Detection of Bitcoin-Based Botnets Using a One-Class Classifier

Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Muttukrishnan Rajarajan

Detection of Bitcoin-based Botnets using a One-class Classifier *

Bruno Bogaz Zarpelão[1,3][0000−0001−9172−3578], Rodrigo Sanches Miani[2][0000−0002−8176−8040], and Muttukrishnan Rajarajan[3][0000−0001−5814−9922]

[1] Computer Science Department, State University of Londrina (UEL), Brazil
brunozarpelao@uel.br
[2] School of Computer Science, Federal University of Uberlândia (UFU), Brazil
miani@ufu.br
[3] School of Mathematics, Computer Science & Engineering, City, University of London, UK
r.muttukrishnan@city.ac.uk

Abstract. Botnets have been part of some of the most aggressive cyberattacks reported in recent years. To make them even harder to be detected and mitigated, attackers have built C&C (Command and Control) infrastructures on top of popular Internet services such as Skype and Bitcoin. In this work, we propose an approach to detect botnets with C&C infrastructures based on the Bitcoin network. First, transactions are grouped according to the users that issued them. Next, features are extracted for each group of transactions, aiming to identify whether they behave systematically, which is a typical bot characteristic. To analyse this data, we employ the OSVM (One-class Support Vector Machine) algorithm, which requires only samples from legitimate behaviour to build a classification model. Tests were performed in a controlled environment using the ZombieCoin botnet and real data from the Bitcoin blockchain. Results showed that the proposed approach can detect most of the bots with a low false positive rate in multiple scenarios.

Keywords: Anomaly Detection · Bitcoin · Blockchain · Botnet Detection · One-class Support Vector Machine

1 Introduction

Botnets are a significant threat to the Internet security. By compromising hundreds or thousands of Internet nodes, attackers can coordinate distributed large-scale attacks, which usually are very aggressive. Botnets are composed of three main elements: bots, botmasters, and the C&C (Command and Control) infrastructure. Bots are regular network nodes that attackers compromise to be under their control. Botmasters are malicious users who design the attacks and

* We would like to thank Dr. Syed Taha Ali for kindly sharing the ZombieCoin's source code with us, as well as Dr. Hassaan Khaliq for all his help. We would also like to thank the financial support from UKIERI (UK India Education Research Initiative).

send instructions to bots for attacks execution. The C&C infrastructure is the logical communication infrastructure that botmasters use to send instructions to bots. C&C architectures have evolved over the years. Attackers have built centralised infrastructures, based on protocols such as IRC (Internet Relay Chat) and HTTP (Hypertext Transfer Protocol), as well as distributed ones, which rely on peer-to-peer networks [1, 8].

Attackers have also been investing in C&C infrastructures that explore legitimate applications to keep their operations covered. In these scenarios, bots and botmasters operate as clients of widely used services such as instant messengers [14], online social networks [11] and blockchains [3]. Once these services are designed to share data with as many people as possible, attackers build their C&C structure on top of them. Therefore, they have resilient environments to rely on, which are kept up by the services' owners. Moreover, their actions are harder to detect because their traffic may be mistaken as the legitimate traffic from regular clients of these services. These botnets are also referred to as stealthy botnets [2, 14].

In this work, we propose an approach to detect Bitcoin-based botnets. In these botnets, botmasters send instructions piggybacked on Bitcoin transactions to the bots. These botnets are not observable in general traffic features such as packet size, volume of packets, and bit rate, because they use the same Bitcoin network protocols as regular users. Therefore, we assume that, in Bitcoin-based botnets, the systematic behaviour typically found in bots can be observed in transactions attributes such as their values, numbers of inputs, numbers of outputs, and addresses.

The proposed approach makes use of the One-class SVM (Support Vector Machine) algorithm and can be divided into two main steps. First, data about legitimate transactions are collected from the Bitcoin blockchain, which is publicly available, to create a classification model with OSVM. Then, transaction data are retrieved from network packets and compared to the model, allowing the classification of the traffic as legitimate or malicious. To evaluate the approach, we built a controlled environment using the ZombieCoin botnet [3]. The results showed that our approach could detect the most of the bots, keeping a low false positive rate.

To the best of our knowledge, no other work has proposed a systematic approach to detect Bitcoin-based botnets. The key contributions of our study are:

- an approach to detect Bitcoin-based botnets that analyses only transactions attributes. It is independent of traffic features such as packets size, volume of packets, and bit rate;
- design of the detection approach based on the OSVM algorithm, which, unlike supervised techniques, does not require samples of malicious observations to create a classification model.

The remaining of this paper is organised as follows. Section 2 presents the related work. In Section 3, we discuss the proposed model. Section 4 shows the evaluation. Finally, Section 5 draws the final conclusions.

2 Related Work

Researchers have proposed multiple approaches for botnet detection in recent years. They range from methods that do not aim at any specific type of botnet to approaches designed specifically to Web-based, P2P, or stealthy botnets.

In [13], Wang and Paschalidis proposed a detection method that does not have any specific botnet type as a goal. They assume that activities are more correlated among malicious nodes than among legitimate ones, and botmasters and attack targets present distinguishable network traffic because they communicate with many other nodes. Their approach starts by employing the large deviations principle to detect anomalies in IP flows. It also models interaction graphs from source/destination data in network packets and makes use again of the large deviations principle to detect anomalies in these graphs. The outputs of these two processes are analysed to find out the most active nodes, which are supposed to be botmasters or targets. Then, community detection techniques are used to identify bots from the interactions of the most active nodes.

Sakib and Huang [8] proposed an approach to detect HTTP-based botnets. The proposed solution applies three anomaly detection algorithms in two steps. In the first step, HTTP requests features are analysed using Chebyshev's Inequality, One-class SVM, and Nearest Neighbor based Local Outlier Factor to find out if the requests were generated by human actions or bots (legitimate or malicious). In the second step, Chebyshev's Inequality is used to classify the bot requests as malicious or legitimate. Hsu et al. [4] also analysed HTTP packets to detect HTTP-based botnets. However, unlike [8], they computed metrics based on the number of distinctly accessed servers and the payload size similarity. To detect the bots, they compare these metrics to thresholds.

Wang et al. [12] proposed the BotCluster, a botnet detector that inspects IP flows to detect P2P-based botnets. At first, it combines unidirectional IP flows to turn them into bidirectional flows. Next, the system filters out non-P2P flows using a whitelist and the flow loss response rate. Then, DBSCAN, an unsupervised clustering algorithm, is applied to separate the legitimate P2P flows from the malicious ones. The authors assumed that botnets behaviour presents high regularity, as well as actions from different bots in the same botnet are correlated, the both being observable in IP flows. Zhang et al. [14] proposed to detect stealthy P2P botnets following a similar sequence of two steps. First, they identified all hosts that were part of P2P communications. Next, they analysed the P2P flows to detect the malicious ones. To do so, they selected the more active P2P clients, considering them as bot candidates. Then, they investigated the hosts the bot candidates interacted to. According to the authors, bots usually communicate with the same hosts, which was observed to distinguish them from legitimate clients.

Albanese et al. [2] also proposed a system to detect stealthy botnets. This system explores the periodicity of traffic associated with data exfiltration using periodogram analysis. Additionally, the authors discussed techniques to find the best places in the network to deploy the solution, and the use of moving target defence to neutralise stealthy bots evasion movements. Venkatachalam and

Anitha [11] proposed a system to detect Stegobot, a stealthy botnet proposed by [6] that makes use of steganography to leak the target data in images shared on online social networks (OSN). The proposed system extracts features from OSN profiles regarding uploaded images, level of activity, and relationships with other profiles. Then, these profiles are classified into legitimate or malicious by supervised classification techniques such as SVM, k-Nearest Neighbour, Decision Tree, and Naive Bayes.

Unlike the previous work [12–14], in this work we do not rely on IP flow analysis. In Bitcoin-based botnets, once bots communicate with peers to disseminate transactions in the same way regular users do, there are no flow statistics that can disclose bot-related activities. Instead, we analyse transaction attributes such as the number of inputs and outputs, values of outputs, and rates of transactions to detect bot traffic. In this sense, our work follows a strategy similar to that used by [4, 8, 11], which also make use of information extracted from bots activities on a low-level basis. Nonetheless, it is important to highlight that, to the best of our knowledge, it is the first time a systematic approach has been proposed to detect Bitcoin-based botnets.

3 Proposed Approach

In this section, we present the proposed approach to detect Bitcoin-based botnets. We discuss first the approach rationale, and then the details on the two modules: Model Creation Module and Botnet Detection Module.

3.1 Approach Rationale

Different works on botnet detection assumed that bots present a more systematic behaviour than regular nodes. Depending on the work, this regularity was observed on the correlation between bots behaviour [12, 13], features extracted from HTTP packets [4, 8], the time the bots kept active [14], and the periodicity of traffic related to data exfiltration [2].

In this work, we explored the regularity of bots behaviour regarding the transactions attributes. Each transaction has different attributes such as number of inputs and outputs, addresses, and values. Fig. 1 illustrates a transaction. It contains an identifier, which is the hash of the transaction, and the inputs and outputs with their addresses and values. In the transaction illustrated, a user is transferring \$1.0 from its account holding the address "12cb...Tu3S" to the accounts with addresses "1Q2T...Jvm3" and "1bee...Vwq8", which will receive \$0.8 and \$0.2, respectively. We assumed that bots present a more systematic behaviour for these attributes because they are programmed, and so they may not follow the same behaviour as a human being making Bitcoin transfers.

To be able to compute the evolution of these attributes throughout multiple transactions, we grouped these transactions according to the users that issued them. The blockchain is composed of a set of transactions denoted as $T = \{t_1, \ldots, t_n\}$. Considering the set of users $U = \{u_1, \ldots, u_n\}$ that issued these

transactions, we assume that each user u_i has its set of transactions T_{u_i}, where $T_{u_i} \subset T$. Each transaction t is defined by the 3-tuple (ts, I, O), where ts denotes the transaction timestamp, I the set of inputs, and O the set of outputs.

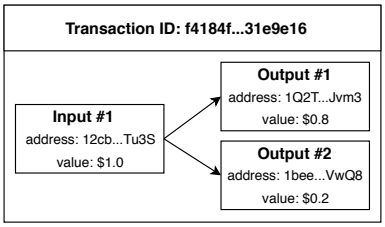

Fig. 1. Example of a transaction.

Having the set of transactions T_{u_i} for a user u_i, we can extract the features to distinguish botnet traffic from regular traffic. Algorithm 1 shows how the features are extracted. In this algorithm, $|X|$ denotes the amount of elements in the set X, e.g., $|t_1.I|$ represents the number of inputs in the transaction t_1.

Algorithm 1: Extracting a feature vector for a set of transactions

Input : List of transactions $T_u = \{t_1, t_2, \ldots, t_n\}$ for user u, and the timestamp $startTime$ referring to the oldest transaction of the dataset
Output: Feature vector f for user u

Function `extractFeatures`$(T = \{t_1, t_2, \ldots, t_n\}, startTime)$
  $f.inMedian \leftarrow$`median`$(|t_1.I|, |t_2.I|, \ldots, |t_i.I|)$;
  $f.inIQR \leftarrow$`iqr`$(|t_1.I|, |t_2.I|, \ldots, |t_i.I|)$;
  $f.outMedian \leftarrow$`median`$(|t_1.O|, |t_2.O|, \ldots, |t_i.O|)$;
  $f.outIQR \leftarrow$`iqr`$(|t_1.O|, |t_2.O|, \ldots, |t_i.O|)$;
  $f.lowVMedian \leftarrow$`median(lowestOutput`(t_1)`,...,lowestOutput`(t_i)`)`;
  $f.lowVIQR \leftarrow$`iqr(lowestOutput`(t_1)`,...,lowestOutput`(t_i)`)`;
  $f.iatMedian \leftarrow$`median`$(t_2.ts - t_1.ts, t_3.ts - t_2.ts, \ldots, t_i.ts - t_{i-1}.ts)$;
  $f.iatIQR \leftarrow$`iqr`$(t_2.ts - t_1.ts, t_3.ts - t_2.ts, \ldots, t_i.ts - t_{i-1}.ts)$;
  $f.txTime \leftarrow (t_i.ts - startTime) \div (|T_u|)$;
  $f.addressPerTx \leftarrow (|$`inputAddrs`$(T_u)|) \div (|T_u|)$;
  return f
end

Almost all the features extracted are based on two statistical measures: the median (denoted in Algorithm 1 as the function $median(x_1, \ldots, x_n)$), and the interquartile range (IQR, denoted in Algorithm 1 as $iqr(x_1, \ldots, x_n)$). The median is a measure of central tendency, while the IQR measures the data dispersion. The objective is to determine the central tendency and the dispersion of some

transaction attributes for a particular user, modelling its behaviour. All the extracted features are discussed next.

- The median ($inMedian$) and the IQR ($inIQR$) of the number of inputs, and the median ($outMedian$) and the IQR ($outIQR$) of the number of outputs of each transaction. Legitimate users can build transactions with various amounts of inputs and outputs depending on their needs, while bots, being programs, might present a more systematic behaviour.
- The median ($lowVMedian$) and the IQR ($lowVIQR$) of the lowest output value of each transaction. Bot transactions cannot involve high amounts of funds, because their operation must be as profitable as possible to the attackers. This way, it is expected that the lowest output value of bot transactions is usually smaller than the one found in legitimate transactions. Besides, unlike bots, legitimate users are expected to present more variability in these values. In Algorithm 1, $lowestOutput(t)$ denotes a function that take as input a transaction and returns its lowest output.
- The median ($iatMedian$) and the IQR ($iatIQR$) of the time intervals between two subsequent transactions. Bots are expected to present more periodicity than legitimate users, since they usually perform some automatic tasks.
- The relation ($txTime$) between the amount of transactions and the time elapsed. Bots may present a high level of interaction with botmasters, particularly when they are receiving instructions to launch attacks. This way, it is expected that they receive more transactions within a particular time interval than a legitimate user.
- The relation ($addressPerTx$) between the number of distinct input addresses and the number of transactions. The decision on how to use (or reuse) addresses is programmed in bots, while legitimate users can change it from one transaction to another. Therefore, the number of distinct addresses can vary from legitimate to bot users. In Algorithm 1, $inputAddrs(T)$ represents a function that takes a set of transactions as input and returns the set of distinct input addresses used in these transactions.

To classify the feature vectors as legitimate or malicious, we employed the OSVM technique. In supervised machine learning techniques, usually employed in botnet detection, a classification model is built from data instances representing the different classes the data can be classified into. For example, to detect botnets, samples from malicious and legitimate behaviour should be labelled and presented to the classifier, which would construct a model that could be used to classify future samples as legitimate or malicious. This can be problematic when the labelling process is labour-intensive and error-prone, or samples of some of the classes hardly occur [5].

A solution to address this issue is to use one-class classifiers. In these techniques, only samples of one class are presented to the classifier to build the classification model, e.g., samples of legitimate behaviour. Then, the upcoming samples are classified as belonging or not to this class. Therefore, the labelling

process is no longer necessary. The OSVM algorithm is a one-class classification technique that was successfully used in many domains [5]. This technique creates hyperplanes that work as boundaries around the region containing the training data. This way, if an instance is inside these boundaries, it is classified as belonging to the modelled class. Otherwise, it is classified as an anomaly. In our case, we provided data from legitimate Bitcoin users to create the model. Then, all data instances that are outside the boundaries set for this class are classified as bots.

3.2 Model Creation Module

The objective of the Model Creation Module, represented in Fig. 2, is to create a classification model from information available on the Bitcoin blockchain.

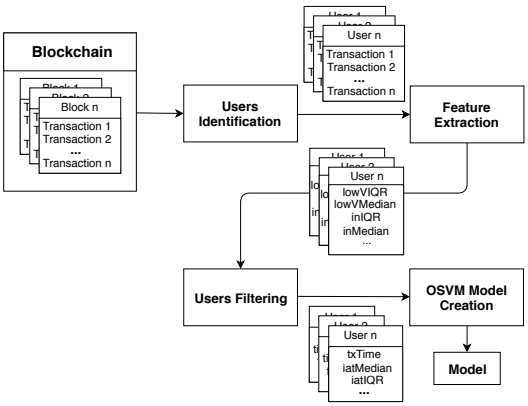

Fig. 2. Creation of classification model from Blockchain data.

The first step for the model creation is to retrieve a set of blocks from the blockchain. All the transactions from the retrieved blocks are extracted to a list denoted as $T = \{t_1, \ldots, t_n\}$. Next, the *Users Identification* step is performed. The objective of this step is to assign every transaction in T to a user u.

As already discussed in Section 3.1, a Bitcoin transaction t has a set of inputs and outputs. Inputs and outputs are associated with addresses, which are derived from the public keys of the users involved in the transaction. These addresses are pseudonyms, and a single user can have multiple addresses. When a user wants to send funds to someone, it signs the transference using its private key and informs the public keys of the fund destinations. It is recommended that users generate new keys for every transaction, avoiding that transactions are traced and the users identified [7]. Even so, it is possible to group Bitcoin transactions according to their source. We followed two strategies to do it:

- As many users do not generate new keys to make new transactions, it is possible to explore this fact to group them. Let t_1 and t_2 be two transactions, and $inputAddrs(\{t_1, \ldots, t_n\})$ a function that returns the set of input addresses for a given set of transactions. If $inputAddrs(\{t_1\}) \cap inputAddrs(\{t_2\}) \neq \emptyset$, then we assume that these two transactions were generated by the same user.
- The second strategy explores a heuristic based on multi-input transactions [7, 10]. Sometimes, a user does not have enough funds attached to a single address to complete a transaction. To avoid breaking this transaction into smaller ones, the user includes multiple inputs, each one associated to an address, in a single transaction. Therefore, it is possible to assume that all the input addresses of this kind of transaction belong to the same user.

After assigning the transactions to the users, the next step is to extract the feature vectors from them. To do so, all the transactions of each user are ordered by timestamp. Then, for a given user u_i, the transactions in T_{u_i} are processed according to Algorithm 2. The main idea of Algorithm 2 is to generate feature vectors as the transactions were being assigned one by one to the user. For example, let's suppose a user has six transactions $\{t_1, \ldots, t_6\}$ and $minTxs$ is 4. Then, three different feature vectors would be extracted: f_1 extracted from t_1, t_2, t_3 and t_4, f_2 extracted from t_1, t_2, t_3, t_4 and t_5, and f_3 extracted from t_1, t_2, t_3, t_4, t_5 and t_6. The function $extractFeatures(T, startTime)$ present in Algorithm 2 is defined in Algorithm 1.

Algorithm 2: Extracting feature vectors for one user

Input : List of transactions $T_u = \{t_1, t_2, \ldots, t_n\}$ for user u, the timestamp $startTime$ referring to the oldest transaction present in all blockchain collected data, and the minimum number of transactions per user $minTxs$

Output: List of feature vectors F_u for user u

```
for i ← minTxs to n do
    f ←extractFeatures({t_1, t_2, ..., t_i},startTime);
    add f to F_u;
end
```

The idea behind this approach is related to the way the Botnet Detection Module works, classifying users every time they have new transactions. If the Botnet Detection Module waits until it gathers a large number of transactions for a particular user to start analysing it, the bot detection might take a long time. Otherwise, if the Botnet Detection Module analyses this user every time a transaction is assigned to it, it might detect a bot quicker, after only a few transactions has been assigned to this user. Once the Model Creation Module is building a classification model for the Botnet Detection Module, it has to emulate the same process followed by the latter one.

After extracting the features, the next step is *Users Filtering*. Users with high dispersion for the number of inputs and outputs and the output values are more likely to be legitimate. Therefore, in this step, we filter this kind of user out. Only legitimate users that may be more similar to bots are included in the OSVM model, since they demand a more sophisticated technique to be differentiated from bots. The filtering process is detailed in Algorithm 3. Three thresholds, t_{in}, t_{out} and t_{low}, are set by computing the first quartile (denoted in Algorithm 3 as the function $firstQuartile(x_1, \ldots, x_n)$) of three features, namely $inIQR$, $outIQR$ and $lowVIQR$. All feature vectors in F are considered to compute these thresholds. Next, all feature vectors that hold values below the thresholds are included in a new list of feature vectors F'. In F', a feature f' is defined by the 5-tuple $(lowVMedian, iatMedian, iatIQR, txTime, addressPerTx)$.

Algorithm 3: Filtering out users with high dispersion

Input : List of feature vectors $F = \{f_1, f_2, \ldots, f_n\}$
Output: List of remaining feature vectors $F' = \{f_1, f_2, \ldots, f_m\}$, where $m \leq n$

$t_{in} \leftarrow$ `firstQuartile(`$\{f_1.inIQR, f_2.inIQR, \ldots, f_n.inIQR\}$`)` ;
$t_{out} \leftarrow$ `firstQuartile(`$\{f_1.outIQR, f_2.outIQR, \ldots, f_n.outIQR\}$`)` ;
$t_{low} \leftarrow$ `firstQuartile(`$\{f_1.lowVIQR, f_2.lowVIQR, \ldots, f_n.lowVIQR\}$`)` ;
foreach f *in* F **do**
    if $f.inIQR \leq t_{in} \wedge f.outIQR \leq t_{out} \wedge f.lowVIQR \leq t_{low}$ **then**
        add f *to* F';
    end
end

The last step performed by the Model Creation Module is *OSVM Model Creation*. In this step, firstly, all features for each feature vector are scaled according to their minimum and maximum values between a range from 0 to 1, as defined by Equation (1):

$$x_f[j] = \frac{x_f[j] - min(x[j])}{max(x[j]) - min(x[j])}, \forall f \in F', \forall j \in J \quad (1)$$

where f corresponds to a given feature vector, F' to all feature vectors extracted, j to a feature in the J feature space, and $x_f[j]$ to the value present in feature j for the feature vector f. The OSVM algorithm receives the normalised feature vectors as input and provide as output a model of the behaviour of the users represented by these feature vectors.

3.3 Botnet Detection Module

The Botnet Detection Module analyses network packets to detect the presence of bots in the monitored network. Therefore, in a real network, this module would be placed in the border between the monitored network and the Internet, being

able to analyse all Bitcoin packets exchanged between internal and external nodes. We assume that Bitcoin-based bots are implemented as SPV (Simplified Payment Verification) clients due to their low memory and traffic footprint. Unlike full Bitcoin clients, SPV clients do not receive all the transactions that other clients broadcast, and do not keep a copy of the entire blockchain. They rely on full nodes, which forward to SPV clients the transactions of their interest. An overview of this module is presented in Fig. 3.

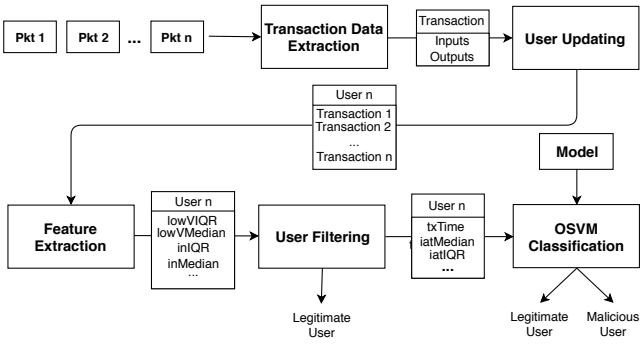

Fig. 3. Botnet detection using data extracted from transactions and the model created previously.

The first step of the Botnet Detection Module is to inspect Bitcoin packets, one by one, and extract transaction data from them. Then, this data is used to update a user profile. The users are determined according to the packets' IP addresses, since SPV nodes only receive transactions of their interest. A user u has a list of transactions $T_u = \{t_1, \ldots, t_n\}$, which were forwarded to the u's IP address. Every time a new t is added to T_u, u is classified again. To classify u, it is necessary to extract its features. The list T_u containing all the transactions that u received so far is passed as an argument to the function $extractFeatures(T, startTime)$ in Algorithm 1. This function returns a feature vector f_u.

With f_u, it is possible to perform an attempt of classification using the thresholds set in Algorithm 3. If the user cannot be classified based on those thresholds, the OSVM algorithm is used. Before applying the OSVM algorithm, the feature vector f_u is scaled following the normalisation step in Equation (1), except that this time, the $max(x[j])$ and $min(x[j])$ are the same as the ones used when the model was created. As it was done in the Model Creation Module, only the features $lowVMedian$, $iatMedian$, $iatIQR$, $txTime$, and $addressPerTx$ are used in the OSVM classification. Algorithm 4 shows how the classification process works. In this algorithm, the normalisation step is denoted as the function $normalise(x)$, and the OSVM classification algorithm is denoted as the function $OSVM(x)$.

Algorithm 4: Classifying user

Input : A feature vector f for user u
Output: Classification of u: "legitimate" or "malicious"

```
if f.inIQR > t_in ∨ f.outIQR > t_out ∨ f.lowVIQR > t_low then
    classification ← "legitimate";
else
    f' ← normalise(f);
    classification ← OSVM(f');
end
```

4 Evaluation

In this section, we describe the experiment performed to evaluate the proposed approach. First, some concepts of the ZombieCoin botnet [3] are presented. Next, the details of the experimental design are provided. Finally, the results are presented and discussed.

4.1 ZombieCoin

To evaluate our approach, we built an instance of the ZombieCoin botnet, which was proposed by Ali et al. [3]. This botnet makes use of the Bitcoin network to allow the botmaster to transmit instructions to the bots. Both the botmaster and the bot are developed with the BitcoinJ library as SPV clients. This botnet explores a function in Bitcoin transactions referred to as OP_RETURN. This function allows users to include up to 80 bytes of data into the transaction. It may be used, for example, to add textual information about the transaction like clients usually do in conventional banking transfers. Next, we present an outline of ZombieCoin operation:

- The botmaster has a key pair (public and private keys) that is used to protect its account and sign its transactions. When the botmaster is installed, it provides a command line interface to the user with a list of instructions that can be transmitted to the bots.
- The bot has the public key of the botmaster hardcoded. Using this key, it can request the botmaster transactions to its peers and authenticate them. Once the bot is installed, it can receive transactions with instructions from the botmaster, decode them and perform actions as requested.
- All the communication between bots and botmaster is based on standard Bitcoin protocol specification.

We implemented three commands in the ZombieCoin bot and botmaster: REGISTER, SYN FLOOD ATTACK, and UDP FLOOD ATTACK. When a bot receives the REGISTER command, it generates a file with a unique bot identifier, the current timestamp, and some information about the compromised host such as the number of processors, processor architecture, operating system,

and available memory. This file is uploaded to a Dropbox account belonging to the botmaster. Botmasters send this kind of command periodically to enumerate the active bots and have some control and detailed information about the compromised machines [3, 14]. When SYN FLOOD ATTACK or UDP FLOOD ATTACK commands are sent, the bot launches the respective DoS attack against the selected target.

4.2 Experiment Design

To emulate the malicious traffic, we built a network with six nodes. An instance of the ZombieCoin botmaster was deployed in an Amazon Web Service (AWS) host. To implement the bots, five Linux containers were created in a host at our laboratory, and each container hosted a ZombieCoin bot instance. Containers are lightweight virtual machines that emulate a machine with its own operating system, but share the host's operating system kernel with other applications. LXC[4] was used to create and manage the containers. The botmaster host had a public IPv4 address attributed by AWS. Private IPv4 addresses were assigned to the bots' hosts. The botmaster and the bots had access to the Internet. The software Wireshark[5] was installed in the machine hosting the bots' containers, being able to capture all the packets the bots transmitted and received.

Bots and botmaster were executed for nearly two hours, and the botmaster sent twelve commands to the bots. The objective was to reproduce a situation with an attacker actively using its botnet to launch DDoS attacks, while checking the status of its bots periodically. These commands will be detailed in the next section, when a particular scenario is discussed to show how the approach detected the malicious commands.

The legitimate behaviour was emulated with 239,495 transactions collected from blocks appended to the main Bitcoin blockchain between 17-06-2018 and 20-06-2018. They were grouped into users, composing a database of legitimate users.

Multiple experimental scenarios were set according to two parameters: the number of legitimate users used to create the model (l_{model}) and the number of legitimate users present in the botnet detection step ($l_{detection}$). The idea behind the first parameter was to observe if the performance proposed by the approach improves depending on the size of the sample of legitimate users used for the model creation. The second parameter was defined to analyse if different numbers of legitimate users could influence the task of distinguishing malicious and legitimate instances. Higher amounts of legitimate users increase the probability of having more diverse legitimate behaviour, which might confuse the botnet detector.

Five different values were assigned to l_{model}: 100, 200, 300, 400, and 500 users. $l_{detection}$ received the values 10, 20, 30, 40, and 50. The combination of these values resulted in twenty five experimental scenarios. Ten rounds were executed

[4] https://linuxcontainers.org/
[5] https://www.wireshark.org/

for each scenario, and, at each round, the legitimate users were randomly selected from our database. The five malicious users were included in all rounds for all scenarios.

The performance of the proposed approach was evaluated according to the following metrics [9]:

- **TPR**: $\frac{TP}{TP+FN}$ among all feature vectors extracted from malicious users, how many were correctly classified.
- **FPR**: $\frac{FP}{FP+TN}$ among all feature vectors extracted from legitimate users, how many were incorrectly classified as malicious.
- **AUC**: $\frac{1}{2}(TPR + (1 - FPR))$ combines TPR and FPR into a single metric, facilitating the comparison between different experimental rounds.

TP, TN, FP, and FN stand for true positives, true negatives, false positives, and false negatives, respectively. The RBF (Radial Basis Function) kernel was used for the OSVM algorithm. For all tests, $minTxs$ was set to 3 transactions.

4.3 Results and Discussion

Firstly, we searched for a common value for the OSVM hyperparameter ν that could allow the different scenarios to reach a good performance. For each scenario, the proposed approach was executed with ν ranging from 0.05 to 0.95 in steps of 0.05. The ν values that yield the best AUC for each scenario were computed, and Fig. 4 shows the histogram for these values. It is possible to observe that $\nu = 0.05$ is the most frequent value, which was assumed for the remaining tests.

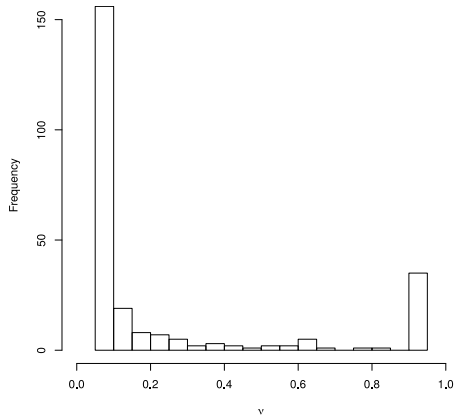

Fig. 4. Histogram for best values of ν considering different scenarios.

Another objective of the evaluation was to investigate how the number of legitimate users selected to create the model (denoted as l_{model}) and the number of legitimate users present in the detection step (denoted as $l_{detection}$) could

affect the results. Fig. 5 shows the performance of the approach taking into consideration the arithmetic mean of the AUC calculated throughout the ten rounds at each combination of l_{model} and $l_{detection}$. Although most of the scenarios presented good results, with the mean AUC above 0.8, the scenario with $l_{model} = 500$ users was clearly the best one. Different values for $l_{detection}$ did not affect the results in any situation. We assumed $l_{model} = 500$ users for the remaining tests.

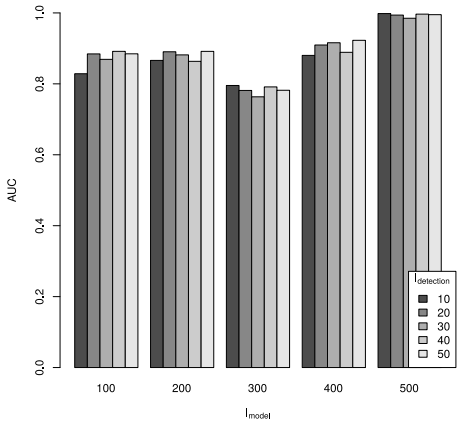

Fig. 5. AUC mean for different combinations of l_{model} and $l_{detection}$.

The next step is to analyse the results for the metrics AUC, TPR, and FPR considering $l_{model} = 500$, $\nu = 0.05$, and $l_{detection} = \{10, 20, 30, 40, 50\}$. The mean (1.00) and the standard deviation (0.00) computed for TPR indicate that this metric was equal to 1.00 for all the rounds. This means that even creating different classification models at each round and changing $l_{detection}$, the proposed approach was able to classify correctly all the feature vectors related to malicious users. The mean FPR was very low (0.01), and its standard deviation (0.02) shows that the values computed for this metric throughout the multiple rounds were not significantly higher than its mean. Once AUC is a function of TPR and FPR, its results were also good, with the mean = 0.99 and the standard deviation = 0.01.

Finally, we analysed a particular case to observe the characteristics of the true positives and the false positives. To carry out this analysis, we selected the case that presented the closest AUC to the mean for this metric, which was 0.99. The selected case was the sixth round of the experiment with $l_{model} = 500$, and $l_{detection} = 40$. For this case, we had 54 true positives, 248 true negatives, 1 false positive, and 0 false negatives. Table 1 presents the classification of the feature vectors generated from the commands received by the malicious users.

As soon as the proposed approach started analysing the transactions, it detected the bots. $minTxs$ was set to 3, so the approach waited for three transac-

Table 1. Malicious commands sent and their detection.

Elapsed time (min)	Command	User 1	User 2	User 3	User 4	User 5
0	REGISTER	NA	NA	NA	NA	NA
7	REGISTER	NA	NA	NA	NA	NA
14	SYN FLOOD ATTACK	TP	TP	TP	TP	TP
22	REGISTER	TP	TP	TP	TP	TP
35	SYN FLOOD ATTACK	TP	TP	TP	TP	TP
37	REGISTER	TP	TP	TP	TP	TP
52	REGISTER	TP	TP	TP	TP	TP
54	UDP FLOOD ATTACK	TP	TP	TP	TP	TP
64	UDP FLOOD ATTACK	TP	TP	TP	TP	TP
67	REGISTER	TP	TP	TP	TP	TP
82	REGISTER	TP	TP	TP	TP	TP
97	REGISTER	TP	TP	TP	TP	TP

tions of a user to begin analysing it. In our tests, the botmaster sent the third transaction 14 minutes after the beginning of the experiment. Table 1 shows that the two first transactions were not analysed ("NA") and, after that, all the analysed transactions sent for the bots were classified as malicious ("TP"). The only false positive was raised to a user that presented a very low dispersion for the number of inputs, outputs, and transferred value. Besides, its number of outputs was unusually high and its transfer value uncommonly low, when compared to other normal users.

Overall, the approach presented a high predictive performance. We evaluated different scenarios with multiple numbers of legitimate users, and the results for true positives and false positives were good for most of them. The scenarios that included more users (500) in the model creation step were the ones with the best results. In these scenarios, all malicious feature vectors were correctly classified as so in all rounds, and there were only a few false positives. Still, in all these scenarios with 500 users for model creation, the proposed approach was able to detect the bots as soon as it gathered the minimum number of transactions that allowed an analysis. This means that, for these cases, the proposed approach detected the bots right after the botmaster sent the third transaction in a row, only 14 minutes after the experiment had started.

5 Conclusion

Botnets have been the protagonists of severe attacks on the Internet. As attackers started building their C&C infrastructures on top of widely-used services, they became harder to be detected and mitigated. In this paper, we proposed an approach to detect Bitcoin-based botnets. The approach is based on the OSVM classifier, which requires only legitimate samples to build the classification model. To detect the bots, the proposed approach extracts information from different transactions belonging to the same user, aiming to identify whether the traffic belongs to a bot. Tests were conducted using the ZombieCoin botnet and real

transaction data from Bitcoin blockchain. The results demonstrated a high predictive performance, with high true positive rates and low false positive rates in several scenarios. The study of a particular case showed that the proposed approach detected the bots after the botmaster had sent only three commands. As future work, we intend to extend the proposed approach to detect botnets based on other blockchain applications such as Ethereum.

References

1. Acarali, D., Rajarajan, M., Komninos, N., Herwono, I.: Survey of approaches and features for the identification of HTTP-based botnet traffic. Journal of Network and Computer Applications **76**, 1 – 15 (2016)
2. Albanese, M., Jajodia, S., Venkatesan, S.: Defending from Stealthy Botnets Using Moving Target Defenses. IEEE Security Privacy **16**(1), 92–97 (2018)
3. Ali, S.T., McCorry, P., Lee, P.H.J., Hao, F.: ZombieCoin 2.0: managing next-generation botnets using Bitcoin. International Journal of Information Security **17**(4), 411–422 (Aug 2018)
4. Hsu, F.H., Ou, C.W., Hwang, Y.L., Chang, Y.C., Lin, P.C.: Detecting Web-Based Botnets Using Bot Communication Traffic Features. Security and Communication Networks **2017** (2017)
5. Khan, S.S., Madden, M.G.: A survey of recent trends in one class classification. In: Irish Conference on Artificial Intelligence and Cognitive Science. pp. 188–197. Springer (2009)
6. Nagaraja, S., Houmansadr, A., Piyawongwisal, P., Singh, V., Agarwal, P., Borisov, N.: Stegobot: A Covert Social Network Botnet. In: Filler, T., Pevný, T., Craver, S., Ker, A. (eds.) Information Hiding. pp. 299–313. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
7. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
8. Sakib, M.N., Huang, C.: Using anomaly detection based techniques to detect HTTP-based botnet C&C traffic. In: 2016 IEEE International Conference on Communications (ICC). pp. 1–6 (May 2016)
9. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Information Processing & Management **45**(4), 427–437 (2009)
10. Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: Extracting Intelligence from the Bitcoin Network. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security. pp. 457–468. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
11. Venkatachalam, N., Anitha, R.: A multi-feature approach to detect Stegobot: a covert multimedia social network botnet. Multimedia Tools and Applications **76**(4), 6079–6096 (Feb 2017)
12. Wang, C.Y., Ou, C.L., Zhang, Y.E., Cho, F.M., Chen, P.H., Chang, J.B., Shieh, C.K.: BotCluster: A session-based P2P botnet clustering system on NetFlow. Computer Networks **145**, 175 – 189 (2018)
13. Wang, J., Paschalidis, I.C.: Botnet Detection Based on Anomaly and Community Detection. IEEE Transactions on Control of Network Systems **4**(2), 392–404 (June 2017)
14. Zhang, J., Perdisci, R., Lee, W., Luo, X., Sarfraz, U.: Building a Scalable System for Stealthy P2P-Botnet Detection. IEEE Transactions on Information Forensics and Security **9**(1), 27–38 (2014)