ETH-TT: A Novel Approach for Detecting Ethereum Malicious Accounts

Ziang Yuan[1], Tan Yang[1(✉)], and Jiantong Cao[2]

[1] State Key Laboratory of Networking and Switching Technology, School of Computer Science (National Pilot Software Engineering School), Beijing, China
tyang@bupt.edu.cn

[2] School of Economics and Management, Beijing University of Posts and Telecommunications, Beijing, China
tony000@bupt.edu.cn

Abstract. Ethereum, as the second generation of blockchain technology, it not only brings many advantages, but also spawns various malicious incidents. Ethereum's anonymity makes it a hotbed of cybercrime, causing huge losses to users and severely disrupting the Ethereum ecosystem. To this end, this paper proposes a method for detecting malicious accounts in Ethereum based on ETH tracking tree (ETH-TT). Firstly, based on the transaction history replay mechanism, an ETH tracking algorithm for tracking the transaction amount of Ethereum is designed to obtain the ETH tracking tree, and extract sequence features from it. Then train the LSTM model to reduce the dimension of the sequence features to obtain the output features. Finally, detection is done by a machine learning classifier, fused with manual features from account transaction history. We uses 5576 malicious accounts and 4968 normal accounts as dataset for experiments. The results show that the ETH-TT method can achieve an F1-score of 95.4% with the cooperation of the XGBoost classifier, which is better than the detection method using only manual features.

Keywords: Blockchain · Ethereum · Malicious accounts detection · ETH tracking tree · LSTM · Machine learning

1 Introduction

As a representative of the blockchain 2.0, Ethereum provides a complete Turing scripting language, allowing users to deploy decentralized applications (DAPP) on Ethereum to implement user-defined methods, bringing more possibilities to the blockchain [9,15]. Based on the account model, Ethereum supports direct transactions between accounts.

While Ethereum brings more functions to users, it also brings more vulnerabilities and threats, and due to the design and implementation of Ethereum,

L. Fang et al. (Eds.): CICAI 2022, LNAI 13605, pp. 84–94, 2022.
https://doi.org/10.1007/978-3-031-20500-2_7

the vulnerabilities caused are more difficult to deal with than in the traditional mode [16]. According to the research of Chen [1], Ethereum faces 44 types of vulnerabilities and 26 types of attacks. The hackers use certain technical means to complete malicious events through accounts on the Ethereum platform, such accounts for malicious behavior are defined as Ethereum malicious account. The following are some of the major events that have occurred in recent years. In June 2016, the famous crowdfunding project The Dao [7] was stolen about US$60 million by attackers using the reentrant vulnerability in the smart contract. In July 2017, a security vulnerability occurred in Parity1.5[1] and above, 150,000 ETH were stolen, worth US$30 million. Upbit [14], one of the largest cryptocurrency exchanges in South Korea, was stolen by an unidentified attacker in November 2019. 342,000 ETH were sent to an Ethereum wallet and quickly spread to about 800 accounts. Therefore, there is an urgent need to use computer technologies such as pattern recognition, machine learning, or neural networks to detect malicious behavior on the Ethereum platform, and further detect malicious accounts in Ethereum [18]. Thereby reducing risks and bringing more security.

In the field of Ethereum malicious account detection, the current research is mainly based on the manual features extracted from the transaction history of the address, and then the machine learning classifier is constructed for training, such as the study of Kumar [8], Poursafaei [11], Farrugia [3], etc. In some papers, it also shows good F-score and Recall. However, the information contained in the manual features used in these studies is incomplete. According to our analysis, malicious accounts often quickly transfer "Black Money" ETH to multiple accounts after conducting malicious acts. This makes it difficult for official institutions to recover them to achieve money laundering purposes. Take Upbit [14] as an example. In a few days, the hacker transferred the stolen 342,000 ETH to more than 800 accounts, which were distributed in more than ten transaction levels [14]. In the end, most of the ETH flowed into the exchange to withdraw money to complete the money laundering work. There are also malicious events such as Cryptopia and Bitpoint [13], which are also similar to the above-mentioned behavior patterns. Therefore, analyzing the transfer behavior and further tracking the flow of malicious transaction ETH between accounts is of great significance for detecting malicious accounts.

Inspired by this, we propose an ETH Tracking Tree (ETH-TT) based method for detecting malicious accounts in Ethereum, and achieves good results in the final evaluation, which is better than the method that only using manual features.

2 ETH Tracking Tree Method

The whole flow of ETH-TT method proposed is shown in Fig. 1. First, determine the parameters of the limits, including the tracking time and the number of tracking tree layers. Then use the key transaction algorithm and the ETH tracking algorithm to obtain the ETH tracking tree, and further calculate the

[1] Multi-signature wallet version 1.5 of Parity Technologies.

traceability rate information. Next, the sequence features are parsed from the tracking tree and feed into the LSTM model to get the output vector features. Finally, the vector features and manual features are connected and input into the machine learning classifier to get the final result.

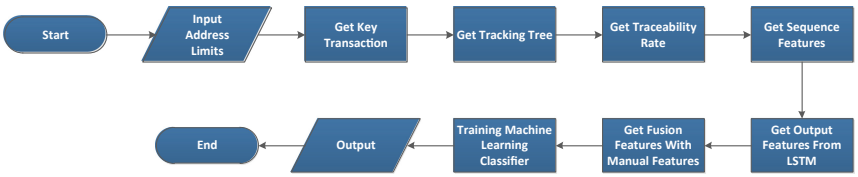


Fig. 1. ETH-TT method flow chart

2.1 ETH Tracking Tree and Traceability Rate

Before describing how to building the ETH tracking tree, we need to find the key transaction corresponding to the account and let it be the starting transaction to initiate the tracking. Specifically, we need to sort the transaction list of addresses and find the most representative transaction as the key transaction. In the research of Wu [16], they proposed that the key characteristics of a transaction are its transaction amount and timestamp. We use his formula to sort the transactions, as shown in Eq. 1.

$$P_t^{0.5} * P_a^{0.5} \tag{1}$$

Among them, P_a represents the ratio of the one transaction's ETH to the total transactions' ETH of the address, and P_t represents the position of the one transaction's timestamp in the entire transaction history time range of the address. The index is used to balance the influence of two parameters, and 0.5 is the best value [16].

The key transaction algorithm is shown in Algorithm 1. After entering an address, find all transactions for the address and sort them in reverse order according to Eq. 1. It then traverses the sorted transaction list until it finds a transaction that meets the traceability conditions, and then returns it as a key transaction. A transaction can be traced, that is, the ETH generated by the transaction has been transferred later.

The ETH tracking tree is constructed by the ETH tracking algorithm, as shown in the Algorithm 2. The algorithm is based on the transaction history replay mechanism, and first takes the ETH of the key transaction as the target ETH. Under some constraints such as tracking time and number of tracking tree layers, track the flow of the target ETH between the accounts to form a corresponding tree structure, called ETH tracking tree. Finally, calculate the traceability rate of the target ETH in the tree's nodes. The following describes the algorithm in detail in two steps.

Algorithm 1. Key Transaction Algorithm

Input: Target Address A
Output: Key Transaction of Address A
1: $Txlist$=A.getTxList().sort(key:$P_a^{0.5}$*$P_t^{0.5}$,reverse:True)
2: **for** each $tx \in Txlist$ **do**
3: **if** $canBeTraced(tx)$ **then** return tx
4: **end if**
5: **end for**

Algorithm 2. ETH Tracking Algorithm

Input: KeyTransaction Tx, Layer Limit $Layers$, Time Limit $Days$
Output: ETH Tracking Tree $Tree$
1: $Tree$=Init()
2: $PathTx$=Init()
3: $Datelist$=getDate(Tx,$Days$)
4: $Tree$.root=Tx.toAddr
5: **for** each $date \in Datelist$ **do**
6: $Txlist$=getTxList($date$).sort(key:timeStamp,blockNumber)
7: **for** each $tx \in Txlist$ **do**
8: **if** $isEligible(tx, Layers)$ *and* $tx.fromAddr \in Tree$ **then** appendToPathTx(tx, $PathTx$)
9: **if** $tx.toAddr \notin Tree$ **then** appendToTree($tx, tx.toAddr, Tree$)
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **for** $tx \in PathTx$ **do** calculateTraceabilityRate(tx,$Tree$)
15: **end for**

The first part of the ETH tracking algorithm is to create the structure of the tree. Specifically, the first step is to determine the number of tracking tree layers and tracking time as the limits of the tracking algorithm. The second step is to calculate the time range according to the key transaction and time limit, and find the corresponding Ethereum transaction list based on this, and finally sort by the timestamp and transaction index (block number), that is, the actual transaction order. The third step is to add the output address of the key transaction to the tracking tree as the root node. The fourth step is to traverse the sorted transaction list after the key transaction, find the output transactions of the root node, insert the output addresses of these transactions into the tree according to the transaction order. For example, address A is already in the tree, and has transactions with address B and address C successively. Then address B and address C are regarded as the child nodes of A. Loop to find all the tree nodes' output transactions and child nodes, until find the exchange addresses, contract addresses, or reach the limit of the number of tracking tree layers. Thus ETH tracking tree is established.

In the tracking algorithm, the concept of traceability rate will be involved. The traceability rate of an account at a certain timestamp indicates the proportion of the target ETH held in its balance to the total target ETH needs to be tracked, that is, the distribution of target ETH among accounts. Which can visually show the information of the flow of ETH. When a transfer transaction occurs in an account with a traceability rate, the traceability rate will be converted accordingly. The formula for traceability rate conversion is shown in Eq. 2. Among them, $Balance_{from}$ represents the balance of the account before the transaction. $Amount$ is the ETH amount of the transaction. $Rate_{from}$ represents the traceability rate of the account before the transaction. Finally, the traceability rate conversion amount can be obtained.

$$Conversion_{rate} = (Amount/(Balance_{from})) * Rate_{from} \tag{2}$$

After describing the concept of traceability rate and traceability rate conversion, we can carry out the remaining process of the tracking algorithm. The second part of the algorithm is to calculate the traceability rate related information of the nodes in the ETH tracking tree. Specifically, after getting the tracking tree, traverse the list of transactions encountered during the tracking process, where each transaction represents a traceability rate conversion process as described above, so the traceability rate of the node is constantly changing. By calculating each traceability rate conversion, some information related to the traceability rate are calculated, i.e., the final traceability rate of each node and the maximum traceability rate reached, and the position of the point in time when the maximum traceability rate is reached. Then we maintain these parameters on the tracking tree nodes for later calculation of sequence features.

Through the above two steps of the ETH tracking algorithm, a simple example of the finally obtained ETH tracking tree is shown in Fig. 2.

2.2 Features Extraction

Generally speaking, features extraction is divided into two parts.

- Firstly we could find the transaction history of target address, and then extract features from it. In this part, we refer to the research of Kumar [8] to extract 45 features. The features include the number of Input/Output transactions, the average handling charge, the Input/Output amount, etc.
- The second part is to use the ETH tracking tree to extract sequence features to obtain the information of the ETH flow. As shown in Fig. 3, each level of the sequence corresponds to each level in the tree. After getting the tracking tree, we first calculate the features of each node in the tree according to the traceability rate information obtained in the Algorithm 2 and some auxiliary information. See the Table 1 for details. The evaluation indicators of these features (including arithmetic mean, variance, standard deviation, mean square

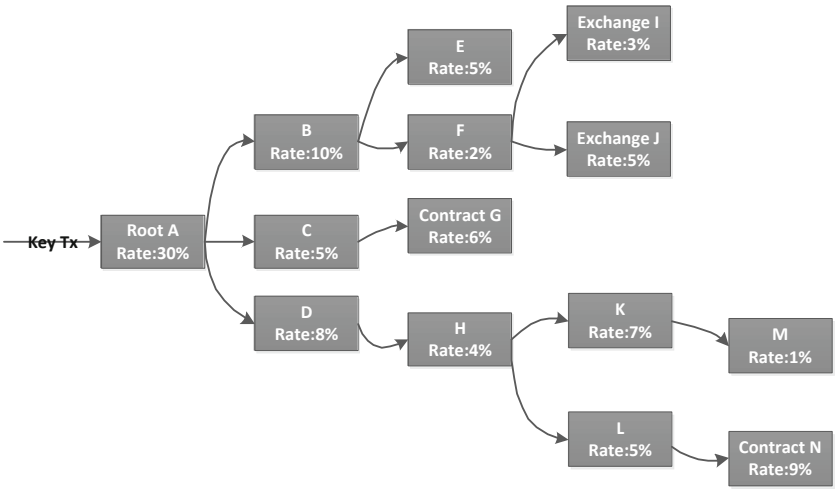


Fig. 2. ETH tracking tree

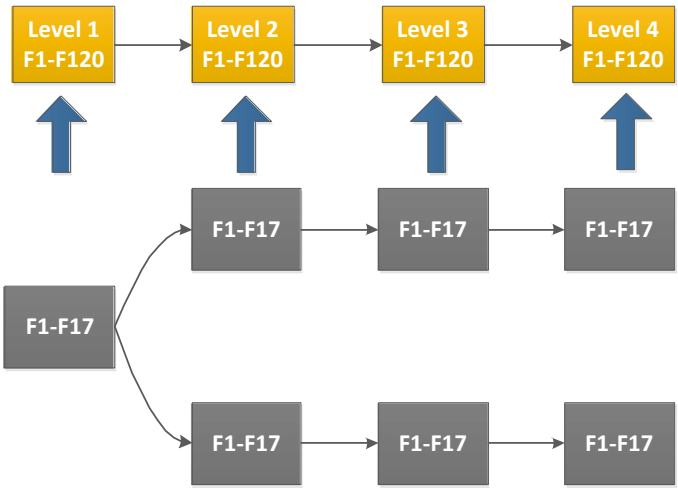


Fig. 3. Extracting sequence features from ETH tracking tree

root value, majority, median, geometric average) are then taken by layer, and the number of nodes in that layer is added as the last feature, with a total of 120 dimensions as the features of each layer. Finally the sequence features of M * N are obtained, where M represents the number of layers of the tracking tree, N stands for a fixed 120-dimensional feature.

Table 1. Tracking tree node features

Features	Descriptions
Input_tx_num	The number of transactions received
Output_tx_num	The number of transactions sent
Tx_num	The total number of transactions
Addr_num	The number of distinct addresses where transactions took place
Max_rate	The maximum of traceability rate ever achieved
Max_rate_position	The position of the time point to reach the maximum traceability rate in the whole transaction sequence
Tx_mean_interval	The average transaction time interval
Rate_mean_change	The average conversion of traceability rate
Rate_max_change	The maximum conversion of traceability rate
Rate_min_change	The minimum conversion of traceability rate
Max_tx_num	The maximum number of transactions with the same address
Max_value_num	The maximum ETH traded
Rate	The final traceability rate
Is_exchange	1 if the address is an exchange address else 0
Exchange_rate	The final traceability rate if the address is an exchange address else 0
Is_contract	1 if the address is an contract address else 0
Contract_rate	The final traceability rate if the address is an contract address else 0

2.3 Model Design

We need to extract sequence features into fixed-dimensional features to train machine learning classifiers. Considering that there is a dependency relationship between the levels of the tracking tree, and the obtained features are long-range sequences, we use Long Short Term Memory (LSTM) as the features extraction model [4,5,17].

The overall design of the model is shown in Fig. 4. We construct a bidirectional multi-layer LSTM, and feed the sequence features (which may need to be filled) into LSTM for training and getting output features. The length of LSTM sequence is determined by the restrictions on the number of tracking tree layers set in the experiment. After the output features is obtained, it is connected with the manual features of the address. Then the fusion features are feed into machine learning classifiers for training, and the final results can are obtained. In addition, the manual features will be directly feed into the machine learning classifier for comparison.

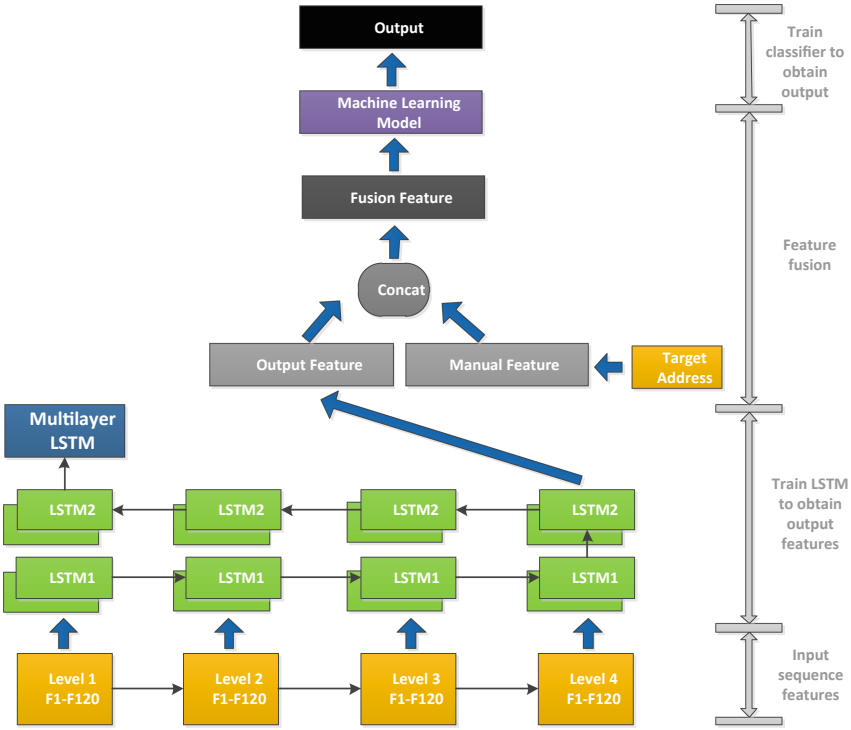


Fig. 4. Overall design of the model

3 Experiment

3.1 Dataset

First, we built a Geth Ethereum node on the server and started to synchronize the block information. Then we use the Ethereum ETL project [10] to store transaction information on the hard disk in the form of a CSV file on a daily basis.

– We need to collect malicious addresses (positive samples) and normal addresses (negative samples) of EOA. Firstly, we collected 5585 positive samples from the tag library of Etherscan [12] and Cryptoscamdb [2], GitHub [6], third-party certification bodies, etc. Including fraud, phishing, money laundering, currency theft and other types. In order to collect negative samples, We continuously randomly took addresses from senders and receivers in the transaction list and put them into the positive sample set (automatic deduplication), and cycled this process until the set size reaches 6000.

– The second step we need to take is preprocessing. First, all sample addresses were traversed, if there were no transactions, they were filtered out. Took advantage of the "create" type transaction feature of smart contract to filtered it out. Then traversed the negative samples and filtered out the ones that have direct transaction relationship with the positive samples. Finally, 5576 positive samples and 4968 negative samples were left as the final dataset.

3.2 Experimental Details

The entire control experiment is divided into two modules, one uses the ETH-TT method, and the other uses only manual features to train the model with the same kind of features as Kumar [8].

In the ETH-TT experiment, we specified four parameter combinations as control experiments to limit the tracking time (number of days) and the number of tracking tree layers, respectively [3 days, 5 layers], [3 days, 15 layers], [5 days, 15 layers], [5 days, 20 layers]. Finally, it is trained in 4 machine learning classifiers: XGBoost, Random Forest, LightGBM and Decision Tree. It should be noted that we randomly divide the entire dataset into a training-set and a test-set, with a ratio of 7:3.

3.3 Experimental Result

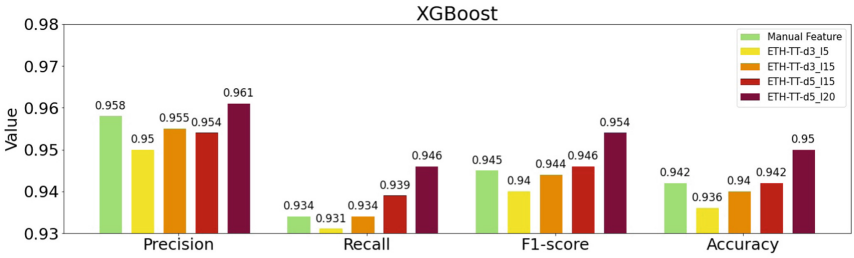
The evaluation results of the above experiments are shown in the Fig. 5. Each sub-graph in the Fig. 5 shows the Manual Feature method and the ETH-TT method under different parameter combinations. The indicators are Precision, Recall, F1-score, Accuracy.

F1-score can comprehensively reflect the effect of the model. Under the parameters of [5 days, 20 layers], the XGBoost model reached the optimal 95.42%. Recall can measure the ability of the model to find positive samples, XGBoost reached the best 94.69%.

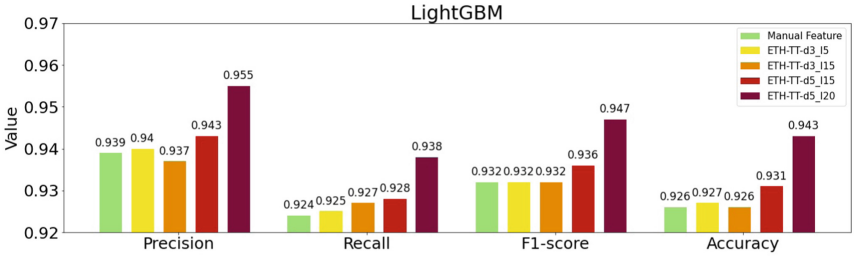
3.4 Result Analysis

The results show that the ETH-TT method is effective for detecting malicious Ethereum accounts, and is better than Kumar [8], that is, the method that only uses manual features. After reaching the parameter combination of [5 days, 20 layers], all indicators are better than the method of Kumar [8] under different machine learning classifiers.

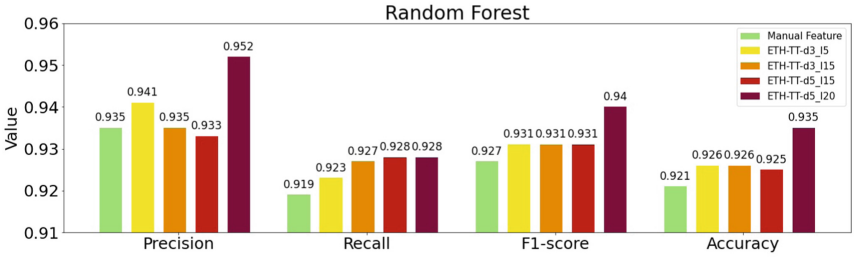
From the perspective of parameter combinations, with tracking time and the number of tracking tree layers are relaxed, the overall effect of the model is getting better and better. This shows that with the expansion of the ETH tracking tree, the contribution of the features extracted from it to the model is magnifying, which further illustrates the effectiveness of the ETH-TT method.



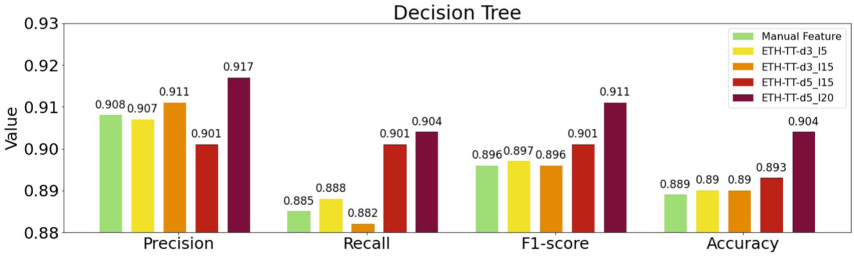
(a) Evaluation results under XGBoost model



(b) Evaluation results under LightGBM model



(c) Evaluation results under Random Forest model



(d) Evaluation results under Decision Tree model

Fig. 5. Evaluation results

4 Availability of Data and Material

The dataset and model files mentioned in the article are available here: https://github.com/yfzjay/ETH-TT.

Acknowledgments. This work was supported by the National Key Research and Development Program of China under grant No. 2019YFC1521101.

References

1. Chen, H., Pendleton, M., Njilla, L., Xu, S.: A survey on Ethereum systems security: vulnerabilities, attacks, and defenses. ACM Comput. Surv. (CSUR) **53**(3), 1–43 (2020)
2. Cryptoscamdb (2019). https://api.cryptoscamdb.org
3. Farrugia, S., Ellul, J., Azzopardi, G.: Detection of illicit accounts over the Ethereum blockchain. Expert Syst. Appl. **150**, 113318 (2020)
4. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
6. IRMkz: Ethereum addresses-darklist. https://github.com/MyEtherWallet/ethereum-lists/blob/master/src/addresses/addresses-darklist.json (2020)
7. Jentzsch, C.: A standard decentralized autonomous organization (DAO) framework written in solidity to run on the Ethereum blockchain (2016). https://github.com/slockit/DAO/tree/v1.0
8. Kumar, N., Singh, A., Handa, A., Shukla, S.K.: Detecting malicious accounts on the Ethereum blockchain with supervised learning. In: Dolev, S., Kolesnikov, V., Lodha, S., Weiss, G. (eds.) CSCML 2020. LNCS, vol. 12161, pp. 94–109. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49785-9_7
9. Lee, X.T., Khan, A., Sen Gupta, S., Ong, Y.H., Liu, X.: Measurements, analyses, and insights on the entire Ethereum blockchain network. In: Proceedings of The Web Conference 2020, pp. 155–166 (2020)
10. Medvedev, E.: blockchain-etl/ethereum-etl (2020). https://github.com/blockchain-etl/ethereum-etl
11. Poursafaei, F., Hamad, G.B., Zilic, Z.: Detecting malicious Ethereum entities via application of machine learning classification. In: 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), pp. 120–127. IEEE (2020)
12. Tan, M.: Ethereum blockchain explore (2015)r. https://etherscan.io
13. Tan, M.: Heist accounts (2015). https://etherscan.io/accounts/label/heist
14. Tan, M.: Upbit hack (2015). https://etherscan.io/accounts/label/upbit-hack
15. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**(2014), 1–32 (2014)
16. Wu, J., et al.: Who are the phishers? Phishing scam detection on Ethereum via network embedding. arxiv prepr. arxiv:1911.09259 (2019)
17. Xingjian, S., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.: Convolutional LSTM network: a machine learning approach for precipitation nowcasting. In: Advances in Neural Information Processing Systems, pp. 802–810 (2015)
18. Xu, J.J.: Are blockchains immune to all malicious attacks? Financ. Innov. **2**(1), 1–9 (2016)