” in *USENIX security symposium*, vol. 5, no. 2, 2008, pp. 139–154.
- [20] T.-F. Yen *et al.*, “Are your hosts trading or plotting? telling p2p file-sharing and bots apart,” in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. IEEE, 2010, pp. 241–252.
- [21] J. Zhang *et al.*, “Detecting stealthy p2p botnets using statistical traffic fingerprints,” in *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 2011, pp. 121–132.
- [22] S. Saad *et al.*, “Detecting p2p botnets through network behavior analysis and machine learning,” in *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. IEEE, 2011, pp. 174–180.

²¹hyperledger.org

- [23] B. Rahbarinia *et al.*, “Peerrush: Mining for unwanted p2p traffic,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 62–82.
- [24] J. Zhang *et al.*, “Building a scalable system for stealthy p2p-botnet detection,” *IEEE transactions on information forensics and security*, vol. 9, no. 1, pp. 27–38, 2014.
- [25] P. Narang *et al.*, “Peershark: detecting peer-to-peer botnets by tracking conversations,” in *Security and Privacy Workshops (SPW), 2014 IEEE*. IEEE, 2014, pp. 108–115.
- [26] S. Mizuno *et al.*, “Botdetector: A robust and scalable approach toward detecting malware-infected devices,” in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–7.
- [27] S.-C. Su *et al.*, “Detecting p2p botnet in software defined networks,” *Security and Communication Networks*, vol. 2018, 2018.
- [28] G. Zyskind *et al.*, “Enigma: Decentralized computation platform with guaranteed privacy,” *arXiv preprint arXiv:1506.03471*, 2015.