Scam Token Detection Based on Static Analysis Before Contract Deployment

Taichi Igarashi and Kanta Matsuura

The University of Tokyo, Tokyo, Japan
{itaichi1, kanta} @iis.u-tokyo.ac.jp

Abstract. In recent years, the number of crimes using smart contracts has increased. In particular, fraud using tokens, such as rug-pull, has become an ignorable issue in the field of decentralized finance because a lot of users have been scammed. Therefore, constructing a detection system for scam tokens is an urgent need. Existing methods are based on machine learning, and they use transaction and liquidity data as features. However, they cannot completely remove the risk of being scammed because these features can be extracted after scam tokens are deployed to blockchain. In this paper, we propose a scam token detection system based on static analysis. In order to detect scam tokens before deployment, we utilize code-based data, such as bytecodes and opcodes, because they can be obtained before contract deployment. Since N-gram includes information regarding the order of code sequences and scam tokens have the specific order of code-based data, we adopt N-gram of them as features. Furthermore, for the purpose of achieving a high detection performance, each feature is categorized into a scam-oriented feature or benign-oriented one to make differences in the values of feature vectors between scam and benign token. Our results show the effectiveness of code-based data for detection by achieving a higher F1-score compared to the methods of another field of fraud detection in Ethereum based on code-based data. In addition, we also confirmed that the position of effective code for detection is near the start position of runtime code in our experiments.

Keywords: Ethereum · Smart contract · Token · Fraud · Scam token.

1 Introduction

Today, a lot of services and applications using blockchain have been developed, and especially in Ethereum, various kinds of services have been realized by smart contract, which is the computer program interpreted by the Ethereum Virtual Machine (EVM). Some of these services utilize their original token to activate transactions because token enables users to trade their assets.

Thanks to the standard of token in Ethereum called ERC (Ethereum Request for Comments) [1], users who want to use the original token in their service easily develop their token by using this standard, and their token can be traded with other token which is developed by using the same ERC standard.

However, a lot of frauds using tokens have occurred in these days. As an ample, rug pull is now prevalent in Decentralized Finance (DeFi). Rug pull is a malicious maneuver where scammers create a new token and promote it to investors, only to abruptly abandon the project after they steal a lot of cryptocurrency from investors. In this fraud, the token issued by scammers, which is called "scam token" in the rest of this paper, seems to be benign but is craftily created to trick investors. For example, except the scammer who creates a scam token, users can only buy it but cannot sell it due to the crafty implementation. In this meaning, a scam token has no value for investors. Users can easily be scammed because scammers usually sell scam tokens by combining social engineering techniques. Today, the number of scam tokens has grown rapidly, as shown in the report of Solidus [2]. 125,084 scam tokens were found in 2022 whereas the number was 1,548 in 2020. This fact shows that there exists a high possibility that a lot of users encounter this type of scam, and it has become one of the big issues that cannot be ignored in recent years. Thus, detecting and reducing the number of scam tokens deployed to blockchain is important. In order to emphasize the meaning that " token " is one kind of smart contract, the word " token " is described as " token contract " in the rest of this paper.

Existing researches on fraud detection in Ethereum are categorized into three types: detection of fraudulent accounts, Ponzi scheme detection, and scam token contract detection. In the field of detecting fraudulent accounts, most researchers focus on identifying users who commit fraud, especially phishing scams. These methods are mainly based on transaction data extracted from blockchain. For detection, they construct a transaction network and create features for machine learning (ML) by applying an embedding method to the network. Ponzi scheme is one of the investment frauds. Scammers aim to go bankrupt at the appropriate time to obtain a lot of cryptocurrency from investors. To achieve their goal, they have to continue to collect a lot of investment, and at the same time, they also pay one part of the new users ' investment to existing investors on the pretense of paying a dividend. Combining social engineering with the smart contracts in which Ponzi scheme logic is incorporated, this type of fraud has occurred in Ethereum recently. Existing researches for detecting Ponzi schemes focus on the transaction data regarding trade of cryptocurrency, or the presence of Ponzi scheme logic in the smart contract. On the contrary to the above two kinds of scams, only a few researches about scam token contract detection have been proposed. Since existing researches in this field are usually based on transaction and liquidity data, which are extracted after token contracts are deployed to blockchain, there exists a high possibility that users are actually scammed even when their detection systems are applied to blockchain. Therefore, detecting scam token contracts before deployment is desirable.

In order to overcome the above issues, we propose a scam token contract detection system based on static analysis. Unlike transaction and liquidity data, code-based data, such as bytecodes and opcodes, can be obtained before the deployment of token contracts to blockchain. Thus, our method can detect it before users are actually scammed. The contributions of this paper are as follows:

1. To the best of our knowledge, our method is the first detection method of scam token contracts based on code-based data.
2. Our experimental results show the effectiveness of code-based data, realizing to detect scam token contracts before deployment to blockchain.
3. From our experiments, we confirmed that the position of the effective code for detection is near the beginning of the runtime code of token contracts.

The rest of this paper is constructed as follows. Related work of fraud in Ethereum is introduced in Section II. Proposed scheme is described in Section III. Evaluation results are shown in Section IV. Limitations and future work are described in Section V. Finally, the conclusion of this paper is presented in Section VI.

2 Related Work

In this section, we review studies related to the solutions against fraud in Ethereum. Though our main goal is to detect scam token contracts, there exist only a few researches directly on such detection to the best of our knowledge. Thus, we expand the range of surveys to find an effective way for detection and seek the requirements that the detection system should achieve by identifying overall shortcomings in this field.

2.1 Fraudulent account detection

Ibrahim et al [5] utilize effective features selected from transaction data, such as token names and the amount of ether that fraudulent accounts send and receive. In their experiment, they realized 98.77% accuracy as the best by using random forest, decision tree, and K-Nearest Neighbor (KNN). Though this method achieves high accuracy, the dataset used in their experiment is imbalanced such that the number of scam tokens is much less than that of benign tokens. Wen et al. [6] make a transaction network from the transaction and address data of phishing accounts and their neighbor nodes. They apply the transaction network-based features to some ML algorithms, such as Support Vector Machine (SVM), KNN, and AdaBoost. In particular, 92.76% accuracy was recorded when using AdaBoost. Although the above methods adopt relatively simple ML algorithms, Duan et al. [7] proposed a new embedding algorithm and Graph Convolutional Network (GCN) model suited for transaction network. Experimental results show that their new embedding algorithm and GCN model recorded better detection performance compared to existing embedding methods and GCN models, and achieved 94.6% accuracy.

2.2 Ponzi scheme detection

Wang et al. [9] proposed a detection method of the smart contract in which Ponzi scheme logic is incorporated, which is denoted as Ponzi smart contract below.

They utilize N-gram of opcode sequences as features for ML. One of the merits of adopting opcode sequences is that they include information of the order of calling function, which is generally effective in analyzing computer programs such as smart contracts. Due to the specific logic of Ponzi scheme, the order of function is considered to be important information. Thus, to focus on opcode sequences is totally reasonable. However, methods using code-based data are highly influenced by the code-reuse problem, which makes it difficult to distinguish between benign and malicious smart contracts when they have similar codes. Fan et al. [10] make a graph representing the deals between users and contracts from transaction data and generate topological features of the graph. Their method used multiple ML models to detect Ponzi smart contracts, and achieved a high F1-score with 0.946 especially when using XGBoost. However, their method cannot completely reduce the risk of being scammed because transaction data can be extracted only after a smart contract is deployed to blockchain. Aljofey et al. [11] utilized both transaction and opcodes data of contracts to make features for ML, and achieved 0.888 F1-score. Although they use both transaction-based and code-based features, which are considered to be effective for identifying Ponzi smart contracts, their detection performance is at the same level with other methods.

2.3 Scam token contract detection

Mazorra et al. [12] proposed a method of detecting scam token contract. They used transaction and liquidity data of token as features for XGBoost model, and recorded 99.36 % accuracy. However, Nguyen et al. [13] pointed out that they do not select effective features, and also multiple features include transaction data extracted after scam actually happens. In order to overcome this problem, Nguyen et al. [13] utilized only the transaction data which are extracted before scam occurs. They also achieved 0.990 F1-score in their experiment.

Mazorra et al. [12] proposed a method of detecting scam token contracts. They used transaction and liquidity data of token contracts as features for XGBoost model, and recorded 99.36 % accuracy. However, Nguyen et al. [13] pointed out that they do not select effective features, and also multiple features include transaction data extracted after the scam actually happens. In order to overcome this problem, Nguyen et al. [13] utilized only the transaction data which are extracted before the scam occurs. They also achieved a 0.990 F1-score in their experiment.

While these methods recorded high performance for detecting scam token contracts, it is difficult to prevent radically the occurrence of frauds caused by deployed scam token contracts. Since rug-pull, which is a fraud using a scam token contract, is executed in a short period as shown in the previous research [14], detection systems have to also detect in a short time to prevent the fraud. In this situation, methods based on transaction and liquidity data have to monitor all token contracts and wait until there is a sufficient amount of transaction and liquidity data related to each token contract to decide precisely whether it is benign or fraudulent. However, considering practical use, it is not realistic to monitor all token contracts due to the large number of them which have been deployed to blockchain already.

2.4 Requirements

Thus, a method which can detect before deployment is required for prevention of frauds.

Due to the above reasons and a growing number of scam token contracts, detecting all of them deployed to blockchain has become a fairly challenging task. Thus, a method that can detect before deployment is required for the prevention of fraud. When we aim at creating such a method, the thing which has to be considered is decreasing False Positive Rate (FPR), which is the metrics representing the extent of how many benign token contracts are misclassified as scam token contracts. This is because the action of benign users will be restricted if this kind of misclassification occurs. Considering these things, it is necessary to achieve the two requirements below:

1. Due to an increasing number of scam token contracts, a method which can detect them before deployment is desirable.
2. For the purpose of not restricting the deployments of benign token contracts, the detection system should achieve a low FPR as possible.

3 Proposed scheme

In order to satisfy the above requirements, we propose a scam token contract detection method based on static analysis. For the purpose of accomplishing the first requirement, namely preventing the deployment of scam token contracts, we focus on code-based data, such as bytecodes and opcodes. In general, smart contract code is interpreted by EVM before the deployment to blockchain. Thus, code-based data can be extracted before deployment, and is useful for achieving our goal. Thinking of code-based data, there exist three possible ways to generate features: using source codes, opcodes, and bytecodes. However, the source codes of most smart contracts are not available in Etherscan [8], which is the source of our datasets. On the contrary, we can obtain bytecodes from Etherscan, and also opcodes can be extracted by disassembling bytecodes. Though the effectiveness of opcodes for Ponzi scheme detection is demonstrated in [9], whether bytecodes are effective or not remains unclear. Therefore, we construct two detection methods, one is based on bytecodes, and the other is based on opcodes. Then, we compare the detection performance between them in order to find a more effective way.

To accomplish the second requirement, we design effective features from code-based data. As demonstrated in [9], N-gram is one of the effective features in the field of detecting malicious computer programs. N-grams are sets of words included in a given window whose size is N, and generally, this window is moved one word forward to compute them. Since N-gram includes information regarding the order of code, it is useful for static analysis of programs. Thus, we design

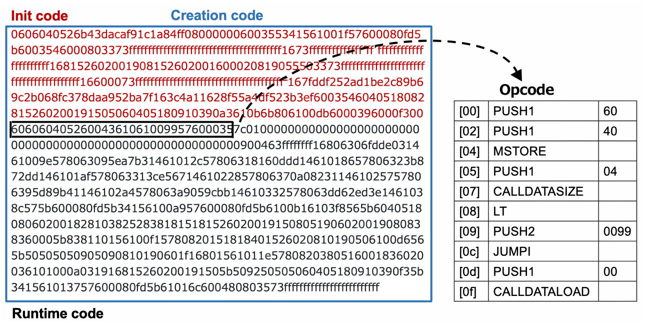
N-gram-based features. Moreover, in order to make differences between scam and benign token contracts, we divide N-gram features into two types: scam-oriented N-gram, and benign-oriented N-gram. Each N-gram feature is classified into one of these classes by comparing the percentage of contracts with the N-gram feature between scam and benign token contracts. By adopting this strategy, a difference appears in the values of feature vectors between them.

In the following subsections, we describe the bytecode of smart contract used in proposed method, feature engineering, and system model of proposed method

3.1 Bytecode of smart contract

The bytecode of smart contracts is mainly divided into two types: creation code and runtime code. Creation code is the specific code which deploys runtime code to blockchain. Unlike creation code, runtime code is actually stored in blockchain and defines the smart contract. What is the relation between them is that creation code has init code, which is responsible for mainly initializing the constructor, before runtime code. In other words, runtime code is a part of the creation code. Opcode is one part of assembly language obtained by disassembling bytecodes. In our method, we got opcodes by applying a disassembling tool [15] to bytecodes.

We extract only runtime code to detect scam token contracts. This is because the logic of token contracts is included in runtime code. Moreover, we can only extract runtime code in Etherscan because creation codes of most smart contracts are not provided. Therefore, we utilize only runtime code.



```
0606040526b43dacaf91c1a84ff08000000600355341561001f57600080fd5
b6003546000803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffff
ffffffffff168152602001908152602001600020819055503373ffffffffffffffffffff
ffffffffffffffff16600073ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2c89b6
9c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef6003546040518082
815260200191505060405180910390a3610b6b806100db6000396000f300
606060405260043610610099576000357c010000000000000000000000000
000000000000000000000000000000000900463ffffffff16806306fdde0314
61009e578063095ea7b31461012c57806318160ddd1461018657806323b8
72dd146101af578063313ce5671461022857806370a082311461025757806
395d89b41146102a4578063a9059cbb1461033257806dd62ed3e146103
8c575b600080fd5b34156100a957600080fd5b6100b16103f8565b6040518
0806020018281038252838181518152602001915080519060200190808 3
8360005b838110156100f157808201518184015260208101905061 00d656
5b505050509050908101906 01f1680156101 1e57808203805160018360 20
036101000a031916815260200191505b509250505060405180910390f35b
341561013757600080fd5b61016c600480803573ffffffffffffffffffffff
```

[00]	PUSH1	60
[02]	PUSH1	40
[04]	MSTORE	
[05]	PUSH1	04
[07]	CALLDATASIZE	
[08]	LT	
[09]	PUSH2	0099
[0c]	JUMPI	
[0d]	PUSH1	00
[0f]	CALLDATALOAD	

Fig. 1. Bytecode and opcode of smart contract

3.2 Feature engineering

Proposed method adopts code-based data as features for scam token contract detection. We designed two types of features, bytecode-based features and opcode-based features, in order to find a more effective one.

The basic scheme to generate a feature vector of each sample is common between these two features: to make N-gram features. For the purpose of generating N-gram features of each sample, N-gram dictionary, which is a set of N-grams and represented as the set ND below, is created in the following steps. First of all, for each sample, code-based data (bytecodes or opcodes) are extracted. Then, N-grams of code-based data are created, and the union of them composes a set NDi, when i is the index of the sample. Finally, when code-based data of all samples are extracted, ND is created as the union of ND_i. When using opcodes, ND is described as an opcode-based N-gram dictionary denoted by OND. In contrast, a bytecode-based N-gram dictionary is denoted by BND.

A feature vector of each sample is created on the basis of ND. In order to clarify the difference between scam and benign token contracts, each N-gram in ND is categorized into two types: scam-oriented N-gram and benign-oriented N-gram. Each scam-oriented N-gram in ND is selected when the percentage of scam token contracts with the N-gram is higher than that of benign token contracts, and vice versa. A feature vector of each sample is created on the basis of ND by assigning 1, -1, or 0 in accordance with the conditions below:

1. If an N-gram of ND is scam-oriented and is included in the sample, value 1 is assigned.
2. If an N-gram of ND is benign-oriented and is included in the sample, value -1 is assigned.
3. If an N-gram of ND is not included in the sample, value 0 is assigned.

In the case of using opcodes as features, however, hexadecimal digits are also obtained besides opcodes, which are the operands taken by the specific opcodes like "push" when converting bytecodes into opcodes. Some of these hexadecimal digits have meaningful information for detection because they may represent the specific address of EVM stack and storage,or signatures of functions. Therefore, besides the N-gram of opcodes, we also utilize these hexadecimal digits as features. For the purpose of using hexadecimal digits as features, a hexadecimal dictionary, represented as the set HD below, is also generated with the same series of the above procedure. The values of a feature vector of each sample based on HD are decided to be also 1, -1, or 0 by classifying each hexadecimal in HD into benign-oriented or scam-oriented. These series of procedures to make feature vectors are described in Appendix as pseudocode.

N-gram features are usually created on the basis of the frequency of each N-gram. However, in this strategy, the size of token contracts highly influences the detection performance because the bigger size of each token contract is, the more specific N-grams emerge in the token contract. Therefore, we assigned values on the basis of the existence of each N-gram to reduce such influence.

3.3 System model

Fig. 2 shows the system model and application place of proposed method. Proposed method firstly collects samples of token contracts from Etherscan [8].

Then, for each token contract, a feature is made using N-grams of code-based data (bytecodes, set of opcodes, and hexadecimal digits). Based on these features, the learning phase is executed by SVM model, which performs well in binary classification tasks. We used RBF kernel-based SVM, and a hyper-parameter C is set to 1.0. Using this model, whether each input sample is a scam token contract or not is decided. Though one of our goals is to realize high detection performance, we do not use other classifiers because selecting a better one is not our main goal, and also SVM is lightweight and has sufficient detection performance. Instead, our goal is to show the effectiveness of code-based data and find the position of effective data.

We assume that proposed method will be applied to Detection system 1 in Fig. 2, which tries to detect scam token contracts before they are deployed to blockchain. Aiming at realizing such a detection system, it needs to be integrated into client software or EVM because each contract deployment transaction is verified by validators. When each validator verifies such transaction, it executes the transaction on the client software and checks the occurrence of error in the calculation. Since the transaction has bytecodes of token contract, to decide whether it is a scam token or not can be realized in this validation process when our model is applied to client software or EVM. On the contrary, existing methods work as Detection system 2 in Fig. 2, which detects only scam token contracts which have been deployed to blockchain already, because they are based on transaction and liquidity data. Due to an increasing number of scam token contracts, to prevent the deployment is important besides to detect ones deployed to blockchain. Thanks to the use of code-based data, proposed method can prevent the deployment itself. Even if proposed method cannot detect some scam token contracts due to misclassification, we can reduce the number of ones deployed to blockchain, meaning that the possibility for users of being scammed and the amount of work of Detection system 2 can be lower.

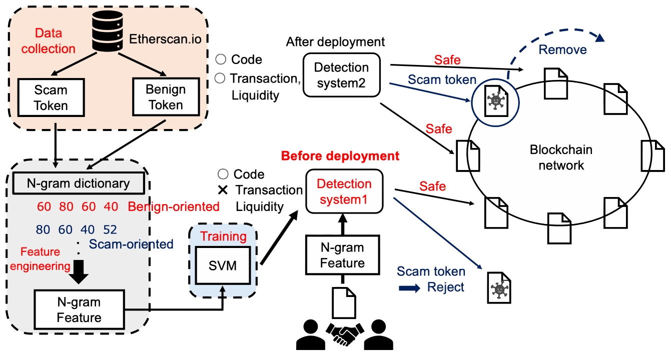


Fig. 2. System model and application place of proposed method

4 Evaluation

4.1 Dataset

Following the existing researches [12, 13], our datasets of token contract samples were collected from Etherscan [8]. The number of scam token contracts is 1,259, whereas benign is 981. For each scam token contract sample, the label is provided as Phish/Hack by the Etherscan Token tracker. On the other hand, we collected benign token contracts by selecting reliable ones which are not labeled as Phish/Hack and have a lot of valid transactions. In the field of fraud detection based on supervised learning in Ethereum, it is common to use labels provided by Etherscan for evaluation. Thus, in our all experiments, we trust the labels from Etherscan. The ratio of train and test data is 7:3.

4.2 Evaluation Metrics

In order to evaluate proposed method, general metrics in classification tasks, such as accuracy, precision, recall, and F1-score are utilized in our experiments. In particular, F1-score is an important metric indicating the overall performance even when the dataset is imbalanced. Let TP, FN, TN, and FP denote the number of scam token samples classified correctly as "scam", the number of scam token samples misclassified as "benign", the number of benign token samples classified correctly as "benign", and the number of benign token samples misclassified as "scam". These metrics are calculated as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} , \tag{1}$$

$$Precision = \frac{TP}{TP + FP} , \tag{2}$$

$$Recall = \frac{TP}{TP + FN} , \tag{3}$$

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} . \tag{4}$$

Furthermore, we also use FPR, which is the ratio of how many benign tokens are misclassified as "scam", for the second requirement. In contrast to the above metrics, the detection performance is well if FPR is low. This metric is defined as

$$FPR = \frac{FP}{FP + TN} . \tag{5}$$

4.3 Experiment

There exist three main goals of our experiments: demonstrating the effectiveness of code-based data, knowing the position of effective information in token contract code, and investigating the influence of fixing the length of bytecode. As a prerequisite, one of the shortcomings of using code-based data is that the detection performance is relatively low when there exists similar code between scam and benign token contracts. This problem is called code-reuse problem, and proposed method can be influenced by this problem because token contracts are generally created by imitating existing ones according to the report [16]. Thus, we aim at achieving the same level of detection performance with methods using code-based data in other scam detection fields on Ethereum. If proposed method records such a level, we can contribute to the field of scam token contract detection significantly in terms of reducing radically the risk of fraud and the amount of work to find deployed scam token contracts. At the same time, we also intended to know the position of effective information which makes a difference between scam and benign token contracts for developing future research. Besides these two goals, we also aim at knowing the effect when the length of the bytecode is fixed. This is because there exists a high possibility that a big difference can appear in the existence of N-gram, which is the basic of feature for proposed method, when all lengths of bytecodes are used. If the detection performance when using the fixed length of bytes is the same level or higher than that of when using all bytes, the former is better in terms of being not influenced by contract size and reducing computational complexity.

Considering the above three goals, we decided to conduct three kinds of experiments. In the first place, we evaluated detection performance when all lengths of bytecodes are used in order to demonstrate the effectiveness of code-based data. In the second place, for the purpose of knowing the position of effective information, detection performances when changing the position of bytecodes with a fixed length were measured. Finally, we investigated the influence of the length and position of bytecodes on detection performance by changing them based on the result of the second experiment.

Experiment using all bytes In order to evaluate the effectiveness of code-based data for scam token contract detection, in this experiment, we utilized all lengths of bytecodes and evaluated the detection performances for each feature: bytecode-based, and opcode-based. For the purpose of reducing the number of dimensions and shortening the learning time, features are created by adopting 2-gram ($N = 2$) of code-based data. Table 1 shows the detection performance when using all lengths of bytecodes. As shown in Table 1, proposed method achieved more than 90% accuracy in both the situation using bytecode and opcode. Compared to [12], which uses transaction and liquidity data, the performance of proposed method is worse because they achieved 99.36% accuracy. However, compared to [9], which is the detection method of Ponzi smart contract focusing on opcode, proposed method outperforms their method in terms of recording a higher F1-score. We achieved 0.918 with opcode features whereas their method recorded 0.90 in their experiment. Furthermore, proposed method also surpasses [11], which is also the detection method of Ponzi smart contract based on various features composed of opcode, source code, and transaction data. This is because the F1-score of their method is at most 0.887 in their experiment, while proposed method achieved 0.918 with opcode features.

Table 1. Detection performance when using all bytecodes

Features	Accuracy	Precision	Recall	F1-score	FPR[%]
Bytecode	0.906	0.942	0.883	0.911	6.56
Opcode	0.911	0.933	0.903	0.918	7.97

Considering the above results and one of our goals, which is achieving the same level of performance with methods which are based on code-based data in other fraud detection fields on Ethereum, the detection performance of proposed method is adequate. Therefore, we can conclude that code-based data, such as bytecode and opcode, are effective for scam token contract detection.

Investigation of the position of effective bytes In order to know the position of effective information, we conducted an experiment demonstrating the differences in performance when the position of the bytecode is changed. Before this experiment, at the beginning, we checked the length of the bytecode for all samples.

Table 2 shows the bytecode length of all samples. As shown in Table 2, we found that about 88% of all samples have over 2048 bytes, and the length which the largest number of samples have is 4097-8192 bytes. Moreover, scam token contracts tend to have a shorter length of bytes compared to benign token contracts in our dataset. Considering these facts, there exists a possibility that the distinctive codes of scam token contracts are located near the initial position of runtime code. Therefore, to identify the position of effective information, we measured detection performances when changing the start position of extracted bytecodes on the assumption that effective information exists at most within the first 4096 bytes. Provided that the token contracts which do not have the bytecodes at the specified position are excluded. The length of extracted bytecodes is fixed to 128 bytes, and N is also set to 2. In addition, only bytecode-based feature is used because we do not aim at optimizing the detection performance but just knowing the position of effective bytecodes in this experiment.

Table 2. Bytecode length of all samples

Length [B]	Scam token	Benign token	Total
1-256	12	1	13
257-512	51	0	51
513-1024	131	13	144
1025-2048	20	44	64
2049-4096	285	224	509
4097-8192	550	412	962
8193-16384	178	235	413
16385-32768	32	52	84

Table 3 shows the detection performance when changing the position of bytecodes. From Table 3, the bytecodes at 129-256-th byte include the most effective information because the detection performance is the highest as they achieved 0.915 F1-score. This result is quite similar to that of when using all bytecodes shown in Table 1, furthermore, outperforms especially in precision, F1-score, and FPR. From this result, we can firstly say that selecting effective information is more important than using all bytes because a high detection performance was achieved without extracting long lengths of bytes. Moreover, the bytecodes up to 1024-th byte seem to have effective information compared to ones after 1024-th byte because all the case when using bytecodes in each of the eight categories up to 1024-th byte (1-128-th, 129-256-th, ..., 897-1024-th) achieved more than 0.880 F1-score, which is recorded when using bytecodes in only eight out of the other twenty-four categories after 1024-th byte. Therefore, we consider that the effective information for scam token contract detection exists near the start position of runtime code. Besides that, we also found some candidates for effective bytecodes. For example, the bytecodes at 3457-3584-th byte can be useful for accomplishing our purpose because not only F1-score is over 0.900 but also the fourth lowest FPR is recorded when using them. The bytecodes at the three categories, 1793-1920-th, 2177-2304-th, and 2561-2688-th byte, are also effective as they contribute to the top eight F1-score. The top eight categories in terms of F1-score are listed in Table 4.

Table 3. Detection performance when changing the position of bytecodes

Position	Accuracy	Precision	Recall	F1-score	FPR[%]
1-128	0.894	0.970	0.842	0.901	3.48
129-256	0.906	0.952	0.881	0.915	5.92
257-384	0.871	0.912	0.860	0.885	11.31
385-512	0.904	0.934	0.884	0.908	7.28
513-640	0.870	0.957	0.814	0.880	5.19
641-768	0.879	0.929	0.842	0.883	7.74
769-896	0.889	0.916	0.863	0.889	8.39
897-1024	0.883	0.920	0.854	0.886	8.36
1025-1152	0.872	0.912	0.853	0.881	10.41
1153-1280	0.842	0.901	0.782	0.837	9.28
1281-1408	0.845	0.904	0.773	0.833	8.25
1409-1536	0.860	0.918	0.797	0.853	7.43
1537-1664	0.869	0.941	0.789	0.858	5.02
1665-1792	0.865	0.943	0.789	0.859	5.26
1793-1920	0.887	0.948	0.839	0.890	5.51
1921-2048	0.880	0.942	0.825	0.880	5.78
2049-2176	0.881	0.955	0.815	0.879	4.35
2177-2304	0.890	0.937	0.843	0.887	6.01
2305-2432	0.879	0.953	0.826	0.885	5.26
2433-2560	0.865	0.927	0.820	0.870	7.87
2561-2688	0.884	0.920	0.871	0.895	9.84
2689-2816	0.880	0.934	0.842	0.886	7.29
2817-2944	0.876	0.960	0.807	0.877	4.07
2945-3072	0.863	0.934	0.785	0.853	5.69
3073-3200	0.847	0.873	0.812	0.841	11.79
3201-3328	0.870	0.976	0.778	0.866	2.24
3329-3456	0.863	0.919	0.797	0.853	7.11
3457-3584	0.903	0.963	0.844	0.900	3.51
3585-3712	0.869	0.931	0.814	0.868	6.91
3713-3840	0.845	0.956	0.737	0.833	3.70
3841-3968	0.880	0.927	0.830	0.876	6.82
3969-4096	0.873	0.971	0.765	0.856	2.23

Table 4. The top eight categories in terms of F1-score

1st	2nd	3rd	4th	5th	6th	7th	8th
129-256	385-512	1-128	3457-3584	2561-2688	1793-1920	769-896	2177-2304

Influence of length and position on detection performance In order to know whether the detection performance is enough when the length and position of bytecodes are fixed, we conducted experiments based on the result of the second experiment on the two conditions below:

1. We compare the detection performances when changing the length of bytecodes extracted from the first 1024 bytes of runtime code, which includes relatively effective information demonstrated by the second experiment.
2. The detection performances when using the bytecodes in the top eight categories regarding F1-score in Table 3 are also checked in order to achieve a better performance of proposed method. Since some of these bytecodes are not included in the samples whose size is small, we also have to confirm if this condition is influenced by token contract size by checking the size of the misclassified samples.

If the performances of proposed method in these conditions are the same level or higher than that of when using all bytes, the formers are better because they are less influenced by token contract size and need lower computational complexity.

In the first place, we conducted an experiment on the first condition. In this experiment, we compared the result when N is set to 2 or 4 (2-gram or 4gram). Table 5 shows the detection performances when the length of bytecodes is changed within the first 1024 bytes. From Table 5, proposed method achieved 0.917 accuracy and 0.925 F1-score at the best when using 2-gram bytecodes features from 1024 bytes. Since this result is higher than that of the first experiment shown in Table 1, to fix the length of extracted bytecodes works positively with regard to being less influenced by token contract size and reducing computational complexity. Moreover, proposed method outperforms [9] and [11], which are the detection methods of Ponzi smart contract focusing on opcode, because each of their methods achieved 0.90 and 0.887 F1-score while the proposed method achieved 0.925. From Table 5, we also consider that the difference of N does not affect detection performances, especially when the length of bytecodes is enough to realize good performance. Though 4-gram was more effective when the length was short, the difference in detection performance between when using 2-gram and 4-gram did not become big as the increase of length. When the amount of information is not sufficient to detect, 4-gram can help increase them because it includes more information regarding the order of code-based data compared to 2-gram. However, this effect can be decreased as the amount of information becomes sufficient. Considering the above things and the influence of the number of dimensions on learning time, we consider that using 2-gram of code-based data is more efficient than using 4-gram.

Table 5. Detection performance when changing the length of bytecodes

		2-gram					4-gram				
Length [B]	Feature	Accuracy	Precision	Recall	F1-score	FPR [%]	Accuracy	Precision	Recall	F1-score	FPR[%]
16	bytecode	0.660	0.629	0.944	0.755	69.00	0.795	0.687	0.941	0.794	65.4
	opcode	0.644	0.619	0.963	0.754	77.01	0.676	0.655	0.885	0.753	58.92
32	bytecode	0.817	0.897	0.761	0.823	11.15	0.841	0.875	0.824	0.849	13.96
	opcode	0.751	0.889	0.655	0.754	11.39	0.810	0.860	0.769	0.812	14.38
64	bytecode	0.824	0.888	0.791	0.837	13.10	0.891	0.923	0.875	0.898	8.88
	opcode	0.821	0.882	0.801	0.840	15.00	0.830	0.913	0.781	0.842	10.25
128	bytecode	0.896	0.948	0.864	0.904	6.19	0.897	0.933	0.887	0.909	8.87
	opcode	0.872	0.916	0.855	0.884	10.45	0.882	0.938	0.847	0.890	7.14
256	bytecode	0.890	0.936	0.861	0.897	7.38	0.897	0.961	0.863	0.909	5.16
	opcode	0.884	0.948	0.836	0.889	5.67	0.872	0.886	0.882	0.884	14.00
512	bytecode	0.905	0.936	0.884	0.909	7.10	0.902	0.957	0.868	0.910	5.24
	opcode	0.885	0.907	0.874	0.890	10.13	0.905	0.935	0.890	0.912	7.71
1024	**bytecode**	**0.917**	**0.940**	**0.910**	**0.925**	**7.48**	0.915	0.931	0.913	0.922	8.20
	opcode	0.906	0.945	0.897	0.920	7.89	0.905	0.940	0.891	0.914	7.64

In order to seek better performance, we used the bytecodes in the top eight categories regarding F1-score shown in Table 4 instead of the first 1024 bytes, which includes the bytecodes in the four out of the top eight categories. From the experiment on the first condition, N is only set to 2 (2-gram) in this condition. We changed the number of bytecodes used as features within the top eight categories, and compared the results. Table 6 shows the detection performance when using the bytecodes in the top eight categories. This result shows that the highest detection performance is recorded when using bytecode-based features constructed from all the bytecodes in the top eight categories. We achieved 0.924 accuracy and 0.933 F1-score, which are the best performances of all our experiments. From Table 5 and Table 6, we can say that bytecode is slightly more effective than opcode in proposed methods. In regard to FPR, [11] achieved 2.14 % while proposed method recorded 6.67 %. This result seems to show that proposed method is inferior to their method for the purpose of achieving a low FPR, which is the second requirement. However, their dataset is highly imbalanced such that the number of benign contracts is 1,596 whereas only 308 scam contracts are used. Therefore, it is easier to learn the benign contracts in their experiments compared to our experiments. Moreover, we consider that the number of benign token contracts is not so many compared to that of scam token contracts in the real world. This is shown in [13] and [12], which are the methods of scam token contract detection, as [13] collected 23,871 scam token contracts and 1,830 benign contracts, and [12] collected 24,870 scam token contracts and only 674 benign contracts. Assuming that 1,880 benign token contracts, which is the largest number of them used in the researches of scam token contract detection, are generated within one year, we have to deal with only 0.334 false positives per day caused by the FPR of proposed method. Considering these facts, we can conclude that the FPR of proposed method is within an allowable range.

In order to know whether the result is influenced by the token contract size or not, we also checked the size of misclassified samples. We found that only

8.16 % of misclassified samples on the first condition do not have a part of bytecodes in the first 1024 bytes while 34.62 % of misclassified samples on the second condition do not include a part of bytecodes in the top eight categories. From these results, we can firstly say that the detection performances are not highly influenced by the size of token contracts on both conditions because most of the misclassified samples have all the specified bytecodes. However, the method on the first condition, using the first 1024 bytes, is more immune to token contract size than that of on the second condition considering the percentage of misclassified samples which do not have a part of specified bytecodes. Therefore, the method using 2-gram of bytecodes extracted from the first 1024 bytes is the better choice when we aim at being independent of token contract size as possible.

Table 6. Detection performance when using the bytecodes in the top eight categories

	bytecode					opcode				
Categories	Accuracy	Precision	Recall	F1-score	FPR [%]	Accuracy	Precision	Recall	F1-score	FPR[%]
Top 2	0.894	0.926	0.879	0.902	8.67	0.899	0.934	0.886	0.909	8.36
Top 3	0.900	0.962	0.860	0.908	4.55	0.876	0.922	0.858	0.889	9.82
Top 4	0.915	0.934	0.905	0.919	7.35	0.879	0.926	0.854	0.889	8.84
Top 5	0.890	0.948	0.855	0.899	6.32	0.893	0.925	0.88	0.902	9.12
Top 6	0.896	0.926	0.880	0.902	8.52	0.900	0.943	0.889	0.915	8.30
Top 7	0.905	0.946	0.882	0.913	6.53	0.896	0.926	0.887	0.906	9.21
Top 8	**0.924**	**0.949**	**0.917**	**0.933**	**6.67**	0.911	0.933	0.903	0.918	7.98

5 Limitation and Future work

Proposed method has some drawbacks. Firstly, the detection performance of proposed method is not sufficient as compared with that of the methods based on transaction and liquidity data. This is because our method is also influenced by code-reuse problem since code-based data are used as features. For not being influenced by this problem, we have to consider the more effective feature engineering technique to make a big difference between scam and benign token contracts. Secondly, the features of proposed method do not have interpretability for detection. Though we reveal the position of effective bytecodes for detecting scam token contracts, their specific functions or behaviors cannot be found in the proposed method. In order to know these things as a future work, the use of Control Flow Graph (CFG), which represents the control flow during the execution of token contracts and is constructed from opcode sequences, can be effective.

6 Conclusion

In this paper, we have proposed a scam token contract detection method based on static analysis. This is realized by using N-grams of code-based data, such as bytecodes and opcodes. Thanks to using code-based data, we realized the scam token contract detection before deployment. N-gram of code-based data and our feature engineering scheme also contribute to the relatively high F1-score and low FPR compared to the methods of other fields of fraud detection in Ethereum based on code-based data. Furthermore, we found that the effective bytecodes exist near the start position of runtime code in our experiment. In particular, proposed method recorded a high performance when using 1024 bytes from the start of runtime code.

Acknowledgement. We thank the anonymous referees for their valuable comments and helpful suggestions. This work was partially supported by JSPS KAKENHI Grant Number 22H03589, and JST CREST Grant Number JPMJCR22M1, Japan.

References

1. Ethereum Improvement Proposals. ERC-20: Token Standard. [Online] Available: https://eips.ethereum.org/EIPS/eip-20 . Accessed: 2023-09-19.
2. Solidus Lab. The Rug Pull Report. [Online] Avalilable: https://www.soliduslabs.com/reports/rug-pull-report . Accessed: 2023-09-19.
3. Loi Luu, Duc H.Chu, Hrishi Olicke, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254-269, 2016.
4. Qi Yuan, Baoying Huang, Jie Zhang, Jiajing Wu, Haonan Zhang, and Xi Zhang. Detecting Phishing Scams on Ethereum based on Transaction Records. in *2020 IEEE International Symposium on Circuits and Systems*, IEEE, pages 1-5, 2022.
5. Rahmeh F. Ibrahim, Aseel M. Elian, and Mohammed Ababneh. Illicit Account Detection in the Ethereum Blockchain Using Machine Learning. in *Proceedings of 2021 International Conference on Information Technology*, pages 488-493, 2021.
6. Haixian Wen, Junyuan Fang, Jiajing Wu, and Zibin Zheng. Transaction-based hidden strategies against general phishing detection framework on ethereum. in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 2021.
7. Xincheng Duan, Biwei Yan, Anming Dong, Li Zhang, and Jiguo Yu. Phishing Frauds Detection Based on Graph Neural Network on Ethereum. in *International Conference on Wireless Algorithms, Systems, and Applications*, Cham: Springer Nature Switzerland, pages 351-363, 2022.
8. Etherscan. The Ethereum Blockchain Explorer. [Online] Available: https://etherscan.io . Accessed: 2023-09-19.
9. Mengxiao Wang, and Jing Huang. Detecting Ethereum Ponzi Schemes Through Opcode Context Analysis and Oversampling-Based AdaBoost Algorithm. in *Computer Systems Science & Engineering*, 47(1):1023-1042, 2023.
10. Shuhui Fan, Shaojing Fu, Yuchuan Luo, Haoran Xu, Xuyun Zhang, and Ming Xu. Smart Contract Scams Detection with Topological Data Analysis on Account Interaction. in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 468-477 2022.
11. Ali Aljofey, Abdur Rasool, Qingshan Jiang, and Qiang Qu 1. A Feature-Based Robust Method for Abnormal Contracts Detection in Ethereum Blockchain. in *Electronics*, 11.18, 2937, 2022.

12. Bruno Mazorra, Victor Adan, and Vanesa Daza. Do Not Rug on Me: Leveraging Machine Learning Techniques for Automated Scam Detection. in *Mathematics*, volume 10, pages 1-24, 2022,

13. Minh H. Nguyen, Son H. Dau, and Xiaodong Li. Rug-pull malicious token detection on blockchain using supervised learning with feature engineering. in *Proceedings of the 2023 Australasian Computer Science Week*, pages 72-81, 2023.

14. Pengcheng Xia, Haoyu Wang, Bingyu Gao, Weihang Su, Zhou Yu, Xipau Luo, Chao Zhang, Xusheng Xiao, Guoai Xu, Demystifying Scam Tokens on Uniswap Decentralized Exchange. in *arXiv 2021*, arXiv:2109.00229.

15. pyevmasm. API Reference. [Online] Available: https://pyevmasm.readthedocs.io/en/latest/api.html . Accessed: 2023-09-19.

16. Ningyu He, Lei Wu, Haoyu Wang, Yao Guo, and Xuxian Jiang. Characterizing Code Clones in the Ethereum Smart Contract Ecosystem. in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, 2020 Revised Selected Papers 24*, pages 654-675. Springer International Publishing, 2020.

Appendix Algorithm of feature engineering

Algorithm 1 Making dictionary

```
Input: All $Samples$
 1: for $Sample_i$ in all $Samples$ do
 2:     Extract code-based data of $Sample_i$
 3:     if code-based data are bytecodes then
 4:         Make $N$-grams of bytecodes
 5:         Create set $BND_i$, the union of all $N$-grams of bytecodes in $Sample_i$
 6:     else if code-based data are opcodes then
 7:         Divide opcodes data into opcodes and hexadecimal digits
 8:         Make $N$-grams of opcode
 9:         Create set $OND_i$, the union of all $N$-grams in opcodes in $Sample_i$
10:         Create set $HD_i$, the union of all hexadecimal digits in $Sample_i$
11:     end if
12: end for
13: Make $BND$, the union of all $BND_i$ if code-based data are bytecodes.
14: Make $OND$ and $HD$, the unions of all $OND_i$ and $HD_i$ if code-based data are
    opcodes.
```

Algorithm 2 Feature engineering

```
Input: Each $Sample$
 1: Extract code-based data of $Sample$
 2: if Code-based data are bytecodes then
 3:     Make $N$-grams of bytecodes
 4:     for all $N$-gram in $Sample$ do
 5:         if $N$-gram is in $BND$ then
 6:             if $N$-gram is scam-oriented $N$-gram then
 7:                 Feature vector[index of the $N$-gram] = 1
 8:             else if $N$-gram is benign-oriented $N$-gram then
 9:                 Feature vector[index of the $N$-gram] = -1
10:             end if
11:         else
12:             Feature vector[index of the $N$-gram] = 0
13:         end if
14:     end for
15: else if Code-based data are opcodes then
16:     Divide opcode-data into opcodes and hexadecimal digits
17:     Execute the same procedure in line 3-14 replacing $BND$ with $OND$ to create
        opcode-based features
18:     Execute the same procedure in line 4-14 replacing $N$-gram and $BND$ with hex-
        adecimal digits and $HD$ to create hexadecimal-based features
19: end if
```