Agarwal *et al. Appl Netw Sci* (2021) 6:9
https://doi.org/10.1007/s41109-020-00338-3

Applied Network Science

RESEARCH Open Access

Detecting malicious accounts in permissionless blockchains using temporal graph properties

Rachit Agarwal*, Shikhar Barve and Sandeep Kumar Shukla

*Correspondence:
rachitag@cse.iitk.ac.in
Computer Science Department, IIT Kanpur, Kanpur 208016, India

Abstract

Directed Graph based models of a blockchain that capture accounts as nodes and transactions as edges, evolve over time. This temporal nature of a blockchain model enables us to understand the behavior (malicious or benign) of the accounts. Predictive classification of accounts as malicious or benign could help users of the permissionless blockchain platforms to operate in a secure manner. Motivated by this, we introduce temporal features such as burst and attractiveness on top of several already used graph properties such as the node degree and clustering coefficient. Using identified features, we train various Machine Learning (ML) models and identify the algorithm that performs the best in detecting malicious accounts. We then study the behavior of the accounts over different temporal granularities of the dataset before assigning them malicious tags. For the Ethereum blockchain, we identify that for the entire dataset—the ExtraTreesClassifier performs the best among supervised ML algorithms. On the other hand, using cosine similarity on top of the results provided by unsupervised ML algorithms such as K-Means on the entire dataset, we were able to detect 554 more suspicious accounts. Further, using behavior change analysis for accounts, we identify 814 unique suspicious accounts across different temporal granularities.

Keywords: Blockchain, Machine Learning, Temporal graphs, Behavior analysis, Ethereum, Suspect identification

Introduction

A blockchain can be modeled as an ever-growing, large directed temporal network with more and more industries starting to adopt it for their businesses. In permissionless blockchains, interactions (also called as transactions) happen between different types of accounts. In Ethereum mainnet public blockchain, these accounts can be either Externally Owned Accounts (*EOA*) or Smart Contracts (*SC*). Here, transactions from an EOA (called as an *external transaction*) are recorded on the blockchain ledger whereas transactions from an SC (called as an *internal transaction*) are not recorded on the ledger. Note that in Ethereum, SCs can issue external transactions by executing external calls also which would be recorded in the blockchain ledger.

With actual money involved in most of the permissionless blockchains, an account must be able to perform secure transactions. Recently, many security threats to various blockchain platforms have been identified (Bryk 2018). For some identified vulnerabilities, counter-measures have been implemented. We do not delve into surveying all the security threats. In Chen et al. (2020), the authors survey security flaws that exist in the Ethereum blockchain. In many of the security vulnerabilities identified in Ethereum blockchain, hackers target other accounts by either hacking SCs or implementing malicious SCs for cyber-crimes such as ransomware, scams, phishing, and hacking of exchanges or wallets (Chainalysis 2019).

With an ever-increasing growth and adoption of the blockchain technology by the industry and the crypto-currency markets, permissionless blockchains are at the epicenter of increased security vulnerabilities and attacks. Our motivation for this work is based on the fact that there is limited work on learning the behaviors of the accounts in permissionless blockchains which are malicious and potentially victimize other accounts in the future. We define malicious accounts as those accounts that have been founded to be involved in illegal activities such as phishing, hacking, scams, and Ponzi schemes. Benign accounts as those that perform *only* legitimate transactions. Although, these illegal activities have different features, in this paper we assume that the transaction behavior and features are consistent and are applicable to all types of malicious activities. The results seem to bear out such assumption. In short, in this paper, we aim to identify malicious accounts so that the potential victims and blockchains can deploy counter-measures. In this paper, henceforth, the term blockchain is used to represent a permissionless blockchain. The techniques proposed in related studies classify accounts as malicious using either Machine Learning (ML) algorithms or motif-based (basic building block/subgraphs of a network) methods. Nonetheless, the features used by the available techniques are: (*a*) limited and not learned from the previous attacks on blockchains, and (*b*) extracted from the aggregated snapshot of time-dependent transaction graphs that do not consider the temporal evolution of the graphs.

The temporal aspects attached to the features are essential in understanding the actual behavior of an account before we can classify it as malicious. For example, inDegree and outDegree features are time-variant and should be considered a time series. Nonetheless, it has been proven that the aggregated node degree distribution for accounts follows a power-law in blockchains such as Ethereum (Chen et al. 2018a). Here, questions that we ask are: *does such behavior exist in all accounts? Is there a burst of degree for certain accounts at certain instances and can the existence of such bursts be used to identify malicious activity?* To answer these questions, we first identify the existence of bursts. Then, we introduce features such as *temporal burst*, *degree burst*, *balance burst*, and *gasPrice burst* to study the effect of bursts.

The fat-tailed nature of power-law degree distribution also gives rise to neighborhood-based fitness preferential attachment in blockchains (Aspembitova et al. 2019). In Aspembitova et al. (2019) authors defined fitness as "the ability of a node to attract new connections" and showed that the accounts that have high fitness sometimes are short-lived and indulge mostly in malicious activities while when they are long-lived they represent large organizations. Here, the authors define the fitness factor considering one previous time instance interactions. As it does not consider a temporal window, one

drawback of the method lies in its ability to correctly classify malicious transactions that appear at an interval of 2 time units or more. Inspired by this, we define a neighborhood-based feature called *attractiveness* that takes into account a temporal window of size θ_a where $(0 < \theta_a < T_{DS})$ and T_{DS} is the duration for which we collect the dataset (DS). Our attractiveness measure takes into account the stability of directed transactions that happened between two accounts in the past. Intuitively, a malicious account will have high attractiveness as it will tend to transact with new accounts while benign accounts will have high neighborhood stability or low attractiveness.

As the behavior of an account can change from malicious to benign or from benign to malicious over time, there is a need for continuous monitoring and analysis of the real-time transactions given the history of transactions performed by an account. We, thus, study the evolution of malicious behavior over different timescales by creating sub datasets and then answer the question: *would a certain account show malicious behavior in the future?* Towards this, we first apply different ML algorithms and identify the most suitable unsupervised ML algorithm for the entire dataset that is able to cluster accounts most accurately. Then we apply the identified algorithm to different sub datasets within a temporal scale to capture the behavior changes.

In summary, following are our main contributions:

- *Feature engineering* We identify *feature vector* for identifying malicious accounts based on *previous attacks on blockchains* and perform time series analysis. As new features, we propose *temporal burst*, *degree burst*, *balance burst*, *gasPrice burst*, and *attractiveness*.
- *Comparative analysis* We perform a *comparative study* with techniques proposed in the related studies and identify the best possible *supervised and unsupervised ML algorithm* with related hyperparameters when we use Ethereum transaction data.
- *Results* Our results demonstrate that for the supervised case, *ExtraTreesClassifier* performs the best with respect to balanced accuracy for the entire dataset while for the unsupervised case, we are able to identify 554 more suspect accounts using *K-Means Clustering*. Analysis of the behavioral changes reveal 814 suspects across different temporal granularities.

The rest of the paper is organized as follows. In Sect. 2, we present background and the state of the art techniques for identifying malicious accounts and compare them. In Sects. 3 and 4, we present a detailed description of our methodology and the feature vector, respectively.This is followed by an in-depth evaluation along with the results in Sect. 5. We finally conclude in Sect. 6 providing details on prospective future work. Further, in Abbreviation section we provide a glossary of the acronyms used in the paper.

Background and related work

There are two types of blockchain technologies, permissionless and permissioned. The major difference between the two technologies is that in a permissioned blockchain prior access approval is needed for performing any action on the blockchain while in permissionless blockchain anyone can perform actions on the blockchain without any approval. Further, there is no way to censor anyone from permissionless blockchains.

Such aspects lead to more frauds and malicious activities to prevail in permissionless blockchains. Ethereum and Bitcoin use permissionless technology.

Ethereum was developed by Buterin (2013) and allows users to run programs in its trusted virtual environment known as Ethereum Virtual Machine (EVM). These programs are called Smart Contracts (SC) and are stored on the ledger along with transactions performed on a given fixed address. Ethereum uses "Ether" as its native crypto-currency for transfer and transaction fees. Smart Contracts (SC) can also send, store, and receive Ether. Once deployed, SC is a hard coded program that could only be fed with input to get output. SCs are also used by some applications for their processing. Such applications are called distributed applications or dapps. Although Ethereum is known for its security and trust, a small bug in a SC code can cause huge loss of crypto-currency (Atzei et al. 2017). Ethereum manages a list of accounts along with the account balance. For a valid transaction usually amount is transferred from a sender to a receiver. If the receiver is an SC, its code is executed and the state of the SC is updated. Internally, a SC could send a message or perform internal transactions with other accounts. Ethereum currently uses a refined form of PoW (Proof of Work) consensus algorithm. PoW is computationally expensive and energy inefficient. Moreover, Appendix 1 presents a brief overview of PoW.

There are vast number of studies in fraud detection (Abdallah et al. 2016). Nonetheless, targeting Ethereum, Chen et al. (2020) base their survey on the attacks and defenses in Ethereum. We do not survey all the attacks and defense mechanisms in this work. However, we provide an in-depth understanding of different methods used to detect accounts involved in malicious activity. Several works have tried to identify or categorize malicious accounts and activities in different types of blockchains. As blockchains have a graph structure, most of these techniques study graph properties (such as node degree) to identify features before applying supervised or unsupervised learning.

In Pham and Lee (2016), authors used a bitcoin transaction network to detect malicious activity. They were able to detect three malicious attacks using unsupervised ML algorithms with a limited amount of transaction data they had. In their followup work, they used a more comprehensive bitcoin transaction dataset (starting from genesis block until April 7*th*, 2013) (Pham and Lee 2017). They employed data in two types of graphs namely *User Graph* and *Transaction Graph*. In the user-graph, nodes represent accounts and edges represent transactions, whereas in the transaction-graph nodes represent transactions and edges represent the flow of bitcoins. They first studied the flow of bitcoins to prove the existence of anomalies and then performed clustering to identify different attacks. They were able to detect the existence of one attack using the *Local Outlier Factor* (LOF). Inspired by Pham and Lee (2017), in Monamo et al. (2016), Monamo et al. also used bitcoin transaction data and proposed an update to counter scaling issues that are inherent in LOF. They validated their approach using *trimmed K-Means*, argued its usefulness in detecting anomalies, and detected 5 out of 30 fraudsters.

In another bitcoin-related malicious activity detection, authors studied the detection of addresses involved in the Ponzi scheme (Bartoletti et al. 2018). They used supervised learning and validated their results after addressing the class imbalance that is inherent in any malicious activity related dataset. They identified that the Gini coefficient of outgoing Ethers and the ratio between incoming and total transactions are the

most important features for detecting Ponzi scheme related accounts. In another Ponzi scheme related study, in Chen et al. (2018b), authors use Ethereum data to extract features from operation codes (opcodes) of the SC's bytecode. Their motivation behind the study was based on the fact that the opcodes reflect the logic implemented in a SC and therefore provide useful features for identifying Ponzi and non-Ponzi SC. They also figured out that opcode based features are more efficient than account based features while detecting Ponzi scheme accounts. In Ostapowicz and Zbikowski (2019), the authors use partial Ethereum transaction data to classify malicious accounts. They also performed a sensitivity analysis to study the effect of different classifiers on the feature set. In Singh (2019), to counter class imbalance, authors assumed that the accounts connected to malicious accounts via incoming transactions are also malicious. They then studied various supervised ML algorithms to identify malicious accounts over this oversampled Ethereum dataset. In a followup study of Singh (2019), in Kumar et al. (2020), the authors used only those benign accounts that have never transacted with malicious accounts. Further, their feature vector had only transaction based properties but not the graph based properties.

N-motifs are frequently occurring subgraphs that serve as a basic building block of a network. The authors in Zola et al. (2019) defined N-motif as a path of length $2N$ between two entities. Here transactions were also considered as vertices. Using N-motifs that are present in the transaction graph, in Zola et al. (2019), the authors studied transactions happening between entities (people or organizations with multiple accounts). They were able to correctly identify malicious accounts involved in gambling. In another study (Goldsmith et al. 2020), the authors analysed transfer of funds within a sub-network and used temporal feature such as how quickly funds are cashed.

We present all the above-mentioned techniques in detail in Table 1 and present the features that the techniques use along with the studied ML algorithms, their hyperparameters, accounts considered in the dataset, and achieved performance score. Note that all these techniques use features that are based on some graph properties, transacting amount, and active state to train the ML algorithm. However, several other studies, such as Chen et al. (2018a) and Cheng et al. (2019), use inferences drawn from the analysis of the transaction graph to mark malicious accounts. In Chen et al. (2018a), the authors try to identify accounts that indulge in DDos attacks and argue that accounts that create multiple rarely used contracts are malicious. A similar approach is followed in Jung et al. (2019) where the authors used only verified SC codes and introduced features like SC size, lifetime, and average time between transactions (i.e. Inter-event time). In Cheng et al. (2019), the authors deploy honeypot and analyze remote procedure call (RPC) requests to identify malicious accounts. They then analyze transactions to mark accounts as suspicious that accept crypto-currency from malicious accounts.They perform behavior analysis to identify *fisher* accounts and attacks such as crypto-currency stealing.

All the above techniques either use a limited set of ML algorithms on a highly scaled-down data inducing over-fitting or apply inferences on the graph structure to identify malicious activities and accounts. In most cases, the studies use features that do not capture temporal behavior and are approximated by the average (mean) behavior, thereby, further inducing a bias in their study thus having high accuracy. On the other hand,

Table 1 Features used in related studies

#	B/C	Used features based on (See Abbreviation section)									ML Algo used	Dataset	Hyperparameters	Performance
		AS	iD	oD	Bal	TF	BB	A	CC	IET				
Pham and Lee (2016)	B	✓	✓	✓	✓	–	–	–	✓	✓	K-means	$100K^{a}$	$k \in [1, 14]$	$k_{opt} = 7, 8$
											Mahalanobis distance		×	0.0256^{MDE}
											ν-SVM		$\nu = 0.005$	0.1441^{MDE}
Pham and Lee (2017)	B	✓	✓	✓	✓	–	-	–	–	✓	Local outlier factor	$6.3M^{a}$	$k = 8$	0.55^{MDE}
Monamo et al. (2016)	B	–	✓	✓	✓	–	–	–	✓	–	K-means	$1M^{a}$	$k \in [1, 14]$	$k_{opt} = 8$
											Trimmed K-means		$k \in [1, 15], \alpha = 0.01$	$k_{opt} = 8$
Bartoletti et al. (2018)	B	✓	✓	✓	✓	–	–	–	–	✓	RIPPER†	$‡6432^{a}$	Cost $\in [1, 40]$	0.996^{ac}
											Bayes network		×	0.983^{ac}
											Random Forest		×	0.996^{ac}
Chen et al. (2018b)	E	–	✓	✓	✓	✓	–	–	–	–	XGBoost	$‡1382^{sc}$	×	$0.94^{p}, 0.81^{r}$
Ostapowicz and Zbikowski (2019)	E	✓	✓	✓	–	✓	–	–	–	–	Random Forest	$350K^{a}$	RFPARAM	$0.85^{r}, 0.05^{p}$
											SVM		$Cost = 1, \gamma = 0.077$	$0.87^{r}, 0.02^{p}$
											XGBoost		XGBPARAM	$0.8^{r}, 0.07^{p}$
Singh (2019)	E	✓	✓	✓	✓	–	–	–	–	–	Decision Tree	300^{a}	×	0.93^{ac}
											SVM		×	0.83^{ac}
											KNN		$k = 5$	0.91^{ac}
											MLP		×	0.86^{ac}
											NaiveBayes		×	0.89^{ac}
											Random Forest		×	0.99^{ac}
Kumar et al. (2020)	E	✓	–	–	✓	✓	–	–	–	–	Decision Tree	9375^{a}	×	0.92^{ac}
											KNN		×	0.92^{ac}
											XGBoost		×	0.96^{ac}
											Random Forest		×	0.95^{ac}

Table 1 (continued)

#	B/C	Used features based on (See Abbreviation section)									ML Algo used	Dataset	Hyperparameters	Performance
		AS	iD	oD	Bal	TF	BB	A	CC	IET				
Zola et al. (2019)	B	–	✓	✓	✓	✓	–	–	–	–	Adaboost	1000M[a]	*Estimators* = 50, *rate* = 1	> 0.2[r]
											Random Forest		*Estimators* = 10	> 0.85[r]
											Gradient boosting		*estimators* = 100, *rate* = 0.1	> 0.93[r]
													Depth = 3	

[B/C] Blockchain, [B] Bitcoin, [E] Ethereum, [a] accounts, [sc] Smart Contracts, [MDE] Dual Evaluation Metric, [ac] accuracy, [p] Precision, [r] Recall, [†] it is a propositional rule learner that relies on a sequential covering logic, [‡] Ponzi scheme data, [RFPARM] features = 3, leaf samples = 10, threshold probability = 0.99, [XGBPARAM] depth = 3, child weight = 8, subsample = 1, probability = 0.99, [×] not provided

techniques that use large datasets and have high class imbalance, either have high recall and low precision or low recall and high precision (Ostapowicz and Zbikowski 2019). Nonetheless, using our features, we identified the ML algorithm that provides not only the best precision but also the best recall. In brief, these measure are described next. Let C be the set of n distinct classes present in a dataset such that $C = \{C_1, C_2, ..., C_n\}$.

- *Precision* For a particular class i, the precision is defined as the ratio of correctly predicted observations of a class to the total observations that are predicted under that class. Let $P_{j,i}$ represents the number of observations of class j that are predicted under class i such that $j \in C$. Formally, precision is defined as:

$$Precision_i = \frac{P_{i,i}}{\sum_{j \in C} P_{j,i}} \tag{1}$$

- *Recall* For a particular class, recall is defined as the ratio of correctly predicted observations of a class to the total observations of that particular class. Let $R_{i,j}$ represents the number of observations of class i that are predicted under class j. Formally, recall is defined as:

$$Recall_i = \frac{R_{i,i}}{\sum_{j \in C} R_{i,j}} \tag{2}$$

- *Balanced accuracy* For an imbalanced dataset, accuracy can be deceiving. However, balanced accuracy gives well enough representation of the performance of the model. Balanced accuracy is defined as the average recall score obtained in each class. Formally, balanced accuracy is defined as:

$$Balanced\_Accuracy = \frac{1}{n} \sum_{j \in C} Recall_j \tag{3}$$

- *F1 score* It is represented by the harmonic mean of precision and recall. It can also be described as a weighted average of precision and recall. Formally, F1 score is defined as:

$$F1\_Score_i = \frac{2 * Precision_i * Recall_i}{Precision_i + Recall_i} \tag{4}$$

- *MCC score* Mathews Correlation Coefficient (MCC) is the correlation between the ground truth and the predicted results. It is defined as:

$$MCC = \frac{\prod_{i \in C} P_{i,i} - \prod_{j \in C - \{i\}} P_{j,i}}{\sqrt{\prod_{i \in C} \left[\left(\sum_{j \in C} P_{j,i} \right) \times \left(\sum_{j \in C} P_{i,j} \right) \right]}} \tag{5}$$

Methodology

We use *Ethereum mainnet blockchain transaction data* and first validate our assumptions and approach. We segment the transaction data into sub-datasets (*SD*) to capture the behavioral changes. We create the *SD*s using different temporal granularities (T_g

such that $T_g \in T_G$) where $T_G = \{Day, Week, Month, Quarter, HalfYearly, Year, All\}$. A granularity becomes coarser as we move from Day to Year. Here, a *SD* in a *Day* consists of transactions of 6000 blocks. The choice of 6000 blocks is based on the fact that in Ethereum approx. 6000 blocks are created every day. At a coarser T_g, a *SD* in a *Week* consists of 7 Days data. Similarly, a *SD* in a *Month* consists of 30 Days data, a *SD* in a *Quarter* consists of 3 Months data, a *SD* in a *HalfYearly* consists of 6 Months data, and a *SD* in a *Year* consists of 12 Months of data.

On all the features that are time series based (features described in Sect. 4), we perform time series analysis of all the *SD*s at different T_g to quantify them. To perform such task, we use *tsfresh* that "extracts characteristics from a time series" (Christ et al. 2016, 2018). The analysis reveals that features such as quantile and median best describe the time series for most of the features we have. We observe this behavior not only in the entire dataset but also in different *SD*s at different T_gs.

We first apply the AutoML pipeline using TPOT (Olson et al. 2016) to identify the best ML classifier on the entire dataset and validate state of the art techniques. From a given set of ML pipelines, TPOT reports the ML pipeline that produces the best results, in our case the best balanced accuracy. We configure TPOT with not only the state of the art supervised ML algorithms and their hyperparameters but also with other hyperparameters to identify best results and to be sure that related studies did identify hyperparameters for which the balanced accuracy was the best. Note that TPOT internally performs imputation and feature scaling also. A detailed explanation of TPOT is out of scope of this work, thus we do not present the internal architecture of TPOT. Besides supervised learning, as our aim is to detect malicious accounts, we also apply clustering to identify accounts that show similar behavior to that of malicious accounts. Note that for training purposes, we use 80% of the dataset while we test on the remaining 20% of the dataset. For the entire dataset, we find that K-Means provides best silhouette score for $k = 9$ when we consider both EOAs and SCs while $k = 10$ when we consider only EOAs. For the clusters identified as malicious, we use cosine similarity to quantify the similarity among the accounts within a cluster. We acknowledge that there are other methods as well to identify similarity, but for this work we use cosine similarity. Assuming that K-Means with hyperparameter $k = 9$ identified for entire dataset (both EOAs and SCs) performs the best for all temporal sub-datasets at different temporal granularities, we determine a probability for an account to be malicious at different temporal granularities.

In summary, Fig. 1 presents an overview of our methodology. Here, for the dataset we have, we first pre-process the data to generate necessary graph structure and segment the it based on different $T_g s$. On the full dataset, we then identify distributions and generate necessary heat maps. All the data segments are then passed to the feature constructor which captures all the needed features. As there are two types of features: time series based and non-time series based features, for all the time series based features in each data segment we ran tsfresh. The results, depending on the data segment, are then integrated with the non-time series based features in that data segment. Once we obtain the needed features, we further segment the data based on EOAs and SCs and perform PCA (Principal Component Analysis) to reduce the dimensionality of our dataset. We then execute supervised ML algorithm on the

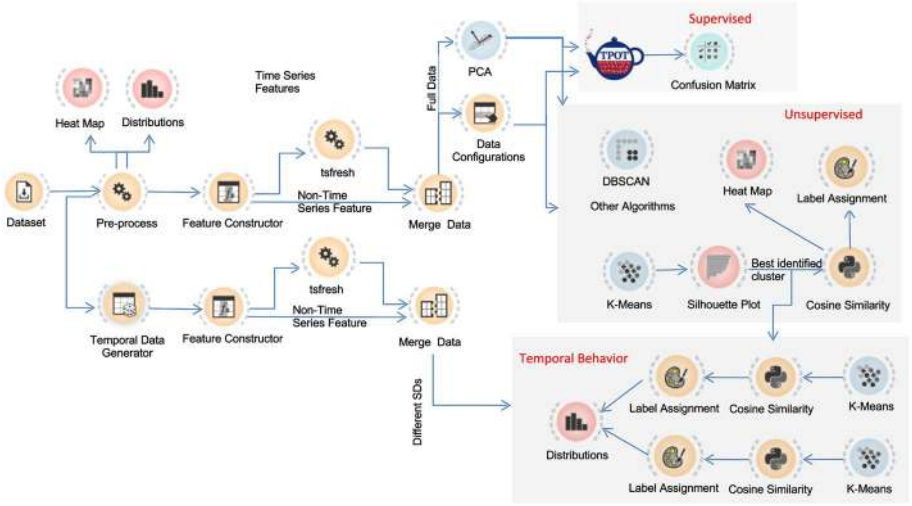


Fig. 1 Overview of our methodology

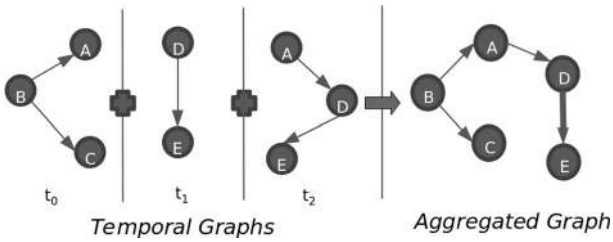


Fig. 2 Sample blockchain temporal graphs and corresponding aggregated graph

derived datasets using AutoML tool to identify the best ML algorithm. Further, we also apply different unsupervised ML algorithms, for example K-means, to identify the number of clusters for which the silhouette score is the best, then perform cosine similarity to identify potential benign accounts that may seem malicious. Once the best setting for clusters is identified, we use it for the temporal analysis on different SDs and generate probability distribution for an account to be deemed malicious.

Feature engineering

A typical blockchain is a temporal-directed graph. Let $G^t(V^t, E^t)$ be the temporal snapshot of the blockchain graph, where V^t is a set of accounts at time t and E^t is the set of directed edges or the transactions that happen between two accounts in V_t at time t. Figure 2 shows 3 temporal snapshots of blockchain graph where 5 accounts interact with each other at different epochs. Here, at time t_0, 3 accounts 'A', 'B', and 'C' interact where one account (account 'B') sends crypto-currency to the other 2 accounts. Similarly, at t_1 and t_2, 2 and 3 accounts interact, respectively. The figure also shows corresponding aggregated graph where the edges with more than one occurrences are weighted with the number of occurrences.

We used both temporal and aggregated graph structure of blockchain to extract features. The set of features (F) defined in the related work is limited and, in most cases, does not convey correct temporal behavior. We extend the feature set and introduce new features to detect malicious accounts. We follow a two-fold methodology to identify the relevant features. First, we study different attacks that have happened in the past to understand what features malicious accounts revealed after engaging in malicious activity. Second, as most of the account features (for example, inDegree) are time series based, we perform time series analysis to identify features that best represent the salient properties of the relevant time series. Below we provide a list of all the features we use:

- *Non Time Series based (set $F_n|F_n \subset F$)*
  - *Active state (AS)* malicious activities are usually short-lived (Aspembitova et al. 2019) and remain, for example, until remediation is introduced. It is thus essential that we consider features such as when the account first transacted (*transactedFirst*), last transacted (*transactedLast*), how long it has been active (*durationActive*), and since when the account is continuously transacting (*activeSinceLast*).
- *Time series based (set $F_t|F_t \subset F$)* We analyze each of the following time series based features using *tsfresh* (Christ et al. 2016, 2018) and select 3 top features identified for each of the following attributes. Nonetheless, as *inter-event time (IET)* itself is a time series, we use it as a feature as well.

  - *inDegree (iD)* it represents the number of transactions in which the account under consideration is a receiver at a particular instant. Most of the malicious activities involve transfer of money to a malicious account. Thus, it is one of the most important features used to understand the behavior of a malicious account. In Singh (2019), the author found *uniqueInDegree* (defined as unique accounts from which the account under consideration has ever received money) to be one of the most critical features for identification of malicious accounts. On top, we also use aggregated *inDegreeAgg* as a feature.
  - *outDegree (oD)* represents the number of transactions in which the account under consideration has sent money at a particular instant. In some attacks such as Bitpoint Hack (O'Neal 2020), after the attacker has received amount of sum from the victims in an alias account, they transferred the received sum to another account they hold or to an exchange. Such attacks increase the importance of outDegree as a potential feature. Similar to the case above, we also use aggregated *outDegreeAgg* as a feature.
  - *Balance (Bal)* our motivation to use it as a feature is based on the fact that most malicious activities in a permissionless blockchains are finance based. For example, in one of the famous Parity Multisig wallets (Palladino 2020) attack, the malicious account drained more than 150k Ethers. Thus, the currency held by an account as well as its flow are important features. We identify balance time series for both in/out case. Besides balance, we identify for each instance, max balance for both in and out cases (*maxInBalance* and *maxOutBalance*), *zeroBalanceTransactions* (transactions where no money was transferred either

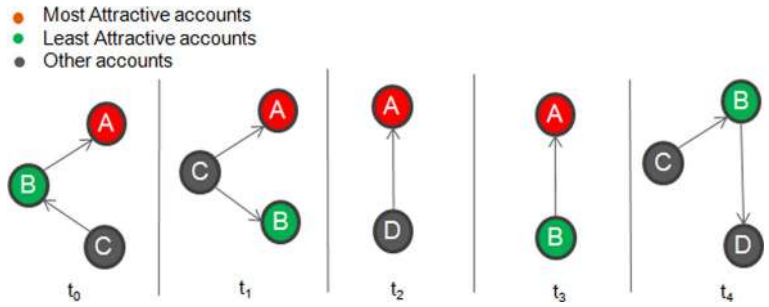


Fig. 3 Sample graph depicting the most attractive and least attractive accounts

to or from an account), *totalBalance* (final balance held with the account), and *averagePerInBalance* (average of received balance) as features.

- *Transaction Fees (TF)* In crypto-currency based blockchains, a transaction is marked by *transaction fees* that a sender is willing to spend on a particular transaction. In the Ethereum blockchain, $TF = Gas \times GasPrice$ where *Gas* is the measure of the amount of computation required to perform the transaction by a miner and *GasPrice* is the amount of Ethers a user is willing to pay per unit Gas. Operations like transferring of Ethers require a fixed sequence of instructions i.e. fixed amount of computation so it consumes fixed amount of Gas (21000). Therefore, if a high transaction fee is paid, it suggests that the user is willingly setting a higher *GasPrice*, since, for a transaction, unit of Gas is a fixed. Several attackers put higher gas price to bribe the miner so that a particular transaction of interest to them is included in the next block (Cheng et al. 2019). Nonetheless, in DDos attack (Buterin 2020), an attacker created multiple accounts at very low gas price to increase synchronization and processing time. Thus, it is also a feature.
- *Attractiveness (A)* mostly, malicious accounts tend to interact with accounts that they have not interacted with before. The probability of interacting with the same account that they have interacted before is very low. Attractiveness can be understood as the change in the neighborhood of an account over time. This can be modeled using the stability of the neighborhood. Consider 5 accounts in the blockchain that interact with each other and perform transactions with each other as shown in Fig. 3. From the figure, we can see that account 'A' received crypto-currency from accounts that change almost in every time slot. On the other hand, account 'B' only received crypto-currency from account 'C'. This stability of incoming transactions deem account 'B' as more stable and less attractive than account 'A' which is more unstable and attractive to other accounts. Consider, N_i^t to be the neighborhood (accounts from whom the account i has received crypto-currency) of account i at time t, $T = \{t, t-1, \cdots, t-\theta_a\}$, and θ_a the time window size. Based on this, we define *attractiveness* (A_i^t) for account i at time t as shown in Eq. 6.

$$A_i^t = \begin{cases} 1 - \frac{\left|N_i^t \cap \left(\bigcup_{j \in T-\{t\}} N_i^j\right)\right|}{\left|\bigcup_{j \in T} N_i^j\right|}, & \text{when } j \geq 0 \text{ and } N_i^t \neq \emptyset \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

In the Eq. 6, the numerator represents the number of unique accounts that the account i has received crypto-currency from that were present in past $\theta_a - 1$ blocks/times. While the denominator represents the total number of unique accounts from which the account has received crypto-currency from, in the last θ_a blocks. To get the attractiveness value, we subtracted this ratio from 1. Note that we specifically considered in-degree and not the degree of the account because we were motivated by the fact that most of the malicious activities involve extracting crypto-currency from a benign account.

As the blockchain graph is a temporal graph where interactions are intermittent, an account's neighborhood is highly dynamic and changes at each epoch. Thus, the attractiveness also changes with respect to time (as new accounts can be added to the neighborhood at the epoch under consideration). Therefore, we define *attractiveness* as a temporal feature.

- *Burst (BB) (set $F_b | F_b \subset F$)* bursty behavior is defined as temporal non-homogeneous sequence of events (Karsai et al. 2012) and has been characterized by a fat-tailed inter-event time (Δt) distribution. In one of the bitcoin blockchain attacks (Allin-vain Theft 2020), a malicious account generated a large number of transactions to taint the bitcoin platform. Motivated by this incident, we define four types of bursts (temporal, degree, balance and gasPrice) that occur in the network under consideration. As an account can either be a sender or a receiver, the following burst types are defined for cases *(a)* when the account acts as a sender, *(b)* when the account acts as a receiver, and *(c)* when the account acts as both a sender as well as a receiver.

  - *Temporal Burst* for an account i, non-homogeneous occurrences of events (in our case transactions) lead to some transactions occurring where Δt is less than a threshold, θ_t^i, while for other transactions Δt is large. If a transaction happens when $\Delta t < \theta_t^i$, we assume that it is a burst. Some burst can be long lived while some burst can be short lived, meaning, some event can happen continuously for long time intervals before going dormant. As features, we identify number of such temporal bursts (*numberOfTemporalBursts*) and the duration of the longest burst (*longestBurstDuration*) for both in and out transactions separately.
  - *Degree Burst* it has been proven that the degree (also inDegree and outDegree) distribution of the aggregated transactions in blockchain such as Ethereum follows a power-law (fat tailed) distribution (Chen et al. 2018a) with $\alpha \in [-2.8, -2.6]$. This suggests that many accounts do not transact often while there are very few accounts that act as hubs (for example, exchanges). Nonetheless, when considering the temporal aspects, we believe such behavior also exists where some accounts have a very high degree for some instant while for other instants they

have a low degree. Thus, we define a degree burst when at a given instant of time the degree of an account, i, is greater than θ_d^i. Similar to the temporal case, for degree bursts we also identify number of degree bursts (*numberOfDegreeBursts*) that happened for an account over time, number of instances where the degree burst happened (*numberOfDegreeBurstInstances*), and the time at which the largest burst of degree happened (*largestBurstAt*). Note that these features except for numberOfDegreeBurstInstances are defined for both in and out transactions separately as well.

- *Balance Burst* in some cases transactions happen from accounts i to account j where the involved sum of crypto-currency was very large (more than a threshold value θ_b^i). For example, some accounts associated to Silk Road (Spagnuolo et al. 2014) or involved in money laundering, sometimes transact large sum for illegal activities. Bursty behavior of transaction amount could be helpful in identifying potential malicious activities and accounts. Similar to the above cases, for an account i, we identify number of unique instances where balance is more than θ_b^i (*numberOfBalanceBurstyInstances*), and number transactions more than θ_b^i (*numberOfBalanceBursts*). Note that, we define these factors for both in and out cases, separately.
- *GasPrice Burst*: As described before, an attacker can put higher gas price (more than a threshold value θ_g^i) to bribe the miner so that the transaction is included in the block. This activity although abnormal is useful in understanding account's behavior. Towards this, similar to previous cases, we define *numberOfGasPriceBurstyInstances* as number of instances where the gasPrice was set more than θ_g^i. This is only defined for senders as gasPrice is only set by the sender.

Note that features such as in/outDegree, burst, attractiveness are some graph-based temporal features. Besides these features, other graph-based properties that we use as features include *clustering-coefficient (CC)* (Watts and Strogatz 1998). For an account i, let $N_i^{t,in}$ be the neighborhood of account i at time t from which the account has received the crypto-currency, $N_i^{t,out}$ be the neighborhood of account i at time t to which the account has paid the crypto-currency. Thus, the total account degree is $deg_i^{tot} = |N_i^{t,in}| + |N_i^{t,out}|$. Let $N_i^{t,\leftrightarrow} = N_i^{t,in} \bigcap N_i^{t,out}$ and $a_{ir} = 1$ if there is a transaction between i and r, otherwise 0. We similarly define a_{is}, a_{ri}, a_{si}, a_{rs}, a_{sr}. For a directed graph, CC of account i (CC_i^t) at time t is defined by Eq. 7 (Fagiolo 2007).

$$CC_i^t = \frac{\sum_r \sum_s (a_{ir} + a_{ri})(a_{is} + a_{si})(a_{sr} + a_{rs})}{2\left[deg_i^{tot}(deg_i^{tot} - 1) - 2|N_i^{t,\leftrightarrow}|\right]}. \tag{7}$$

Results and evaluation

We evaluate the effectiveness of our method using Ethereum's external transactions data which is publicly available for download using the *Etherscan APIs* (Etherscan 2020a). Note that the APIs do not provide any information about the account (such as the name and the account type). Nonetheless, as the hash of the accounts is available, one can check the associated information using the Ethereum Blockchain Explorer (Bitfly Gmbh 2020). We perform all our evaluations using Python version 3.7. Within the Python environment, we use supporting libraries such as TPOT version 0.10.2, tsfresh version 0.13.0, scikit-learn version 0.21.3, pandas version 0.25.1, and numpy version 1.16.5. All the result plots are generated using matplotlib version 3.1.1.

Dataset

Ethereum as on 20th December 2019 had ≈79M accounts. Out of these accounts, 3362 accounts were already tagged to be involved in malicious activities. The tags mainly include Phishing (3168 accounts), Gambling (8 accounts), Cryptopia-Hack (6 accounts), Heist (16 accounts), Suspicious (4), Bitpoint Hack (2 accounts), Compromised (21 accounts), Spam (10 accounts), Upbit-Hack (123 accounts), Unsafe (1 account), Scam (1 account) and Bugs (2 accounts). Note that to get such ground truth about the accounts, we explicitly browsed through all the tags available in Etherscan and marked those accounts as malicious where the associated tags had caution warnings. Moreover, we chose the accounts associated with the tag *Gambling* as this type of activity is illegal in most countries. We look for other sources such as Cryptoscam.db (MyCrypto Inc 2019) as well to know the ground truth about the accounts as only some of the accounts are tagged as malicious. As a result, we find 329 more malicious accounts with a total of 3691 unique malicious EOAs and SCs. Upon further investigation, we find that out of these 3691 EOAs and SCs, 746 never transacted and were mostly involved in the token trade until 7th December 2019. We, thus, remove them from our malicious accounts dataset. In these remaining set of malicious accounts, there are 158 SCs and 2 marked compromised exchanges. Note that for these accounts we collect only-but-all external transactions (transactions from EOAs to SCs, and between different EOAs). Also note that at the time of this study Etherscan had removed most of the malicious tags. But recently Etherscan provided new tags and marked more accounts as malicious. As of 27th May 2020, there were 4708 malicious accounts out of which 2019 were newly tagged accounts. Out of these 2019 accounts only 1252 accounts ever transacted. Out of these 1252 accounts 1029 were created before 7th December 2019 in which only 3 are present in our dataset. As the number of malicious accounts is constantly evolving, we take this opportunity to cross validate accounts that our analysis found malicious.

There is a high class imbalance in the dataset as the number of benign accounts is large. Thus, we perform random under-sampling to uniformly sample 697K benign accounts from the 79M Ethereum accounts. In the total ≈700K accounts we have, there are 7 exchanges and 23,141 SCs while the remaining accounts are EOAs.

A unique external transaction, *Tx*, retrieved using Etherscan API contains information like blockHash, blockNumber, source, destination, gas, gasPrice, Transaction hash, balance, and timestamp of the block. A typical Tx is shown in Fig. 4. Here,

- *blockNumber* is the number associated with a block in which the transaction (represented by a 32-byte long transaction hash stored in *hash*) is stored by the miner

```
{
   "blockNumber":"9385624",
   "timeStamp":"1580414942",
   "hash":"0x649660bd207ecccfab0d207b5809cb21054486c7fcad3cf3c177e9c44a16a0e4",
   "nonce":"0",
   "blockHash":"0xf5b1b6c29fa4e26c9e31c8d999139a513446c9638ef8a96abc011caa8cd45212",
   "transactionIndex":"81",
   "from":"0x8f6e4a32c37b1e,4c4f8bec45c85b73fe310290a1",
   "to":"0x00dbc5d9f5c3cfb4f9f889ad0bf20650a21c6c39",
   "value":"1000000000000000000",
   "gas":"21000",
   "gasPrice":"3000000000",
   "isError":"0",
   "txreceipt_status":"1",
   "input":"0x",
   "contractAddress":"",
   "cumulativeGasUsed":"2453188",
   "gasUsed":"21000",
   "confirmations":"767669"
}
```


Fig. 4 A sample Ethereum transaction

at index represented by *transactionIndex*. A block is also uniquely identified by a 32-byte long hash (i.e. *blockHash*) . The *timestamp* represents the epoch timestamp, in seconds, at which the transaction was mined.

- The *from* and *to* fields are both 20-byte long hashes of public keys of sender and receiver accounts. These accounts can be either EOAs or SCs. In a transaction, if the *to* field is empty, then it represents that the transaction is a contract creation transaction. In this case the *input* field contains the SC code. However, if *to* field is a hash of a SC, then *input* field may contain a function call. In other cases, the *input* field is an arbitrary message or Null.
- In a transaction, *nonce* represents the number of outgoing transactions performed by an EOA sender before that particular transaction. This nonce, in the case of SC, represents the number of secondary SCs created by it. *Nonce* along with the sender's address is used to calculate the address of the SC.
- The *value* field represents Ethers transferred (in Wei). 1 Ether is equal to 10^{18} Wei.
- The maximum amount of gas provided for the transaction is specified using *gas*. The gas is the upper limit of computation work that sender allows a miner to perform the transaction. Whereas *gasUsed* is the actual amount of computation performed. The cost per unit of gas that the sender is willing to pay for a transaction is represented by *gasPrice* (in GWei $= 10^9$ Wei). An attacker can set a higher gasPrice for better chances of his transaction being included in the block.

Note that the Tx data does not include the actual timestamp of when a transaction was performed by the account. The available timestamp is the time when the miner added the Tx to the block. The only time related information, we are able to extract is the information about when a block is mined. However, currently we do not use this information. We assume a time bin of 1 block for our study. We assign respective *blockNumber* as a timestamp to all the transactions[1]. Based on this notion of timestamp, we also segment

[1] The block numbers are monotonically increasing thus giving a notion of timestamp.

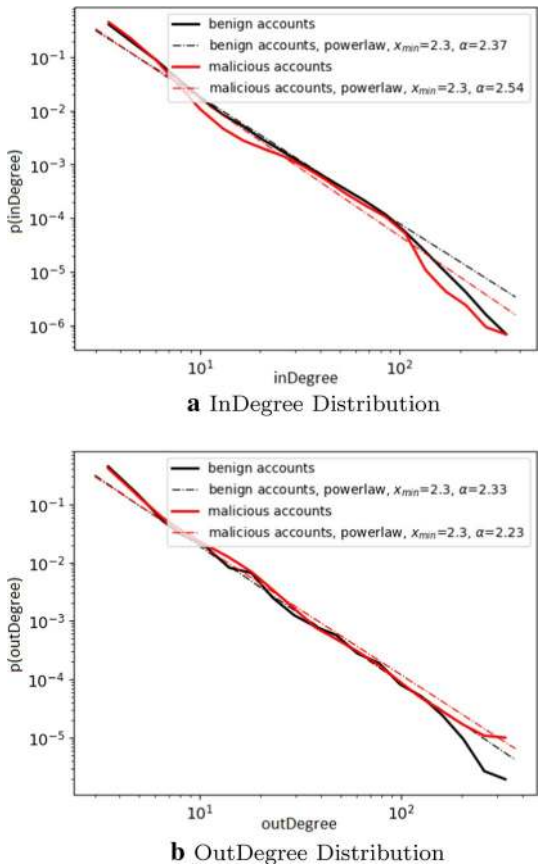


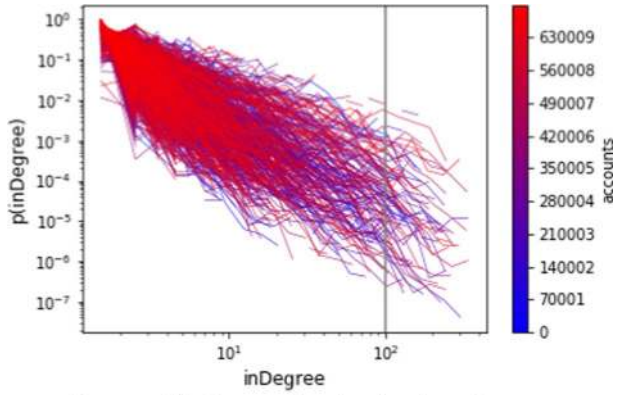
Fig. 5 Degree distribution of accounts

the data into several *SD*s of different T_g and study the behavior of the accounts. We describe in the Sect. 1 the different T_gs we consider. For statistical purposes, we have 1,531 Day *SD*s, 219 Week *SD*s, 52 Month *SD*s, 18 Quarter *SD*s, 9 HalfYearly *SD*s, 5 Year *SD*s, and the entire dataset. A total of 1835 datasets.

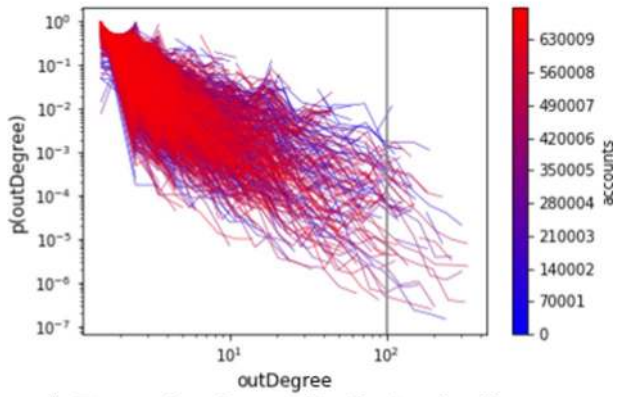
For our study we assign: *(i)* $\theta_t = 2$ so that continuous burst of smallest size are also captured, *(ii)* for an account i, $\theta_d^i = 0.8 \times (max(d))$ where d is the in/outdegree of an account in the considered *SD*, *(iii)* $\theta_b^i = 0.8 \times (max(b))$ where b is the transaction balance for either in or out case, *(iv)* θ_a^i is set to be equal to the duration of the *SD* to keep the entire history of neighbors that a particular account transacted with in the past in the given sub-dataset, and *(v))* $\theta_g^i = 0.8 \times (max(gasPrice))$ where gasPrice is the the gas price for transactions associated with account i. We then analyse different time series based features to identify there characteristics as potential features.

Results

In this subsection, we present results obtained after analysing the dataset and using different ML techniques. Here, we also present results obtained after performing behavioral analysis.



a Temporal inDegree distribution for all accounts



b Temporal outDegree distribution for all accounts.

Fig. 6 Temporal degree distribution of individual accounts

Data analysis

For the entire dataset, we first study inDegree and outDegree distribution for both malicious and benign accounts to validate the fat-tailed behavior of the degree distribution. From Fig. 5, we identify that power-law distribution (Alstott et al. 2014) with $x_{min} = 2.3$, $\alpha \in [2.37, 2.54]$ and $\alpha \in [2.23, 2.33]$ fits inDegree and outDegree distribution, respectively, for both malicious and benign accounts. Here α and x_{min} are the powerlaw exponent and minimum x from where the powerlaw distribution is observed, respectively.

The fat-tailed nature of degree is evident because some accounts interact with more number of accounts at a certain instant, thereby inducing a bursty behavior. We study the distribution of inDegree for all individual accounts to understand if such behavior is shown by all the accounts. Figure 6a presents distribution of inDegree for different accounts. We identify that the inDegree of very few accounts is high (> 100) for very few time instances while most of the time it is low suggesting the existence of bursts. We observe a similar behavior for outDegree as well (see Fig. 6b).

Next, we validate the existence of temporal bursts. For this we study the distribution of inter-event time (Δt) for all accounts. We find that it follows power law with $x_{min} = 3$ and $\alpha = 1.25$ and $\alpha = 1.76$ for benign and malicious cases, respectively (see

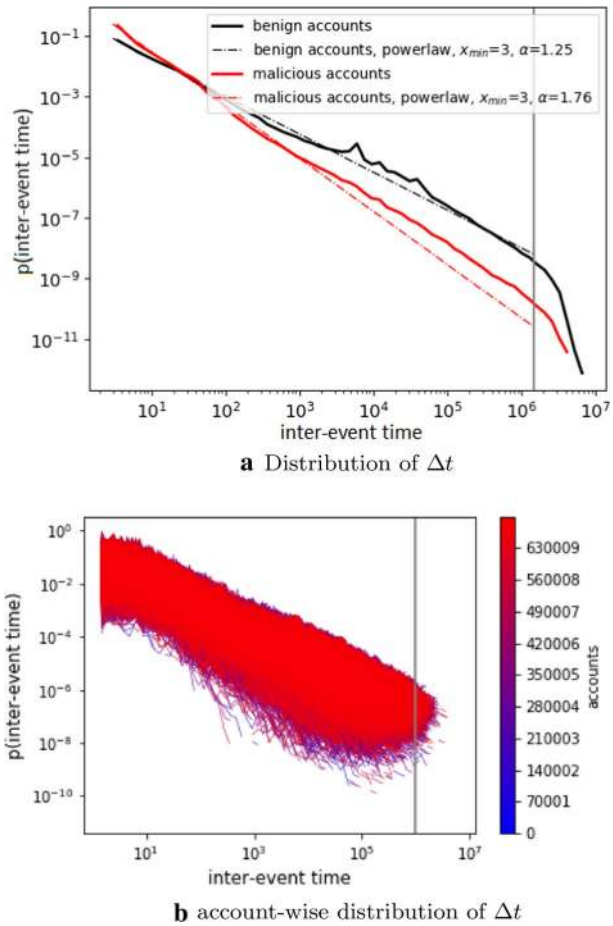


Fig. 7 Distribution of Δt

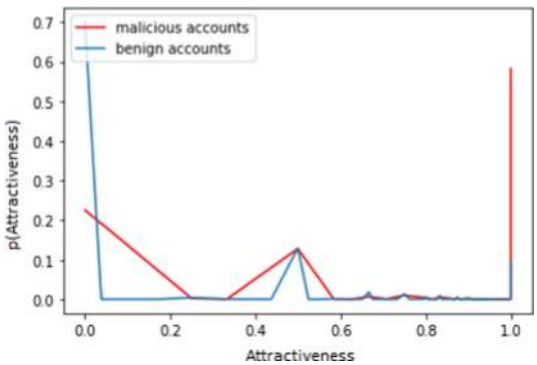


Fig. 8 Attractiveness

Fig. 7a). Nonetheless, we also observe a truncation at 1.5×10^6 blocks. The truncation reflects that some accounts are inactive or did not perform any transactions for long period of time. When looking at the individual level, we observe that only few accounts have very large inter-event time ($> 1 \times 10^6$) where the probability of

occurrence of such events is very low. Most of the activity happens where the inter-event time is very small (see Fig. 7b).

The attractiveness behavior of malicious and benign accounts differ significantly (see Fig. 8). Most malicious accounts have high attractiveness value while most of the benign accounts have low attractiveness value. This justifies our assumption that most malicious accounts target those accounts that they have not previously interacted with. Some attacks (Upbit Hack - Fake_Phishing1431: '0xdf9191889649c442836ef55de5036a7b694115b6') uses multiple accounts to evade detection while transferring money to exchanges. They use multiple accounts as buffer between account and exchange. This is the reason for relatively high probability ($p(A = 0) > 0.2$) for the low values of attractiveness ($A = 0$) for malicious accounts. Similarly, for some benign accounts $p(A = 1) = 0.1$ because such accounts only have 1 incoming transaction in whole lifetime portraying that the account interacted only with new accounts.

Feature extraction

For the entire dataset, after applying tsfresh, for every temporal feature $F_t^j \in F_t$ we get a set of features ($\hat{F}_t^j$) that describes F_t^j. tsfresh in most of the cases reported more than 400 features. But, due to computational resource constraints, we were not able to use all the features from $\hat{F}_t^j$. Therefore, we choose only the top three features identified using *Gini* as the scoring method. Here, we have selected 3 features because for most of the $F_t^j \in F_t$, top 3 features turned out to be equally important. After this process, we get a total of 59 features. Appendix 2 provides extracted feature vector of a sample account. For the entire dataset, using *pearson correlation*, we remove highly correlated features and find 36 important features. We also perform PCA to identify 28 features that cover >98.2% variance to further reduce the feature space in the entire dataset.

Approach using supervised learning

For the analysis purposes, besides performing PCA to identify 28 features and before running the AutoML tool (TPOT) to identify the best supervised learning algorithm, we segment the entire dataset into six dataset configurations. Note that these six dataset configurations are different from the temporal *SD*s. Three out of these six dataset configurations use all types of accounts (EOA and SC) and have 59, 36, and 28 features, respectively. While for the remaining three, we separate EOAs from SCs and use only EOAs. These three configurations again have 59, 36, and 28 features, respectively. We configure TPOT with all the supervised ML algorithms used in the state of the art studies along with other supervised ML algorithms to identify the algorithm that gives best balanced accuracy.

As mentioned in the Sect. 3, from a given set of ML pipelines, TPOT reports the ML pipeline that produces the best results, in our case the best balanced accuracy. This results in the evaluation of multiple ML models that are even beyond those that are reported by the related works. We, thus, decided to limit ourselves from reporting results of all the models and instead report the model for which the best balanced accuracy was achieved.Table 2 lists different dataset configurations we have used along with the algorithm that provided the best balanced accuracy along with precision, recall, and f1-score for each class and reports Mathews Correlation Coefficient (MCC).For each

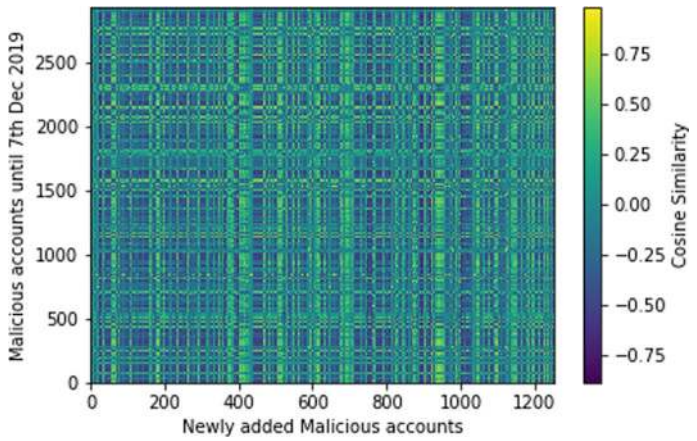


Fig. 9 Cosine similarity between malicious accounts identified on different dates

Table 2 Balanced accuracy, Precision, and Recall, and F1 score for both malicious (Mal) and benign (Ben) accounts with best identified ML algorithm for supervised case when using different dataset configurations

Features	Data segment	Best classifier	Accuracy balanced	Precision		Recall		F1 score		MCC score
				Mal	Ben	Mal	Ben	Mal	Ben	
28	Only EOA	ExtraTrees	0.872	0.38	1.00	0.75	0.99	0.50	1.00	0.510
(PCA)	EOA and SC		0.873	0.22	1.00	0.76	0.99	0.34	0.99	0.421
36	Only EOA		0.876	0.11	1.00	0.78	0.97	0.19	0.99	0.254
	EOA and SC		0.882	0.24	1.00	0.78	0.99	0.37	0.99	0.429
59	Only EOA		0.881	0.26	1.00	0.77	0.99	0.38	0.99	0.425
	EOA and SC		0.887	0.29	1.00	0.78	0.99	0.42	1.00	0.473

Here, we also report the Matthews Correlation Coefficient (MCC) score

28 (PCA) EOA ExtraTreesClassifier(class_weight = 'balanced', max_features = 0.4, max_samples = 0.3, min_samples_leaf = 11, min_samples_split = 19, n_estimators = 600)

28 (PCA) EOA and SC ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.25, max_samples = 0.15, min_samples_leaf = 13, min_samples_split = 4, n_estimators = 800, n_jobs = 20, random_state = 100)

36 EOA ExtraTreesClassifier(bootstrap = true, class_weight = 'balanced', max_features = 0.15, max_samples = 0.7, min_samples_leaf = 8, min_samples_split = 18, n_estimators = 200, n_jobs = 10, random_state = 100)

36 EOA and SC ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.45, max_samples = 0.75, min_samples_leaf = 18, min_samples_split = 6, n_estimators = 200)

59 EOA ExtraTreesClassifier(class_weight = 'balanced', max_features = 0.2, max_samples = 0.75, min_samples_leaf = 13, min_samples_split = 19)

59 EOA and SC ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.3, max_samples = 0.3, min_samples_leaf = 14, min_samples_split = 20, n_estimators = 200)

dataset configuration and the algorithm that provided the best balanced accuracy, we only provide values of those hyperparameters for which the values are different from the default case. We identify that ExtraTreesClassifier provides overall best balanced accuracy for all the dataset configurations. Among the different dataset configurations we have, the dataset with 59 features and containing both EOAs and SCs shows the best balanced accuracy. The difference in balanced accuracy score between the dataset configurations when 36 and 59 features are used is only 0.5% for both when we consider only EOAs and all the accounts, respectively. Given such results, we show that correlated features do not provide much gain and can be removed without the loss of accuracy.

Table 3 Balanced accuracy, Precision, and Recall, and F1 score for both malicious (Mal) and benign (Ben) accounts for supervised learning algorithms used in state of the art approaches where hyperparameters were reported

Reference	Classifier	Accuracy balanced	Precision		Recall		F1 score		MCC
			Mal	Ben	Mal	Ben	Mal	Ben	Score
Ostapowicz and Zbikowski (2019)	RandomForest	0.64	0.98	1.00	0.29	1.00	0.44	1.00	0.52
	SVM*	–	–	–	–	–	–	–	–
	XGBoost	0.85	0.97	1.00	0.7	1.00	0.91	1.00	0.81
Singh (2019)	KNN	0.73	0.89	0.99	0.48	0.99	0.62	0.99	0.64

Here, we use 59 features and both EOA and SC dataset configuration. Here, we also report the Matthews Correlation Coefficient (MCC) score

*Did not converge as the dataset was large

Further, for completeness, the Table 3 shows balanced accuracy, precision, recall, and MCC score for the ML algorithms and their hyperparameters that the state of the art algorithms use. Here, we specifically focus on the Ethereum based state of the art algorithms for which the hyperparameters were reported. From the Table 3, we can see that the best balanced accuracy achieved was 85% while using our approach, we were able to achieve $>$ 88.7% balanced accuracy. However, when comparing MCC scores, we note that although XGBoost has a high MCC score, it has relatively low Recall on malicious account. Having a low Recall would mean that algorithm is deeming a large number of malicious accounts as benign. This is not desired as there could be an adversarial attack. Though for our case the MCC score is low, recall is high. Therefore, our model is more robust towards adversarial attack.

To validate our results, we test ExtraTreesClassifier with identified hyperparameters on newly identified set of 1252 malicious accounts. The classifier achieves 50% balanced accuracy. However, when we train the classifier with identified hyperparameters on the total dataset (dataset consisting of previously used 700k accounts and new 1252 accounts), we were able to achieve $\approx$ 92% balanced accuracy. This makes us wonder if the new malicious nodes have different characteristics. We check cosine similarity between the old 2946 malicious accounts and the new 1252 malicious accounts (see Fig. 9). We find that most of the newly added malicious accounts had low similarity score. Only one new malicious account had similarity score $>$ 0.985 with only one old malicious account. In many cases the similarity score even reached < -0.89 showing that the accounts are not similar and there are some new aspects used by new malicious accounts. Note that to identify cosine similarity we do not use features such as *transactedlast* and *transactedFirst* because many of the accounts were created after 7th Dec 2019. This shows malicious actors also evolve the ways they use to misuse crypto-currency platforms for their fraudulent activities.

Approach using unsupervised learning

We next test unsupervised learning algorithms such as K-Means, DBSCAN, HDBSCAN, and oneClassSVM to identify suspect accounts in the entire dataset. We find that for the six dataset configurations (mentioned above and not the *SD*s) and different values of $k \in [3, 24]$, K-Means provide the best silhouette score (score $= 0.365$) when $k = 10$ clusters and when we use all the features but only EOAs ('59 - EOA') (see Fig. 10). Among

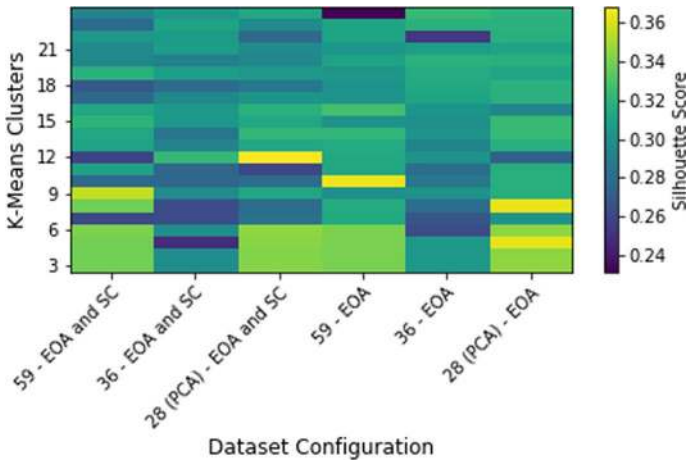


Fig. 10 Silhouette scores of clusters identified by K-Means where $k \in [3, 24]$ for different dataset configurations

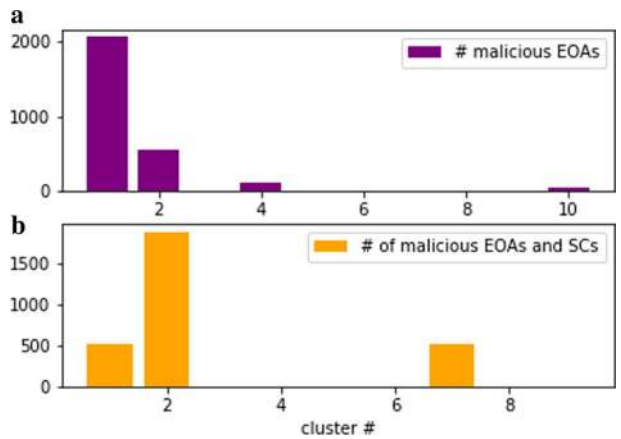


Fig. 11 Clusters with number of malicious accounts for **a** when only EOAs are considered, **b** when both EOAs and SCs are considered

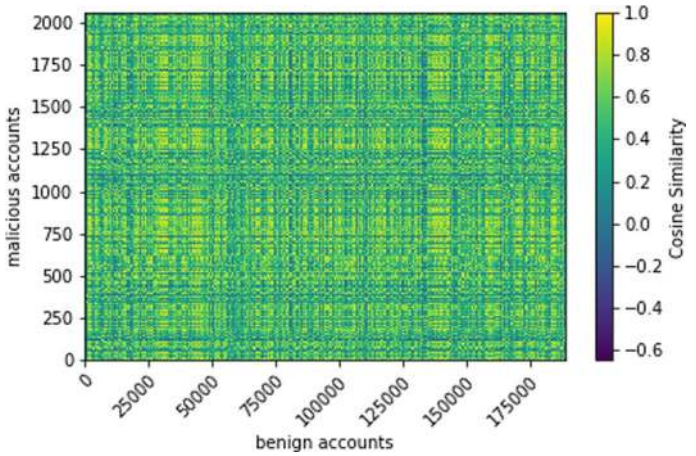


Fig. 12 Cosine similarity between malicious accounts and benign accounts in the cluster with best Silhouette score

these 10 clusters, for one initial condition, one cluster had the most number of already known malicious EOAs ($\approx$ 73.9% (2062/2788)) (see Fig. 11). We then identify the similarity between all the accounts in the identified cluster. We identify 554 benign accounts whose behavior (cosine similarity) (see Fig. 12) is within $1-\epsilon$ where $\epsilon \to 0$ to that of malicious accounts. For our analysis we use $\epsilon = 10^{-7}$. We cross validate the transactions performed by these 554 benign accounts and find that *(a)* most of the EOAs have small *transactedLast* value, meaning, those accounts never transacted in recent past (in past 6 months 494 EOAs never interacted), *(b)* atleast 38 EOAs only have incoming transactions and are not exchanges, and *(c)* *totalBalance* $\in$ [0.0, 150.0] Ethers with a median of 0.001 Ethers.

When considering both EOAs and SCs, we obtain the best silhouette score (score = 0.356) for $k = 9$ clusters but for the case when we use all the 59 features ('59 - EOA and SC') (see Fig. 10). In this case, for one initial condition, there was one cluster with a maximum number of already tagged malicious EOAs ($\approx$ 64.3% (1793/2788)) and malicious SCs ($\approx$ 62.6% (99/158)). We identify 293 potential suspects EOAs and no suspect SCs within this cluster using our previous method. Out of these 293 accounts, 160 EOAs were also detected in the set of 554 accounts. We further tested if the accounts we identified as suspects are present in the list of newly tagged malicious accounts. We found that none of the 3 new malicious tagged accounts that transacted during our analysis period were not in our list of suspects. This is possible as the accounts must have changed their behavior and become malicious after our collection period. We do not reveal the account hash for the sake of privacy and not maligning benign accounts in interacting with any of these 554 or 293 suspects until they are officially tagged malicious. Other unsupervised ML algorithms did not perform better than K-Means. The range of silhouette scores for HDBSCAN was $\in [-0.06, -0.022]$ while oneClassSVM did not converge.

Given such results, we advocate the use of the approach using unsupervised learning. Approach using supervised learning failed to classify correctly new malicious accounts. Moreover, for the unsupervised case, even in the initial dataset, our approach was able to identify more accounts as suspects. Thereby aligning with our motivation of detecting suspects in the blockchain. Keeping, such results in mind, we used unsupervised learning approach to detect behavior changes before deeming any account as malicious.

Behavior analysis

To further understand the temporal behavior changes before classifying the accounts as malicious, we use temporal sub-datasets (*SD*s) created at different temporal granularities (T_g, see Sect. 1). Consider a $T_g \in T_G$ of several *SD*s. Let this set be $SD(T_g)$ where $SD(T_g) = \{SD(T_g)_1, SD(T_g)_2, \cdots, SD(T_g)_j, \cdots, SD(T_g)_n\}$. Further, consider an account i. We first analyse all the time-series based features in each $SD(T_g)_j$ and characterise them. We employ a similar approach as before where we identify $\hat{F}_t^i$ using tsfresh for a $F_t^i \in F$ in a given $SD(T_g)_j$ and use three features in $\hat{F}_t^i$ with the highest *gini* scores.

We then use K-Means with previously identified hyperparameter ($k = 9$) and perform clustering. As before, we tag accounts in each $SD(T_g)_j$ as malicious and benign after identifying cosine similarity. This results in a vector (M) for each account of size n_i where each element (M_j) in M is either 0 or 1 and n_i is the number of *SD*s in a T_g in which the account appears. Here 0 represents not identified as malicious. Let this

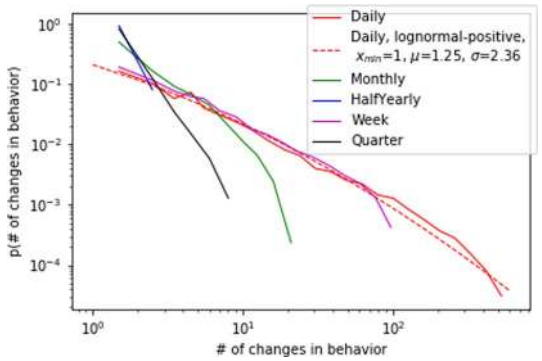


Fig. 13 Probability distribution of number of changes in behavior of accounts with certain probability for being benign at different T_gs

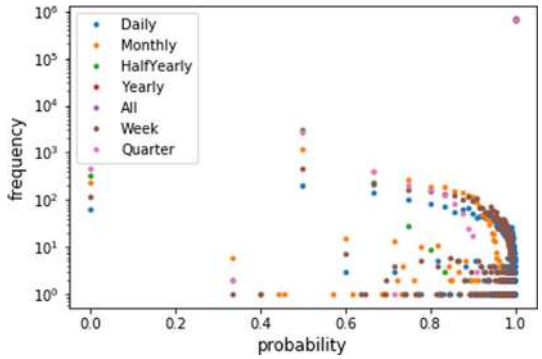


Fig. 14 Number of accounts with certain probability for being benign at different T_gs on a semi-log scale

set of *SD*s be $SD(T_g) = \{SD(T_g)^i_1, SD(T_g)^i_2, \cdots, SD(T_g)^i_j, \cdots, SD(T_g)^i_n\}$. M depicts the behavior of an account i where a change in behavior is captured if $M_j \neq M_{j+1}$. We note that only one benign account, as per our analysis, has changed its behaviour most number of times (591) in the $T_g = Day$. Figure 13 shows probability distribution of the number of behavioral changes observed in an accounts. The figure only considers those accounts where the change happened at least once. For the daily case, as the data was significant we identify that lognormal-positive distribution with parameters $x_{min} = 1$, $\mu = 1.25$, and $\sigma = 2.36$ best fits the data. Further, across all T_gs there were 9254 unique benign accounts that showed unstable behavior.

From M, the probability of a particular account i to be malicious in a given T_g is given by $p^i_m = \frac{\sum_{j \in SD(T_g)^i} M_j}{n_i}$. Number of accounts with certain probability for being benign at different T_gs is shown in Fig. 14. Intuitively, a benign account should have $p^i_m = 0$ while a malicious account should have $p^i_m = 1$. If a known benign account has a high probability then it cannot be trusted. Using such approach, we identify 814 unique accounts across different T_gs as suspects that have $p^i_m = 0$. Further, as seen from the figure, most of the accounts were identified as benign.

In summary, capturing account's behavior changes advocates if the account should be considered as malicious or benign at the time of performing a transaction with it. It is possible that, in past, an account might have evaded detection as malicious.

Knowledge of such changes in the behavior provides an understanding on whether an account is trusted and is not involved in any illegal activity. Our approach provides one solution to know beforehand and predict if an account would behave maliciously.

Conclusion

Blockchains technology and concept has found its implementation not only in the financial sector such as crypto-currency market, hedge-fund, and insurance but also in sectors such as governance, education, healthcare, and law enforcement. Although blockchains are privacy-preserving, with an increase in its adoption, security threats are inevitable. The misuse will be more diverse and deployed using novel techniques. It is essential to have secure transactions. Motivated by the fact that there is limited work in identifying accounts involved in potential malicious activities and those available do not target temporal aspects of blockchains, in this work, we present a way to detect malicious accounts considering the temporal nature of the blockchains.

In this work, we present graph-based temporal features (such as *burst* and *attractiveness*) that are inspired by the existing attacks in the blockchain on top of existing features used to identify malicious accounts. To do so, we first conduct a systematic study of the temporal behavior of the blockchain graph on a collected transaction data in one of the blockchains called Ethereum. Our results show that ExtraTreesClassifier performs best under the supervised setting and achieves balanced accuracy in the range [87.2, 88.7] for different dataset configurations. Moreover, under the unsupervised settings, K-Means was able to cluster max 73.9% known malicious accounts together and identify 554 more suspects that had similar behavior to that of malicious accounts. When considering behavioral changes over time and studying them over different temporal granularities, we are able to detect the probability of an account being malicious at a particular temporal granularity.

Given such results, we expect that benign accounts would be more careful while transacting with suspects and safe-guard themselves from any fraud and security threats. However, the current technique is only applicable to permissionless blockchain. We would like to investigate the applicability of our method to blockchains where features such as *Transaction Fees* and *Balance* are missing. Despite whether a particular blockchain is permissionless or permissioned, there are many other centrality measures such as closeness, betweenness and page-rank that are applicable in a blockchain graph. Another future research direction is to incorporate these measures as features and study the behavior of the accounts before tagging them as malicious or benign. Nonetheless, in this work, we detected suspects using supervised learning and unsupervised learning algorithms. Reinforcement learning is another type of ML that can be applied and studied to detect malicious activity. As our validations failed on the newly tagged malicious accounts one perspective is to study new features and new methods that the new malicious accounts are using and deploying to perform illegal activities. In this work, we considered all the malicious activities under one class. In the future, we would also like to investigate which features are more important for a particular activity and study based on the malicious activity type.

Abbreviations
EOA: Externally Owned Account; CC: Clustering coefficient; SC: Smart Contract; Bal: Balance; ML: Machine Learning; TF: Transaction Fee; AS: Active state; BB: Burst; iD: inDegree; IET: Inter-event time; oD: outDegree; A: Attractiveness; PoW: Proof of work; LOF: Local Outlier Factor; EVM: Ethereum Virtual Machine; MCC: Matthews Correlation Coefficient; RPC: Remote procedure call.

Acknowledgements
Not applicable.

Authors' contributions
RA, SB and SKS designed the research, RA and SB conducted experiments. All authors read and approved the final manuscript.

Funding
This work is partially funded by the National Blockchain Project at IIT Kanpur sponsored by the National Cyber Security Coordinator's office of the Government of India (Grant No. NCSC/CS/2017518) and partially by the C3i Center funding from the Science and Engineering Research Board of the Government of India (Grant No. SERB/CS/2016466).

Availability of data and materials
The list of 2946 malicious accounts used will be made available upon request. Nonetheless, list of all 4708 malicious accounts is publicly available on Etherscan and can be crawled using Etherscan (2020b). The Ethereum transaction Data is also public and can be downloaded using Etherscan APIs (Etherscan 2020a).

Competing interests
The authors declare that they have no competing interests.

Appendix 1: Proof of work

Proof-of-Work (PoW) is a type of consensus algorithm typically used in a Blockchain. Using consensus algorithms, distributed systems can achieve agreement on a single data value. In a lot of crypto-currencies, miners solve a computationally expensive and time consuming problem to add a block to the ledger. Miners use a costly highly specialized computer hardware (ASIC) for mining. They are awarded Ethers as mining reward when their block get confirmed in the ledger. Ethereum uses a *Ethash* PoW algorithm which is prominently designed to be ASIC resistant and supports GPU-based mining. It uses a memory intensive approach rather than a CPU intensive computation.

Nonetheless, PoW consensus algorithms partially provide defense from Sybil attacks because such attacks are computationally expensive and require a lot of computational resources. Therefore, although, an attack is possible, it is computationally very expensive.

Appendix 2: List of features

The Table 4 below list all the features used for our analysis. Note that the values are scaled values. Here, # Features represent the features present in the different data configurations.

Table 4 Sample feature data

tsfresh identified	Feature names (as used in code)	# Features		Value
		59	36	
	indegreeTimeInv	✓	✓	− 0.0086
	outdegreeTimeInv	✓		− 0.0178
	degreeTimeInv	✓		− 0.0128
	numberOfburstTemporalInOut	✓		− 0.0045
	longestBurstTemporalInOut	✓		− 0.1089
	numberOfburstTemporalIn	✓	✓	− 0.0035
	longestBurstTemporalIn	✓	✓	− 0.0375
	numberOfburstTemporalOut	✓	✓	− 0.0079
	longestBurstTemporalOut	✓	✓	− 0.1289
	numberOfburstDegreeInOut	✓		− 0.0077
	longestBurstDegreeInOutAtTime	✓		− 0.0728
	numberOfburstDegreeIn	✓	✓	− 0.0062
	longestBurstDegreeInAtTime	✓	✓	− 0.047
	numberOfburstDegreeOut	✓	✓	− 0.0074
	longestBurstDegreeOutAtTime	✓	✓	− 0.0569
	zeroTransactions	✓	✓	0.0
	totalBal	✓	✓	− 8.6e−5
	transactedFirst	✓	✓	− 0.1942
	transactedLast	✓	✓	− 0.8879
	activeDuration	✓		− 0.3908
	averagePerInBal	✓		− 0.0222
	uniqueIn	✓	✓	− 0.0061
	lastActiveSince	✓	✓	0.0
✓	indegree__index_mass_quantile__q_0.1	✓	✓	0.3977
✓	indegree__energy_ratio_by_chunks_num_segments_10__segment_focus_0	✓		− 0.3957
✓	indegree__linear_trend__attr_"pvalue"	✓	✓	1.1364
✓	ittime__quantile__q_0.7	✓	✓	− 0.2804
✓	ittime__fft_coefficient__coeff_0__attr_"real"	✓	✓	− 0.3908
✓	ittime__median	✓		− 0.2861
✓	outdegree__energy_ratio_by_chunks__num_segments_10__segment_focus_0	✓	✓	− 0.4218
✓	outdegree__enegy_ratio_by_chunks__num_segments_10__segment_focus_1	✓	✓	0.9724
✓	outdegree__fft_coefficient__coeff_0__attr_"real"	✓	✓	− 0.0178
✓	gasPrice__quantile__q_0.2	✓		− 0.3171
✓	gasPrice__quantile__q_0.1	✓	✓	− 0.3036
✓	gasPrice__cwt_coefficients__widths_(2, 5,10, 20)__coeff_0__w_20	✓	✓	− 0.0015
✓	attractiveness__median	✓	✓	0.7886
✓	attractiveness__quantile__q_0.4	✓		0.6248
✓	attractiveness__mean	✓		0.9815
✓	balanceOut__quantile__q_0.1	✓	✓	− 0.0098
✓	balanceOut__quantile__q_0.3	✓		− 0.0118
✓	balanceOut__cwt_coefficients__widths_(2, 5,10, 20)__coeff_0__w_2	✓		− 0.0128
✓	balanceIn__quantile__q_0.4	✓	✓	− 0.0189
✓	balanceIn__cwt_coefficients__widths_(2, 5,10, 20)__coeff_0__w_20	✓	✓	− 0.0231
✓	balanceIn__quantile__q_0.3	✓		− 0.0183
✓	maxInPayment__quantile__q_0.3	✓		− 0.0188

Table 4 (continued)

tsfresh identified	Feature names (as used in code)	# Features		Value
		59	36	
✓	maxInPayment__quantile__q_0.2	✓		− 0.0183
✓	maxInPayment__cwt_coefficients__widths_(2, 5, 10, 20)__coeff_0__w_5	✓	✓	− 0.0173
✓	maxOutPayment__quantile__q_0.6	✓		− 0.0162
✓	maxOutPayment__quantile__q_0.1	✓		− 0.0098
✓	maxOutPayment__cwt_coefficients__widths_(2, 5, 10, 20)__coeff_0__w_2	✓	✓	− 0.0128
	clusteringCoeff	✓	✓	− 0.0807
	burstCount_gasPrice	✓		− 0.0091
	burstCount_balanceIn	✓		− 0.1115
	burstCount_balanceOut	✓		− 0.0073
	burstInstance_indegree	✓	✓	− 0.0108
	burstInstance_outdegree	✓	✓	− 0.0150
	burstInstance_maxInPayment	✓	✓	− 0.1136
	burstInstance_maxOutPayment	✓	✓	− 0.0073
	burstInstance_gasPrice	✓	✓	− 0.0121

Received: 19 June 2020 Accepted: 19 November 2020
Published online: 08 February 2021

References

Abdallah A, Maarof M, Zainal A (2016) Fraud detection system: a survey. J Netw Comput Appl 68:90–113. https://doi.org/10.1016/j.jnca.2016.04.007
Allinvain Theft (2020). https://bitcointalk.org/index.php?topic=83794.0#post_toc_20. Accessed 01/06/2020
Alstott J, Bullmore E, Plenz D (2014) Powerlaw: a python package for analysis of heavy-tailed distributions. PLoS ONE 9(1):85777. https://doi.org/10.1371/journal.pone.0085777
Aspembitova A, Feng L, Melnikov V, Chew L (2019) Fitness preferential attachment as a driving mechanism in bitcoin transaction network. PLoS ONE 14(8):1–20. https://doi.org/10.1371/journal.pone.0219346
Atzei N, Bartoletti M, Cimoli T (2017) A survey of attacks on ethereum smart contracts SoK. In: Proceedings of the 6th international conference on principles of security and trust. Springer, Berlin, pp 164–186. https://doi.org/10.1007/978-3-662-54455-6_8
Bartoletti M, Pes B, Serusi S (2018) Data mining for detecting bitcoin ponzi schemes. In: Crypto valley conference on blockchain technology, Zug, pp 75–84. https://doi.org/10.1109/CVCBT.2018.00014
Bitfly Gmbh (2020) Etherchain—the Ethereum blockchain explorer. https://www.etherchain.org/. Accessed 01/06/2020
Bryk A (2018) Blockchain attack vectors: vulnerabilities of the most secure technology. https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors. Accessed 13 Dec 2019
Buterin V (2013) Ethereum: a next-generation smartcontract and decentralized application platform. https://ethereum.org/whitepaper/
Buterin V (2020) Transaction spam attack: next steps. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/. Accessed 01/06/2020
Chainalysis (2019) 2019 crypto crime report: decoding hacks, darknet markets, and scams. https://go.chainalysis.com/2019-Crypto-Crime-Report.html. Accessed 30 Mar 2020
Chen T, Zhu Y, Li Z, Chen J, Li X, Luo X, Lin X, Zhang X (2018a) Understanding Ethereum via graph analysis. In: IEEE INFOCOM 2018. IEEE, Honolulu, pp 1484–1492. https://doi.org/10.1109/INFOCOM.2018.8486401
Chen W, Zheng Z, Cui J, Ngai E, Zheng P, Zhou Y (2018b) Detecting Ponzi schemes on Ethereum: towards healthier blockchain technology. In: World Wide Web conference, Lyon, pp 1409–1418. https://doi.org/10.1145/3178876.3186046
Chen H, Pendleton M, Njilla L, Xu S (2020) A survey on ethereum systems security: vulnerabilities, attacks and defenses. ACM Comput Surv 53(3):1–43. https://doi.org/10.1145/3391195
Cheng Z, Hou X, Li R, Zhou Y, Luo X, Li J, Ren K (2019) Towards a first step to understand the cryptocurrency stealing attack on Ethereum. In: 22nd international symposium on research in attacks, intrusions and defenses. USENIX, Beijing, pp 47–60. https://www.usenix.org/conference/raid2019/presentation/cheng
Christ M, Kempa-Liehr A, Feindt M (2016) Distributed and parallel time series feature extraction for industrial big data applications. arXiv:1610.07717
Christ M, Braun N, Neuffer J, Kempa-Liehr A (2018) Time series feature extraction on basis of scalable hypothesis tests (tsfresh—A python package). Neurocomputing 307:72–77. https://doi.org/10.1016/j.neucom.2018.03.067
Etherscan (2020a) Ethereum developer APIs. https://etherscan.io/apis. Accessed 01/06/2020
Etherscan (2020b) Label Word Cloud. Accessed 01/06/2020. https://etherscan.io/labelcloud/

Fagiolo G (2007) Clustering in complex directed networks. Phys Rev E 76:026107. https://doi.org/10.1103/PhysRevE.76.026107
Goldsmith D, Grauer K, Shmalo Y (2020) Analyzing hack subnetworks in the bitcoin transaction graph. Appl Netw Sci. https://doi.org/10.1007/s41109-020-00261-7
Jung E, Tilly M, Gehani A, Ge Y (2019) Data mining-based Ethereum fraud detection. In: 2nd international conference on blockchain. IEEE, Atlanta, pp 266–273. https://doi.org/10.1109/Blockchain.2019.00042
Karsai M, Kaski K, Barabási A, Kertész J (2012) Universal features of correlated bursty behaviour. Sci Rep 2(397):1–7. https://doi.org/10.1038/srep00397
Kumar N, Singh A, Handa A, Shukla S (2020) Detecting malicious accounts on the Ethereum blockchain with supervised learning. In: 4th international symposium on cyber security cryptology and machine learning (CSCML 2020). Springer, Be'er Sheva, Israel
Monamo P, Marivate V, Twala B (2016) Unsupervised learning for robust bitcoin fraud detection. In: Information security for South Africa (ISSA). IEEE, Johannesburg, pp 129–134. https://doi.org/10.1109/ISSA.2016.7802939
MyCrypto Inc. (2019) CryptoScamDB. https://cryptoscamdb.org/. Accessed 07/12/2019
Olson R, Bartley N, Urbanowicz R, Moore J (2016) Evaluation of a tree-based pipeline optimization tool for automating data science. In: Genetic and evolutionary computation conference. ACM, Denver, pp 485–492. https://doi.org/10.1145/2908812.2908918
O'Neal S (2020) Bitpoint hack shows that regulators' scrutiny does not equal safety. https://cointelegraph.com/news/bitpoint-hack-shows-that-regulators-scrutiny-does-not-equal-safety. Accessed 01/06/2020
Ostapowicz M, Zbikowski K (2019) Detecting fraudulent accounts on blockchain: a supervised approach. In: Cheng R, Mamoulis N, Sun Y, Huang X (eds) Web information systems engineering. Springer, Hong Kong, pp 18–31. https://doi.org/10.1007/978-3-030-34223-4_2
Palladino S (2020) The parity wallet hack explained. https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/. Accessed 01/06/2020
Pham T, Lee S (2016) Anomaly detection in bitcoin network using unsupervised learning methods. arXiv:1611.03941
Pham T, Lee S (2017) Anomaly detection in the bitcoin system a network perspective. arXiv:1611.03942
Singh A (2019) Anomaly detection in the Ethereum network. Technical report, Indian Institute of Technology, Kanpur
Spagnuolo M, Maggi F, Zanero S (2014) Bitlodine: extracting intelligence from the bitcoin network. In: Christin N, Safavi-Naini R (eds) Proceedings of 18th financial cryptography and data security. Springer, Christ Church, Barbados, pp 457–468. https://doi.org/10.1007/978-3-662-45472-5_29
Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393(6684):440–442. https://doi.org/10.1038/30918
Zola F, Eguimendia M, Bruse J, Urrutia R (2019) Cascading machine learning to attack bitcoin anonymity. In: 2nd international conference on blockchain. IEEE, Atlanta, pp 10–17. https://doi.org/10.1109/Blockchain.2019.00011

Publisher's Note