Detecting Malicious Accounts on the Ethereum Blockchain with Supervised Learning

Nitesh Kumar, Ajay Singh, Anand Handa(✉), and Sandeep Kumar Shukla

C3i Center, Department of CSE, Indian Institute of Technology, Kanpur, India
{niteshkr,ajay,ahanda,sandeeps}@cse.iitk.ac.in

Abstract. Ethereum is a blockchain platform where users can transact cryptocurrency as well as build and deploy decentralized applications using smart contracts. The participants in the Ethereum platform are 'pseudo-anonymous' and same user can have multiple accounts under multiple cryptographic identities. As a result, detecting malicious users engaged in fraudulent activities as well as attribution are quite difficult. In the recent past, multiple such activities came to light. In the famous Ethereum DAO attack, hackers exploited bug in smart contracts stole large amount of cryptocurrency using fraudulent transactions. However, activities such as ponzi-scheme, tax evasion by transacting in cryptocurrency, using pseudo-anonymous accounts for receiving ransom payment, consolidation of funds accumulated under multiple identities etc. should be monitored and detected in order to keep legitimate users safe on the platform. In this work, we detect malicious nodes by using supervised machine learning based anomaly detection in the transactional behavior of the accounts. Depending on the two prevalent account types – Externally Owned Account (EOA) and smart contract accounts, we apply two distinct machine learning models. Our models achieve a detection accuracy of 96.54% with 0.92% false-positive ratio and 96.82% with 0.78% false-positive ratio for EOA and smart contract account analysis, respectively. We also find the listing of 85 new malicious EOA and 1 smart contract addresses between 20 January 2020 and 24 February 2020. We evaluate our model on these, and the accuracy of that evaluation is 96.21% with 3% false positive.

Keywords: Ethereum blockchain · Malicious accounts · Machine learning · Anomaly detection · Feature extraction

1 Introduction

In the last decade, blockchain has emerged as an innovative technology platform for a variety of cryptocurrency as well as other applications. The Bitcoin

This research is partially funded by the Office of the National Cyber Security Coordinator, Government of India.

S. Dolev et al. (Eds.): CSCML 2020, LNCS 12161, pp. 94–109, 2020.
https://doi.org/10.1007/978-3-030-49785-9_7

cryptocurrency ecosystem [12] is built on the blockchain technology. Transactions between participants of a blockchain platform are verified and agreed on through a distributed consensus mechanism which obviates the need for a centralized authority. While cryptocurrency was the first demonstrated application of blockchain technology, due to the fact that blockchain enables tamper resistant property to the history of transactions using cryptographic hashing, and it enables authentication of transactions through public key cryptography, it has proven itself to be a potential technology for building trusted interaction platform between multiple participants involved in mutual transactions without having to trust any individual participant. Bitcoin, Ethereum, Monero etc., are blockchain based cryptocurrency platforms for financial transactions and also offers pseudo-anonymity to users. This has also given rise to a lot of malicious activities on these platforms which makes it unsafe for legitimate users on these platforms. Therefore, automated detection of users who might be engaging in malicious activities is of utmost importance.

The pseudo-anonymity of participants led the hackers and money launderers to be part of the network without any fear attribution. However, since pseudo-anonymity does not provide guaranteed anonymity, researchers have been engaged in deducing pattern of transactions that could then be matched against fraudulent transaction patterns. It is worth noting cryptocurrencies are still illegal in some countries as the cryptocurrencies are generated in these platforms without any connection to the central banking system in the countries, leading to tax evasion, illegal transactions, ransom payments etc. Soon after the inception of the bitcoin, Online underworld marketplaces like Silk Road emerged for selling contraband drugs and other illegitimate items. A vulnerability in the Parity multi-signature wallet on the Ethereum network resulted in a loss of 31 million US Dollars in a few minutes. If some benevolent hackers had not stopped the ongoing exploitation, it might have resulted in a loss of 180 million US Dollars [1].

It is therefore, our focus in this work to find irregularities and the fraudulent transactional behaviors in the Ethereum network. We investigate the past Ethereum transaction data from its genesis till a certain date (Ethereum being a public blockchain, one can download the entire data) in search of abnormal activities. We extract relevant information to train machine learning models for anomaly detection. The main contributions of this work are as follows:

- We collect the malicous Ethereum addresses of various attack types like phish-hack, cryptopia-hack, etc. from multiple sources and filter them to obtain relevant addresses. We also label non-malicious addresses after data preprocessing.
- We extract features from the transactions and use feature engineering to find the relevant features for classification.
- We detect the malicious nodes in the Ethereum network with a good accuracy.
- We evaluate our model on newly collected 85 malicious EOA and 1 smart contract addresses between 20 January 2020 and 24 February 2020. The model achieves a good evaluation accuracy.

The rest of the paper is organized as follows: Sect. 2 briefly describes the Ethereum blockchain. Section 3 describes relevant related work. Section 4 discusses the proposed methodology. Section 5 describes evaluation results. Section 7 concludes the work.

2 Background

Vitalik Buterin developed Ethereum [6] in 2013. It is a step forward in the blockchain technology which brought advances over the Bitcoin blockchain technology by introducing a programming language which is Turing-complete, and providing a program execution platform in the blockchain. The programs that run on Ethereum are called smart contracts. One can build complex decentralized applications using smart contracts. The cryptocurrency of Ethereum is called Ether, which fuels the Ethereum network. Ethereum Virtual Machine (EVM) is the computing infrastructure for Ethereum nodes. Currently the main consensus mechanism used by Ethereum blockchain is Proof of Work (POW), but Ethereum announced that it will switch to Proof of Stake (POS). The reason is that that Proof of work is a computationally-intensive process and consumes an enormous amount of energy.

2.1 Ethereum Accounts

Ethereum has two types of accounts which participate in transactions on the platform. Figure 1 shows how these accounts interact with each other.

1. **Externally Owned Account (EOA):** The end-users create EOAs to become participants in the Ethereum network. Participants generate private key for each account to digital sign transactions. An externally controlled account may have a non-zero Ether balance, and can perform transactions with other EOAs and contracts.
2. **Contract/Smart Contracts:** These are the self-executing code which can be invoked by EOAs or by another contract as an internal transaction. A contract also may have an Ether balance and an associated code which performs arbitrary complex operations on execution.

2.2 Ethereum Transactions

There are three types of transactions in the Ethreum network and they are as follows:

– **Fund Transfer Between EOAs:** In this type of transaction, one EOA transfers funds to another EOA as shown in Fig. 2.

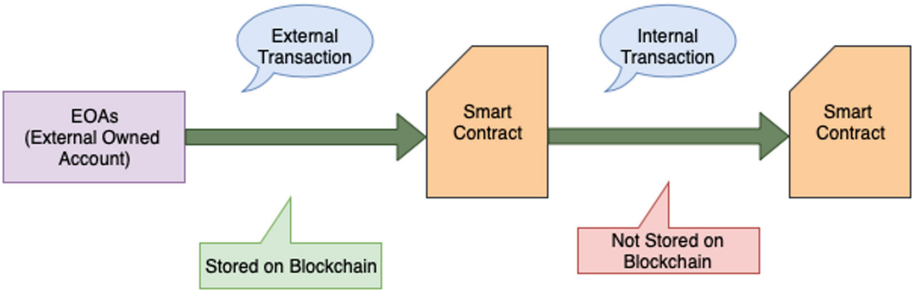

Fig. 1. Account interaction

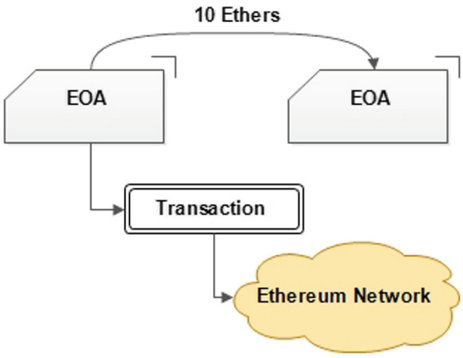

Fig. 2. Fund transfer between EOA

– **Deploy a Contract on the Ethereum Platform:** In this type of transaction, EOA deploys a contract using a transaction on the Ethereum network, as shown in Fig. 3.

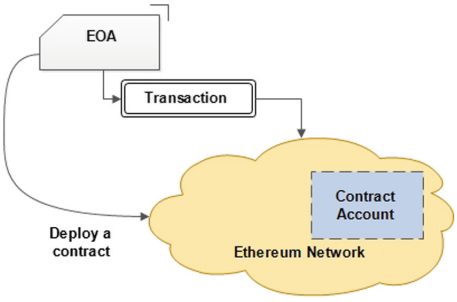

Fig. 3. Deploy a contract on ethereum network

– **Execute a Function on a Deployed Contract:** In this type of transaction, Ethereum sends a transaction to execute functions defined in a contract. The transaction gets performed after the contract deployment, and Fig. 4 shows such a transaction.

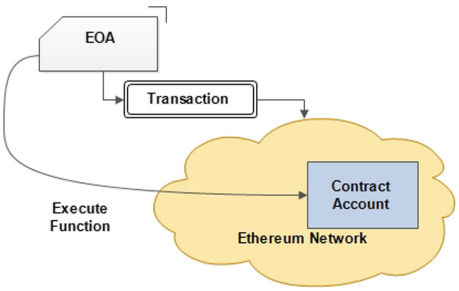

Fig. 4. Execute a function on a deployed contract

2.3 Ethereum Transaction Structure

An Ethereum Transaction record as it is formed and eventually persisted on the blockchain has a number of fields.

1. **From:** This field contains the transaction sender's address. The length of this field is 20 bytes. An address is a hash of a public key associated with the account.
2. **To:** This field has the address of the receiver of the transaction. The length of this field is 20 bytes. This field can be the address of either an EOA or a contract account or empty, depending on the type of transaction.
3. **Value:** This field has the amount in terms of wei (1 ether = 10^{18} weis) transferred in the transaction.
4. **Data/Input:** In case of contract deployment, this field contains the bytecode and the encoded arguments and is empty when there is a fund transfer.
5. **Gas Price and Gas Limit:** Gas price is the amount (in terms of wei) for each gas unit related to the processing cost of any transaction which a sender is willing to pay. In a transaction, the maximum gas units that can be spent is the gas limit. The gas limit ensures that there is no infinite loop in a smart contract execution.
6. **Timestamp:** It is the time at which the block is published or mined. Below is an example of an Ethereum transaction structure.

```
{"status":"1",
"message":"OK",
"result":
{"blockNumber":"6026742",
"timeStamp":"1532511199",
"hash":"0x94917b89296051b066db2ac572987d...",
"nonce":"2560067",
"blockHash":"0xad77b360c7a8401ea81e875a8fbc9...",
"transactionIndex":"8",
"from":"0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be",
"to":"0x0a0ba956038d4a66002d612648332b9c4ab7646c",
"value":"500000000000000000",
"gas":"21000",
"gasPrice":"60000000000",
"isError":"0",
"txreceipt_status":"1",
"input":"0x",
"contractAddress":"",
"cumulativeGasUsed":"227318",
"gasUsed":"21000",
"confirmations":"3212860"
}
}
```


3 Related Work

In this section, we discuss some existing work related to the anomaly detection in blockchain, more specifically to Bitcoin and Ethereum blockchain. BitIodine is a framework to deanonymize the users [16] and is used to extract intelligence from the Bitcoin network. It labels the addresses automatically or semi-automatically using the information fetched from web scrapping. The labels used for addresses are gambling, exchanges, wallets, donations, scammer, disposable, miner, malware, FBI, killer, Silk Road, shareholder, etc. BitIodine first parses the transaction data from the Bitcoin blockchain. Then it performs clustering based on user interaction and labels the clusters and users. Their objective is to label every address in the network into one of the mentioned categories. Also, they detect some of the anomalous addresses in the network by tracing their transactions. The authors verify their system performance on some of the known theft and frauds in the Bitcoin platform. BitIodine detects addresses that belong to Silk Road cold wallet, CryptoLocker ransomware. The proposed modular structure is also applicable to other blockchains. However, BitIodine does not use any machine learning techniques.

In [17], the authors propose a Graph-based forensic investigation of Bitcoin transactions and perform analysis on Bitcoin transaction data and evaluate the network data. They use 34,839,029 Bitcoin transactions and 35,770,360 distinct

addresses. The objective is to detect money theft, fraudulent transactions, and illegal payments made to the black market. The proposed framework retrieves all the transaction details of a given address. The proposed framework does not attempt to detect the anomalous addresses in the network, but it provides detailed information on addresses. They use clustering to group users together and multiple graph-based techniques to analyze the money flow within the network. They analyze the flow of money using algorithms like Breadth-First Search (BFS) algorithm, edge-convergent pattern, and the existence of cycles in the network to detect any money laundering activity.

Thai T. Pham et al. [13,14] propose an anomaly detection method in the Bitcoin network using the unsupervised learning classifiers like K-means clustering, Mahalanobis distance, and Support Vector Machine (SVM). The aim is to detect the suspicious transactions that take place within the network and mark the users based on these transactions. They use user graph and transaction graph as the underlying space on which clustering are performed based on a 6 features of each node in the user graph, and 3 features in the transaction graphs. They also ran into computational difficulty and had to limit their study to a limited number of nodes.

Xiapu Luo et al. [11] perform a graph-analysis of the Ethereum network. They claim to be the first to perform a graph-based analysis of Ethereum blockchain. The model constructs three different graphs to analyze money transfer, smart contract creation, and smart contract invocation. The size of the dataset is – 28,502,131 external transactions and 19,759,821 internal transactions. After analyzing the graph, they have given the following preliminary insights – the participants use the Ethereum network more than smart contracts for money transfer. The insights made by them is pretty obvious as the number of transactions done by a regular user is not comparable to a huge number of transactions performed in exchanges. Every user does not know the Solidity or Golang to deploy their contracts. Hence, only a few of them can deploy the contract and use it. All participants have different requirements for which they interact with the Ethereum network, so they have the same behavior.

Although some of the above approaches try to find an anomaly in the Bitcoin network, but none of them has a sophisticated method for anomaly detection. Like BitIodine [16], the authors attempt to detect paths by searching the network manually, but the proposed method does not have an automated mechanism to detect malicious addresses. Although in [13], the authors use machine learning techniques for anomaly detection, the reported accuracy is not very good. Therefore there is a need for an automated and efficient mechanism for anomalous addresses detection in any blockchain network with high accuracy.

4 Proposed Methodology

In the Ethereum network, addresses which try to carry out tasks for which they are not authorized or addresses that attempt to execute the fraudulent transactions are suspicious addresses. We call their behavior as anomalous. In

this work, we focus on the past Ethereum transactions to detect the anomaly in behavior/actions by addresses. We train supervised machine learning models using features we extract from the transactions performed by the addresses on the Ethereum network. We mark the addresses as malicious and non-malicious after the classification by the trained model. We train two models for a different account types of the Ethereum platform – EOA and smart contract accounts because both accounts have distinct characteristics and behavior. Our anomaly detection method performs the following steps:

1. Collection of already publicly available malicious and non-malicious addresses from various repositories.
2. Collection of transactions executed by all such addresses in the past.
3. Data preprocessing, feature extraction, training and evaluation for:
   - EOA Analysis
   - Smart contract account analysis.

4.1 Collection of Malicious and Non-malicious Addresses

We use supervised machine learning classification methods to detect malicious and non-malicious addresses in the Ethereum network. Therefore, we collect the labeled malicious and non-malicious addresses from various sources. We collect malicious addresses from the sources, namely etherscan [7], cryptoscamdb [5], and few addresses from a GitHub repository [9]. Malicious addresses are publicly listed based on different kinds of attack such as a heist, cryptopia-hack, Upbit-hack, phish-hack, etc., that these addresses have carried out in the past. These attack types are the same as the ones used by etherscan label word cloud [8]. We fetch non-malicious addresses from the same sources cryptoscamdb and etherscan [4]. Initially, we collect a total of 6,154 malicious addresses and 0.1 million non-malicious addresses.

4.2 Collection of Transactions for a Given Address

In this step, we extract all the transactions performed by all malicious and non-malicious addresses from the Ethereum Blockchain data that we had previously collected. Transactions contain various fields such as address of the sender, address of the receiver, timestamp, gas value used for the transaction, the gas limit for the transaction, transaction hash, block number, etc. Algorithm 1 shows an approach to extract the transactions from a given address. We collect all the transactions using the etherscan API and save the transactions in a JSON file for further processing.

Algorithm 1. Algorithm for Extraction of Transactions for a given address

Input: *Address* `// List of Ethereum account addresses (Malicious/Non-Malicious)`

Output: *Txns* `// List of transactions for a given address`

foreach *address* ∈ *Address* **do**
| *Txns* ⟵ `ExtractTxn(`*address*`)`
end

Function `ExtractTxn(`*address*`):`
| *F* ⟵ `'curl -X GET http://api.etherscan.io/api?module=account&action=txlist&address=`*address*`&sort=asc&apikey=`*ApiKeyToken*`'`
| **return** *F*;
End Function

4.3 Data Preprocessing

In data preprocessing, out of the collected 6,154 malicious addresses, we find that there are a few duplicate addresses, so we filter them using string comparison because the addresses contains the alphanumeric values and we are left with 5,000 unique malicious addresses. After the string comparison, we find that few addresses are left, which have the same transactions. This problem occurs because some addresses are present in two different formats. For example, an address is present as `0xfea28ca175a80f5a348016583961f63be8605f80` and `0xFeA28ca175A80F5A348016583961f63bE8605f80`, but when we compare them as a string both are different. Therefore, we first convert all the addresses to lowercase and then we remove all the duplicate addresses. There are a few addresses in our dataset which have the null transaction. Hence, we remove all of them too, and finally, we are left with `4,375` malicious address. We apply the same technique to select the unique non-malicious address. After the unique address collection, we perform data preprocessing in two steps – filter contract account & EOA addresses and select verified non-malicious Ethereum account addresses.

Select Verified Non-malicious Ethereum Account Addresses. Figure 5 shows the process of selection of verified non-malicious addresses for further analysis. We filter all the non-malicious addresses by checking the `"to"` and `"from"` fields from all transactions performed by a given address. These fields provide the addresses with which the non-malicious addresses perform the transactions. If the non-malicious address performs a transaction with any malicious address, then we drop that address. The assumption is that such an address engaging in business with a known malicious address could itself be suspicious and hence we do not want it to represent non-malicious addresses. Finally, we select only those addresses which do not perform any transaction with any of the malicious addresses present in our dataset.

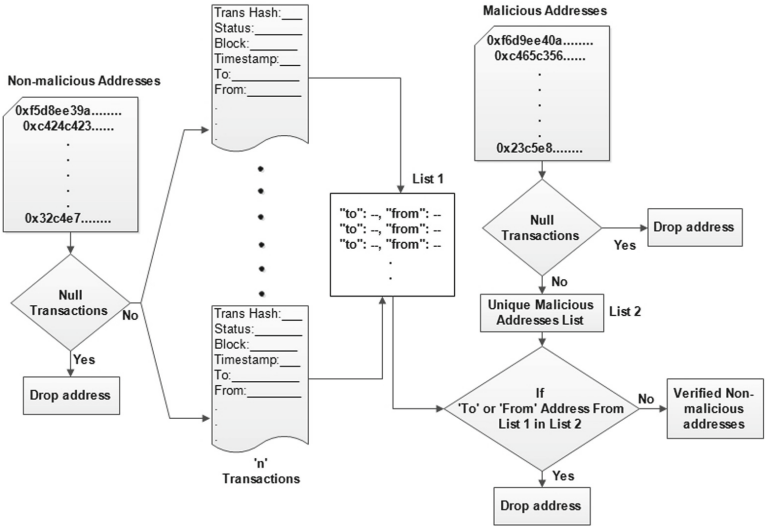

Fig. 5. Verification of non-malicious addresses

Filter Contract Account and EOA Addresses. We filter the EOAs and contract account addresses because both the account have different transaction behavioral features and they need to be analyzed separately. To filter the EOA and contract account addresses, we check the input data field from the collected transactions and find that in the case of EOA addresses, the input data field contains the `"0x"` value. However, in the case of contract account addresses, this field contains the bytecode of smart contract source code. Also, in the first transaction of the contract account addresses the `"to"` field is null, and the `"contractAddress"` field includes the address, which is opposite in case of EOA addresses. At last, after filtering the contract account and EOA addresses, we are left with 4,124 EOA and 251 contract account addresses out of 4,375 unique malicious addresses. Similarly, we randomly choose 5,000 non-malicious EOA addresses and 450 contract addresses for EOA and contract account address analysis, respectively.

4.4 EOA Analysis

In this section, we discuss the features extracted from the transactions performed by EOA addresses. All the transactions are stored in JSON file format. We use Python's JSON library to load, parse the file, and extract the pieces of information from the stored transactions. We extract the information from various fields of the transaction structure such as `"to"`, `"from"`, `"timestamp"`, `"gas"`, `"gasPrice"`, `"gasUsed"`, `"value"`, `"txreceipt_status"`. The features such as `Value_out`, `Value_in`, `Value_difference`, `Last_Txn_Value`, `Avg_value_in`, `Avg_value_out`, and other features related to ether values sent

and received are extracted from the `value` field. We extract features from the `timestamp` field such as first, last, and mean transaction time among all the transactions performed by an address. The `txreceipt_status` field from the transaction structure provides information about success and failed transactions. If the `txreceipt_status` field returns 1 then the transaction is successful or vice versa. We extract features like the number of failed and successful transactions in the incoming and outgoing transactions with the help of `txreceipt_status` field. The percentage of gas used for the transaction is calculated using `gasUsed` field value divided by the `gas` field value, which is set for the transaction. We get the percentage of gas used for all the incoming and outgoing transactions and the average value is taken to calculate the `AP_gasUsed_in and AP_gasUsed_out` features. All the features related to gas price, which is set in the transaction by the user who is willing to pay per gas used are extracted from the `gasPrice` field. All the extracted features from the transactions are shown in Table 1.

We extract 44 features for EOA addresses analysis in our feature extraction phase. Though we understand that all the extracted features are not essential to train the classifiers, and some may make the results of classification models worse because they do not participate in improving the performance of classification models. Therefore, we use the Information gain algorithm as a feature reduction method for dimensionality reduction of the feature vector. We select the top-10, top-20, top-30, top-40, and top-44 features with the highest info-gain score, as shown in Table 2. To do this selection process, we apply Random Forest [10], XGBoost [3], Decision Tree [15], and k-nearest neighbour (k-NN) [2] machine learning classifiers on top-10, top-20, top-30, top-40, and top-44 features. The final feature vector for EOA addresses consists of the selected top-30 features because we obtain the maximum ten-fold cross-validation accuracy for the top-30 features using the XGBoost classifier as shown in Table 5.

4.5 Smart Contract Account Analysis

There are two kinds of transactions present in the contract account addresses – contract creation and contract invocation by an EOA address as described in Subsect. 2.2. We first remove all the contract addresses before starting the analysis for a smart contract that contains a similar bytecode that is present in the input data field of the transaction structure. Finally, we have 250 malicious and 300 non-malicious smart contract account address for the analysis. The information we extract from the transactions performed by the contract account addresses. It is based on the interaction of the EOA account with the contract account. The various fields of the transaction structure such as `"to", "from", "contractAddress", "timestamp", "gas", "gasPrice", "gasUsed", "value"` are used to extract the features. Table 3 shows the extracted features for the smart contract analysis in the Ethereum network. From Table 3, one can observe that we extract the features from both creation and invocation transactions present in contract addresses. Features from feature id `F_1` to `F_4` are derived from the contract creation

Table 1. Extracted features for EOA analysis

F_ID	Feature	Description
F_1	TxnSent	The total number of transactions sent
F_2	TxnReceived	The total number of transactions received
F_3	Value_out	The total ether value sent
F_4	Value_in	The total ether value received
F_5	Value_difference	Absolute difference between value sent and received [(Value_out) - (Value_in)]
F_6	distinct_address	Number of distinct Address contacted
F_7	Total_Txn	Total Number of Transactions sent and received
F_8	Unique_TxnSent	The total number of transactions sent to unique addresses
F_9	Unique_TxnReceived	The total number of transactions received from unique addresses
F_10	First_Txn_time	The timestamp of the block in which the first ever transaction is made
F_11	Last_Txn_time	The timestamp of the block in which the last transaction is made so far
F_12	Active_duration	Active duration in second [(Last_Txn_time) - (First_Txn_time)]
F_13	Last_Txn_Bit	0/1 (0 if last transaction is incoming transaction else 1)
F_14	Last_Txn_Value	The ether value transferred in the last transaction
F_15	Avg_value_in	Average ether value received in incoming transaction
F_16	Avg_value_out	Average ether value sent in outgoing transaction
F_17	AP_gasUsed_in	The Average percentage of gas used in incoming transactions
F_18	AP_gasUsed_out	The Average percentage of gas used in outgoing transactions
F_19	gasPrice_out	The total number of gasPrice used in outgoing transactions
F_20	gasPrice_in	The total number of gasPrice used in incoming transactions
F_21	Avg_gasPrice_in	The Average gasPrice used in incoming transactions
F_22	Avg_gasPrice_out	Average gasPrice_out used in outgoing transaction
F_23	Failed_Txn_in	Total number of failed incoming transactions
F_24	Failed_Txn_out	Total number of failed outgoing transactions
F_25	Total_Failed_Txn	Total number of failed transactions (Failed_Txn_in + Failed_Txn_out)
F_26	Success_Txn_in	Total number of Success incoiming transactions
F_27	Success_Txn_out	Total number of Success outgoing transactions
F_28	Total_Success_Txn	Total number of Success transactions (Success_Txn_in + Success_Txn_out)
F_29	gasUsed_in	Total gasUsed in incoming transaction
F_30	gasUsed_out	total gasUsed in outgoing transaction
F_31	Per_TxnSent	Percentage of transactions sent from all the transactions
F_32	Per_TxnReceived	Percentage of transactions received from all the transactions
F_33	Std_value_in	Standard deviation of ether value received in incoming transaction
F_34	Std_value_out	Standard deviation of ether value sent in outgoing transaction
F_35	Std_gasPrice_in	Standard deviation of gasPrice used in incoming transactions
F_36	Std_gasPrice_out	Standard deviation of gasPrice used in outgoing transactions
F_37	First_Txn_Bit	0/1 (0 if last transaction is incoming transaction else 1)
F_38	First_Txn_Value	The ether value transferred in the first transaction
F_39	mean_in_time	Average time difference between incoming transaction
F_40	mean_out_time	Average time difference between outgoing transaction
F_41	mean_time	Average time difference between all transactions
F_42	Txn_fee_in	Total Transaction fee spent in incoming transaction
F_43	Txn_fee_out	Total Transaction fee spent in outgoing transaction
F_44	Total_Txn_fee	Total Transaction fee spent in all transaction (incoming + outgoing)

Table 2. Infogain results for EOA analysis

Rank	F_ID	Feature	Rank score	F_ID	Feature	Rank score	F_ID	Feature
0.46029	F_11	Last_Txn_time	0.19721	F_21	Avg_gasPrice_in	0.10026	F_44	Total_Txn_fee
0.42853	F_39	mean_in_time	0.19601	F_35	Std_gasPrice_in	0.10004	F_12	Active_duration
0.37441	F_10	First_Txn_time	0.19324	F_3	Value_out	0.07407	F_28	Total_Success_Txn
0.34817	F_40	mean_out_time	0.19112	F_16	Avg_Value_out	0.07407	F_7	Total_Txn
0.25935	F_17	AP_gasUsed_in	0.15878	F_19	gasPrice_out	0.05612	F_34	Std_value_out
0.24183	F_22	Avg_gasPrice_out	0.15177	F_29	gasUsed_in	0.02647	F_36	Std_gasPrice_out
0.23784	F_9	Unique_TxnReceived	0.14819	F_26	Success_Txn_in	0.01895	F_8	Unique_TxnSent
0.23590	F_33	Std_value_in	0.14819	F_2	TxnReceived	0.01500	F_13	Last_Txn_Bit
0.23146	F_15	Avg_Value_in	0.14327	F_5	Value_difference	0.01096	F_1	TxnSent
0.23082	F_4	Value_in	0.14327	F_6	distinct_address	0.01096	F_27	Success_Txn_out
0.22359	F_38	First_Txn_Value	0.13781	F_18	AP_gasUsed_out	0.00797	F_37	First_Txn_Bit
0.22202	F_32	Per_TxnReceived	0.13506	F_41	mean_time	0	F_25	Total_Failed_Txn
0.22202	F_31	Per_TxnSent	0.13322	F_14	Last_Txn_Value	0	F_23	Failed_Txn_in
0.21338	F_20	gasPrice_in	0.12016	F_43	Txn_fee_out	0	F_24	Failed_Txn_out
0.19777	F_42	Txn_fee_in	0.11065	F_30	gasUsed_out			

Table 3. Extracted features for smart contract account analysis

F_ID	Feature	Description
F_1	Contract_Create	Contract creation time
F_2	Txn_fee_contract_create	Transaction fee spent in contract creation
F_3	Per_gasUsed_contract_create	The percentage of gas used during contract creation
F_4	gasPrice_contract_create	Gas price used to create a contract
F_5	First_contract_invoke_time	Timestamp for first contract invocation
F_6	Last_contract_invoke_time	Timestamp for last contract invocation
F_7	Active_duration	Active duration (seconds) of contract address
F_8	Total_invoke	Total number of contract invocations
F_9	unique_invoke	Total number of contract invocations using unique address
F_10	Avg_Per_gasUsed_contract_invoke	The average percentage of gas used during contract invocations
F_11	gasPrice_contract_invoke	Total gas price used for contract invocations
F_12	Avg_gasPrice_contract_invoke	Average gas price used for contract invocations
F_13	Txn_fee_contract_invoke	Total transaction fee spent in contract invocations
F_14	Avg_Txn_fee_contract_invoke	Average transaction fee spent in contract invocations
F_15	Value_contract_invoke	Total ether value used in contract invocations
F_16	Avg_Value_contract_invoke	Average ether value used in contract invocations
F_17	gasUsed_contract_invoke	Total gas used for contract invocations
F_18	Avg_gasUsed_contract_invoke	Average gas used for contract invocations

transactions and features from feature id `F_5` to `F_18` are taken from the contract invocation transactions.

For smart contract address analysis, we extract 18 features. Similar to EOA address analysis, we use infogain as a feature selection algorithm to reduce the dimensionality of the feature vector. We select top-5, top-10, top-15, and top-18 features with the highest infogain score as shown in Table 4 and then apply the

same set of classifiers to train and test the model. Finally, we select the top-10 features to train the final model. The reason for selecting the top-10 features is that these set of features provide the highest ten-fold cross-validation accuracy using XGBoost classifier as shown in Table 5.

4.6 Classification

We use different machine learning classifiers using Python's Sckit-learn library, namely k-NN, Decision Tree, Random Forest, and XGBoost for the classification of malicious addresses in the Ethereum network. The experiments are carried out using the Intel i7 octa-core processor having Ubuntu 18.04 LTS with 32 GB RAM. We split the dataset into 70%-30% for the training and testing of our model. To check the performance of our model, we apply ten-fold stratified cross-validation. Also, we tune parameters to minimize the misclassification error.

Table 4. Infogain results for smart contract analysis

Rank score	F_ID	Feature	Rank score	F_ID	Feature
0.5732	F_6	Last_contract_invoke_time	0.2498	F_13	Txn_fee_contract_invoke
0.5389	F_17	gasUsed_contract_invoke	0.2498	F_4	gasPrice_contract_create
0.5387	F_5	First_contract_invoke_time	0.2006	F_12	Avg_gasPrice_contract_invoke
0.3442	F_18	Avg_gasUsed_contract_invoke	0.1418	F_10	Avg_Per_gasUsed_contract_invoke
0.3442	F_8	Total_invoke	0.141	F_7	Active_duration
0.3047	F_9	unique_invoke	0.1091	F_14	Avg_Txn_fee_contract_invoke
0.3047	F_3	Per_gasUsed_contract_create	0.1064	F_16	Avg_Value_contract_invoke
0.2498	F_11	gasPrice_contract_invoke	0.0459	F_15	Value_contract_invoke
0.2498	F_2	Txn_fee_contract_create	0.0459	F_1	Contract_Create

5 Experimental Results

This section describes the results achieved from the EOA analysis and smart contract account analysis. We perform the analysis for both the account types separately and extract the features from the behavior of the transactions present. We apply ten-fold cross-validation for both the analysis to evaluate our machine learning models' performance. Table 5 presents the 10-fold cross-validation results for separate machine learning classifiers on various numbers of selected features. First, we do the experiments for EOA addresses and examine the results presented in Table 5. We achieve the highest accuracy that is 96.54% with a False Positive Rate (FPR) of 0.92% for EOA analysis using XGBoost classifier with top-30 features. Secondly, we perform the analysis of smart contracts and examine the results presented in Table 5. For smart contracts analysis, we achieve the highest accuracy of 96.82% with an FPR 0.78% using the XGBoost classifier and top-10 features.

Table 5. Experimental results for EOA and smart contract analysis

Experimental results for EOA analysis				
No. of features	Random Forest(%)	XGBoost(%)	Decision Tree(%)	k-NN(%)
top-10	95.28	95.74	92.75	92.75
top-20	95.28	96.08	91.93	91.93
top-30	95.62	**96.54**	92.63	92.63
top-40	95.97	96.31	93.78	93.78
top-44	95.74	96.43	92.86	92.86
Experimental results for smart contract analysis				
top-5	94.73	94.73	94.73	94.73
top-10	94.73	**96.82**	94.73	94.73
top-15	94.73	96.49	96.49	96.49
top-18	94.82	96.49	96.49	96.49

6 Evaluation

Since 20^{th} January, when we last collected our experimental data – 85 new EOA addresses and only 1 new contract address are flagged as malicious. To further validate our models, we do the ensemble of all the machine learning classifiers used earlier to improve the detection accuracy. We test them on the data collected after 20^{th} January. Out of 85 malicious EOA addresses, our EOA address analysis model detects 81 as malicious. We also randomly choose 100 non-malicious addresses that are not part of our earlier dataset. Out of 100 non-malicious EOA addresses, our EOA address analysis model detects 97 as non-malicious, i.e., the overall accuracy of our model is 96.21% with FPR of 3% and FNR 4.71%. Similarly, our contract address analysis model detects the one newly collected contract address as malicious. This validates that our model works to a reasonable extent.

7 Conclusion

In this work, we train two classifiers using transactions performed by the Ethereum addresses on the Ethereum network for EOA analysis and smart contract account analysis. We collect malicious and non-malicious addresses from various sources. Still, the most important challenge is to label the non-malicious addresses because this work aims to detect malicious and non-malicious address with the help of supervised learning. We perform data preprocessing to select the verified non-malicious addresses and to filter the contract account and EOA addresses. We extract and select the features from the transactions of addresses and train different machine learning models, namely Random Forest, Decision

tree, XGBoost, and k-NN for EOA and smart contract account analysis. Finally, we achieve the highest accuracy of 96.54% and 96.82% for EOA and smart contract account analysis respectively. In the future, we will investigate how to reduce the false positives and false negatives.

References

1. FreeCodeCamp (2017). https://www.freecodecamp.org/news/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce/
2. Nearest neighbors (2018). http://scikit-learn.org/stable/modules/neighbors.html
3. Xgboost (2018). http://xgboost.readthedocs.io/en/latest/python/python_api.html
4. Contracts with verified source codes only (2019). https://etherscan.io/contractsVerified?filter=opensourcelicense
5. Cryptoscamdb (2019). https://documenter.getpostman.com/view/4298426/RzZ7nKcM?version=latest
6. Ethereum (2019). https://www.ethereum.org/
7. Ethereum blockchain explorer (2019). https://etherscan.io/
8. Etherscan label word cloud (2019). https://etherscan.io/labelcloud
9. MyEtherWallet Ethereum Darklist (2019). https://github.com/MyEtherWallet/ethereum-lists/blob/master/src/addresses/addresses-darklist.json
10. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
11. Chen, T., et al.: Understanding ethereum via graph analysis. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, pp. 1484–1492, April 2018. https://doi.org/10.1109/INFOCOM.2018.8486401
12. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
13. Pham, T., Lee, S.: Anomaly detection in bitcoin network using unsupervised learning methods. arXiv preprint arXiv:1611.03941 (2016)
14. Pham, T., Lee, S.: Anomaly detection in the bitcoin system-a network perspective. arXiv preprint arXiv:1611.03942 (2016)
15. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1993)
16. Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: extracting intelligence from the bitcoin network. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 457–468. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_29
17. Zhao, C.: Graph-based forensic investigation of bitcoin transactions (2014)