Honeypot Contract Risk Warning on Ethereum Smart Contracts

Weili Chen
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
chenwli28@mail.sysu.edu.cn

Xiongfeng Guo
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
530416041@qq.com

Zhiguang Chen*
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
zhiguang.chen@nscc-gz.cn

Zibin Zheng*
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
zhzibin@mail.sysu.edu.cn

Yutong Lu
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
yutong.lu@nscc-gz.cn

Yin Li
Institute of Software Application Technology Guangzhou
Chinese Academy of Sciences
Guangzhou, China
liyin@gz.iscas.ac.cn

***Abstract*—As Ethereum's smart contracts have boomed, it has become an integral part of the blockchain ecosystem. Unfortunately, some malicious users also find the opportunity to use fraudulent means to profit. A new reported approach is to lure new users or other attackers into the contract in an attempt to make a profit by exposing seemingly obvious flaws in the contract. But in fact, the contract contains a hidden trap that ultimately benefits the creator of the contract. Such contracts are known as honeypot contracts in the blockchain ecosystem. Previous studies proposed two methods to identify such smart contracts by using symbolic execution and contract behaviors. However, these methods either make it difficult to discover new categories or fail to warn users before they lose money. To solve this problem, we propose a machine learning model to detect honeypot contracts based on N-gram features and LightGBM. Extensive experiments show that our proposed model performs well in different conditions.**

***Index Terms*—Blockchain, Ethereum, Smart contract, Honeypot, LightGBM**

I. Introduction

Since the creation of Bitcoin in 2009, blockchain technology has attracted a lot of attention ranging from researchers to developers and investors. It is regarded as a disruptive technology that will revolutionize many traditional industries. Simply speaking, a blockchain can be seen as an append-only, verifiable distributed ledger system. In the system, transactions that happened in a certain period were packaged into a *block* and linked to the previous block through cryptographic means, thus forming a *blockchain*. The transactions are the data stored in the blockchain. A blockchain usually maintains some kind of digital asset or cryptocurrency, and the transactions are records of asset transfer. In the Bitcoin blockchain, bitcoin is the corresponding cryptocurrency.

Bitcoin's popularity has led to the creation of many blockchains. Among these blockchains, Ethereum is the biggest in terms of market capitalization according to *coinmarketcap.com*. The biggest difference between Ethereum and Bitcoin is that Ethereum is a Turing-complete platform for smart contracts. Smart contracts implemented on blockchains are automatically enforced computer protocols between mutually distrusting participants [1, 2]. The difference between smart contracts and ordinary contracts is that a smart contract may control a certain amount of cryptocurrency and the execution of the smart contract cannot be tamped. This makes smart contracts combined with blockchain have a wide range of applications [3], for example, cloud computing [4], IoT [chen2018iot], healthcare [5] and insurance [6].

However, due to the immutable characteristics of blockchain, vulnerabilities in smart contracts cannot be fixed. This makes smart contracts a target of various attacks in the blockchain. As the cryptocurrency becomes more and more valuable, finding vulnerabilities and exploiting them has become more active in the community. In fact, Ethereum already faced several devastating attacks on vulnerable smart contracts, for example, the DAO hack in 2016 [7] and the Parity Wallet hack in 2017 [8], which together causing a loss of over $400 million.

As various tools for detecting vulnerabilities before deploying contracts are proposed [9, 10, 11], and as developers mature, vulnerabilities in smart contracts have become fewer and harder to find. Interestingly, recent research has shown that there are attackers who deliberately release flawed contracts to lure ordinary attackers [12, 13]. This new type of smart contracts is called "Honeypot" [14, 15]. Specifically, honeypot contracts are smart contracts on Ethereum, but unlike regular contracts that seek security, honeypot contracts are deliberately designed by developers to have a flaw in them to entice their attackers to exploit that flaw. Often, honeypot contracts require the attacker to transfer a certain amount of ether to exploit the vulnerability. However, the contract did not run as expected after the payment, and the investment cannot be recovered. The important reason for the success of honeypot contracts is that some greedy attackers are eager to make money by exploiting vulnerabilities after they find them, without thinking that this

978-1-7281-6981-1/20/$31.00 ©2020 IEEE
DOI 10.1109/JCC49151.2020.00009

is the bait laid by more sophisticated attackers. Therefore, honeypot creators often publish the source to facilitate the detection and validation of their "vulnerable" smart contracts. There are many types of honeypot contracts, and they may trap victims at different levels, including the underlying EVM, the language of the contract (such as Solidity), and even the browser for the contract (such as Etherscan.io) [12].

We have noticed that there is two research focused on it. As the first systematic analysis of honeypot smart contracts, paper [12] provides a taxonomy of honeypot techniques and a tool that employs symbolic execution and well-defined heuristics to expose honeypots. The other research [13] presents a data science approach based on contract transaction behavior.These two researches uses their own tools to analyse some comments on Etherscan and get the label which was tagged by Etherscan.The dataset is a little different from ours: we do not include some comments which was not virified by the website. Though this method may bring more positive honeypot samples, we think these unverified comments are not so reliable and may take a negative influence on our model. In this paper, we propose a machine learning method based on N-gram of the opcode for the detection of honeypot. Compared with [12, 13], this proposed method has two advantages: 1) we do not depend on the contract's behavior, thus we can give early warning at the moment of the contract deployed (before causing any user loss); 2) This approach can learn the pattern of honeypot contracts at the bytecode level, thus facilitating the discovery of new types.

The remaining of this paper is organized as follows. Section II provides a brief introduction to the related work. A detailed description of the data, the N-gram feature extraction method and the classification model are presented in Section III. Section IV summarized and discussed the Experimental results. Finally, we conclude the paper and discuss future work in Section V.

II. Related Work

This paper focused on detecting honeypot smart contracts by using machine learning and data mining method, three types of research related to this work can be found in the literature. The most closely related works are reference [12] and [13], both focused on detecting honeypot by using different methods. Paper [12] presents the first systematic analysis of honeypot smart contracts, by investigating their prevalence, behavior, and impact on the Ethereum blockchain. They provide a taxonomy of honeypot techniques and build a honeypot detection tool based symbolic execution method and some well-defined heuristics. The advantage of this tool is that it can achieve a good identification effect for honeypots of a given type. But it is not conducive to discovering new categories. Paper [13] presents a data science approach based on contract transaction behavior. They analyzed the fund movement between the contract creator, the contract, the sender of the transaction and other participants. Based on the results and some other features, they trained a classification model. This study allows us to see the behavioral characteristics of many honeypots, but honeypot detection that relies on behavioral characteristics also has obvious disadvantages. On the one hand, as mentioned in the paper, many honeypots do not have enough transaction records, which makes feature extraction very difficult. On the other hand, when there are enough transactions for a honeypot, it also means that more than one user may have been trapped. In order to overcome the shortcomings, we proposed a honeypot recognition method based on bytecode characteristics. This not only facilitates the discovery of potential honeypot categories but also enables early warning of honeypot when the contract is created.

Since honeypot can be regarded as a certain vulnerability (attack) of smart contract, there are a lot of researches related to the identification and repair of vulnerabilities. Paper [16] analyzed the security vulnerabilities of Ethereum smart contracts, providing a taxonomy of common programming pitfalls that may lead to vulnerabilities, and show a series of attacks that exploit these vulnerabilities. Paper [9] introduces several new security problems and propose ways to enhance the operational semantics of Ethereum to make contracts less vulnerable. Besides, the authors build a symbolic execution tool to find potential security bugs for developers. Because smart contracts are so important, several tools have been proposed to identify potential vulnerabilities, for example, SmartCheck [17], ReGuard [10], ContractFuzzer [18]. In the Ethereum blockchain, a smart contract is a series of bytecode of Ethereum Virtual Machine (EVM). The EVM is a simple stack-based virtual machine that supports an instruction set of opcodes. In order to rigorously verify the security of smart contracts, paper [19] presents the first complete small-step semantics of EVM bytecode and defines a number of central security properties for smart contracts.

Finally, if honeypot is regarded as a scam, many studies are focusing on the identification of scam in the blockchain ecosystem. Our previous studies [20, 21] proposed a machine learning framework to detect smart Ponzi scheme, a classic scam under the veil of smart contract on Ethereum. Different from our study, paper [22] presents a comprehensive survey of Ponzi schemes on Ethereum, and analyses their behavior and their impact from various viewpoints. As Bitcoin is more valuable than Ethereum, a lot of scams were targeted at the bitcoin ecosystem. Paper [23] analyzes the problem of Ponzi scheme on Bitcoin and applies data mining techniques to detect Bitcoin addresses related to Ponzi schemes. Paper [24] discusses several Bitcoin-based scams, including Ponzi schemes, mining scams, scam wallets, and fraudulent exchanges. Furthermore, the authors present evidence that the most successful scams depend on large contributions from a very small number of victims. Because of the anonymity of bitcoin, it is often necessary to identify entities in bitcoin in order to identify scams. Paper [25] presents a novel approach for reducing the anonymity of the Bitcoin blockchain by using Supervised Machine Learning to predict the type of yet-unidentified entities.

III. Data, Feature Extraction and Classification Model

In order to establish a data mining and machine learning method for the detection of honeypot contracts, we need to collect enough positive and negative samples (for simplicity, we call honeypot contracts as positive samples and other contracts as negative samples) and provide an effective feature extraction method. This section provides an overview of the data collected, the N-Gram feature extraction method and the proposed classification method.

A. Data

In order to collect enough samples, we crawled 857 addresses of detected honeypot contracts from the *HONEYBADGER* project[1]. (In the following, for the sake of narrative convenience, we treat the contract as equivalent to its corresponding address.) These contracts are labeled with honeypot according to the previous study [12]. The authors developed a tool that detects whether the contract is a honeypot, and for each contract, they verified the results by manually inspecting the contract source code. Furthermore, they divided them into eight categories based on the technology used in the honeypot contract, including Hidden State Update (HSU), Inheritance Disorder (ID), Uninitialised Struct (US), Straw Man Contract (SMC), Balance Disorder (BD), Hidden Transfer (HD), Skip Empty String Literal (SESL), and Type Deduction Overflow (TDO). In order to save space, we will not introduce the specific meaning of each category. Readers are advised to read [12] for details. We treat these contracts as positive samples in our study. For negative samples, We launch an Ethereum client, Parity[2], to download the ledger of Ethereum. In this way, we collected all the contracts and their corresponding bytecode created between the very first block to block 8, 099, 999 (July 6th, 2019). there are 16, 609, 273 contracts in total. It is a very huge number, which will make the class extremely imbalance. Fortunately, many contracts on Ethereum replicate each other [26], this suggests that we can remove duplicates from the contract before further analysis. Thus, for all contracts with the same bytecode, we keep only one. By doing this, we obtained 218, 250 negative samples.

B. Feature Extraction

Features are very important for a classification model. In order to detect honeypot at its creation, we need to extract features from the bytecode of the contract, because when the contract is created, only the bytecode is required and must be provided. For the convenience of feature extraction, we convert the bytecode of the contract into opcodes by using the *evm* toolkit of geth[3]. Figure 1 shows an opcode file fragment of a contract[4]. As can be seen, the opcode file contains three types of information: instruction address, opcode (i.e., instruction), operation and operand (some instructions may have no operand).

[1] https://github.com/christoftorres/HoneyBadger

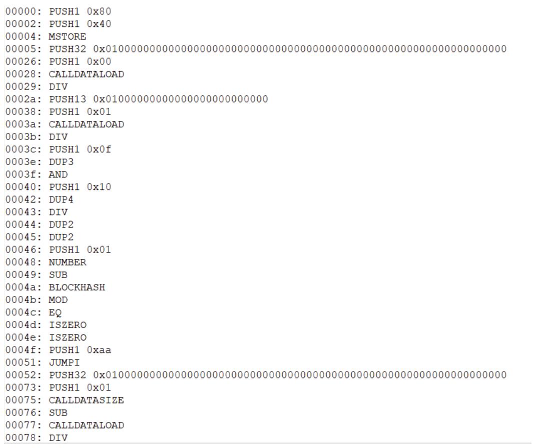
[2] https://www.parity.io/ethereum/

[3] https://geth.ethereum.org/downloads/

[4] 0x00000a8e943f66d0e20b107333a024da0c865dad

TABLE I
Statistics of opcode use frequency.

Category	Number
Mean	3486000
Standard deviation	9158000
Min	604
25% quantile	22053
50% quantile	132333
75% quantile	1945800
Max	78380000



```
00000: PUSH1 0x80
00002: PUSH1 0x40
00004: MSTORE
00005: PUSH32 0x0100000000000000000000000000000000000000000000000000000000000000
00026: PUSH1 0x00
00028: CALLDATALOAD
00029: DIV
0002a: PUSH13 0x01000000000000000000000000
00038: PUSH1 0x01
0003a: CALLDATALOAD
0003b: DIV
0003c: PUSH1 0x0f
0003e: DUP3
0003f: AND
00040: PUSH1 0x10
00042: DUP4
00043: DIV
00044: DUP2
00045: DUP2
00046: PUSH1 0x01
00048: NUMBER
00049: SUB
0004a: BLOCKHASH
0004b: MOD
0004c: EQ
0004d: ISZERO
0004e: ISZERO
0004f: PUSH1 0xaa
00051: JUMPI
00052: PUSH32 0x0100000000000000000000000000000000000000000000000000000000000000
00073: PUSH1 0x01
00075: CALLDATASIZE
00076: SUB
00077: CALLDATALOAD
00078: DIV
```

Fig. 1. Opcode file fragment of a smart contract.

In order to extract the n-gram features, we ignore the instruction address and operand in the file. It turns out that there are 146 opcodes. To get a preliminary understanding of the use of opcode, we analyzed their use frequency. Figure 2 shows the 10 most used opcodes and their corresponding use ratios. Table I shows some statistics of the use frequency of all the opcodes. These data show that there are significant differences in the frequency of opcodes used in contracts, and it is feasible to judge the type of contracts from the bytecode of contracts. In fact, our previous works have shown that opcodes of a contract are useful in determining whether a contract is a Ponzi scheme [20, 21]. However, in our previous work, only the uni-gram features (i.e, the frequency of the opcodes) is considered. This approach does not take into account the strong contextual relevance of the code. Thus, in this paper, we also used bi-gram to avoid this problem. Specifically, bi-gram features are the frequency of opcode combination. Besides, as a more objective reflection of usage characteristics, we also introduced the use ratio of each opcode and each combination.

C. Classification Model

In this section, we introduce the adopted classification model LightGBM. LightGBM is a novel GBDT (Gradient

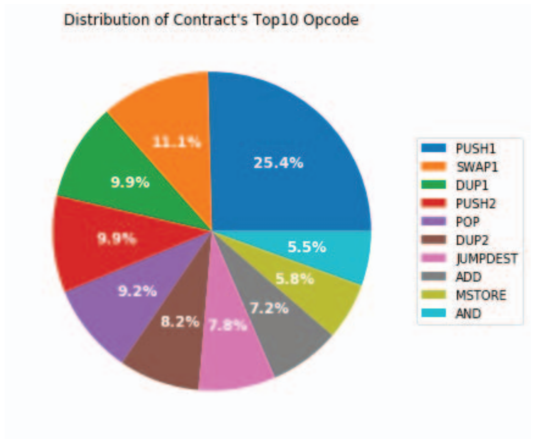


Fig. 2. The top 10 opcodes and its use ratios.

Boosting Decision Tree) algorithm, proposed in [27]. It has successfully used in many different kinds of data mining tasks, such as classification, regression and ordering [27, 28, 29]. In short, GBDT is an integrated learning method using the CART regression tree model in the Boosting framework.

In each iteration of GBDT, assume that the strong learner obtained by the previous iteration is $h_{t-1}(x)$, the loss function is $L(f(x), h_{t-1}(x))$, then our aim for the current iteration is to find a week learner using CART regression tree model which denoted as $h_t(x)$, to minimize the formula $L(f(x), h_{t-1}(x) + h_t(x))$. Here [27] come up with an idea to constrcut a CART regression tree by using the negative gradient of the loss function to approximate the loss function for the current iteration. Specifically, in iteration t, the negative gradient for sample i can be represented as below:

$$r_{ti} = \frac{\partial L(y_i, h_{t-1}(x_i))}{\partial h_{t-1}(x_i)}.$$

As we are facing an binary classification problem, we can use the Log-likelihood loss as loss function

$$L(y, h(x)) = log(1 + exp(-yh(x))),$$

where $y \in [-1, 1]$. by using this loss function, we can simplify the negative gradient of sample as below:

$$r_{ti} = -\frac{\partial L(y_i, h_{t-1}(x_i))}{\partial h_{t-1}(x_i)} = \frac{y_i}{1 + exp(y_i h(x_i))},$$

where $i = 1, 2, \cdots, m$.

By using the formula, we can know that, for those samples with little gradient, we reached a state where only a little training loss is made by them. LightGBM chooses to remove these small gradient samples from the training set to make the model pay more attention to those samples which cause great Loss. This technique is called Gradient-based One-Side Sampling (GOSS). The algorithm in detail can be found in [27].

When constructing the CART regression tree, we need to continue to choose features to make the tree grow. When a feature is selected, we have to choose an optimized split point to split the feature. The trivial method is pre-sorted and tests one by one, which is very time-consuming. In lightGBM, it uses the histogram method: put n continuous feature values into k bins ($k << d$) and choose the split point in the k bins. Because this method greatly reduces the range of possible points to choose from, both the memory consumption and training speed are better than the trivial method. In the test of actual data sets, it is shown that the discretization of the split point has a little or better effect on the final accuracy. The reason is that the CART tree itself is a weak learner, and the adoption of a histogram algorithm will play a regularization effect and effectively prevent the overfitting of the model.

It can be seen that compared with the traditional GBDT model, LightGBM model has the following advantages:

- It uses gradient-based one-side Sampling (GOSS) technology. It avoids the adverse effects of some low gradient long-tail distributions by discarding many small gradients.
- A histogram is adopted to replace the pre-sorted method in the optimization stages. This substitution greatly reduces the memory and running time required by the model, and can reduce the risk of overfitting.

In addition, LightGBM also has the following advantages:

- Exclusive Feature Bundling. The number of features can be greatly reduced by binding mutual exclusion features, and the splitting selection point can be guaranteed to be selected at a relatively appropriate point.
- Compared with XGBoost [30], LightGBM adopts leaf-wise instead of a level-wise decision tree growth strategy. It can control the growth of the tree more precisely and control the risk of overfitting.

We also have tried some other classification models.However the result is worse than LightGBM obviously.Thus we choose this model and tune it to perform better.

IV. Experimental results and Discussion

In this section, we present our experimental results. ## Dataset As discussed in Section III-A, we obtained 616 positive samples (honeypot contract) and 218, 250 negative samples (non-honeypot contracts). Table II shows the number of honeypot contracts in each category. It can be seen that most honeypot contracts belong to the Hidden State Update (HSU), accounting for more than 50%. However, for some categories, such as Type Deduction Overflow (TDO), Skip Empty String Literal (SESL), Hidden Transfer (HT), and Balance Disorder (BD), the sample size is relatively small, and it is very difficult for the model to identify these types because too few samples do not provide enough information for the model to learn its characteristics. Therefore, the main objective of our experiment is at the creation of the contract, 1) if there is a high probability that the sample is a honeypot contract, then 2) further judge its subcategories.

TABLE II
NUMBER OF HONEY IN EACH CATEGORY.

HSU	ID	US	SMC	BD	HT	SESL	TDO
350	74	64	64	27	22	10	5

A. Experimental setting

In order to reflect the effect of the model more accurately and avoid the contingency caused by the problem of train and test set partition, this paper adopts the evaluation method of *k*-fold cross-validation. That is, we divide the whole data set into *k* parts, each time we choose one part as the test set in turn, and the remaining *k*-1 part as the training set, test k times in total and take the mean value of *k* test as the final result. In this experiment, we set *k*=3 as the number of positive samples is relatively small. In other words, 66.7% of data are used for training to evaluate the remaining 33.3% samples.

As the imbalance ratio (the number of negative samples divided by the number of positive samples) is too large, it is not conducive to learning, so we need to properly conduct sampling operations on negative samples. However, in order to ensure that our model can be applied to different sets of negative samples, we carried out undersampling of negative samples several times and constructed a model with the sampled negative samples and positive samples each time. The final results are the mean of all the model results.

B. Evaluation Metrics

Because of the class imbalance problem, we choose *Precision*, *Recall*, *AUC*, and *F1* as evaluation metrics. Specifically, we can divide the samples into four groups: TP (True Positive), FP (False Positive), TN (True Negative) and FN (False Negative) according to the combination of its true class and predicted class. Then,

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

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

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}.$$

AUC is the area under the ROC curve, which formed by FPR and TPR under different thresholds, where

$$FPR = \frac{FP}{FP + TN}$$

and

$$TPR = \frac{TP}{TP + FN}.$$

These metrics were used to evaluate the experimental results from different dimensions: precision focused on evaluating the reliability of the samples that the model identified as a honeypot, and recall was used to evaluate whether the model could find the honeypot address as much as possible. These two metrics often show the opposite changes. F1 is a comprehensive consideration of these two indicators, while AUC can reflect the performance of the model without considering the threshold.

TABLE III
NUMBER OF FEATURES BEFORE AND AFTER SELECTION.

Category	Initial.Number	After.Selection
Unigram	295	169
Unigram+Bigram	32713	1409
Unigram+Bigram+Trigram	207899	3913

C. Experimental Results

1) Feature selection: The second column in Table III shows the initial number of features in different categories, as can be seen, the feature family is very large. However, most of these features are extremely sparse, and the inclusion of them into the feature family does not improve the effect of the model, thus we need to do the necessary selection of the constructed features. Specifically, we remove all these features with *NA* elements accounting for more than 90%. By doing this, the number of feature families becomes acceptable (The third column in Table III). In the following experiments, only those selected features are used.

2) Undersampling of negative samples: Because of the small sample size of honeypot contracts and a large number of other contracts, we face a serious class imbalance problem. In order to avoid the influence of class imbalance on the training effect of the model, we choose undersampling of negative samples, which is commonly used in class imbalance problems [31], to improve the training effect of our model. How to select the proportion of undersampling is also a problem to be considered. In the subsequent experiments, we tested the influence of different positive and negative sample ratios on the model results, so as to determine the appropriate sample proportions. Here we define the ratio of positive and negative samples as:

$$Unbalance = \frac{number\ of\ non\_honeypot\ contract}{number\ of\ honeypot\ contract}.$$

3) Binary Classification of Honeypot: For the binary classification of a honeypot, we made a comparative test on the combination of uni-gram, bi-gram, and tri-gram features, and the results of the cross-validation test on different metrics are shown in Table IV.

As can be seen from Table IV that the unigram features have achieved good results, and the combination of bigram further improves the effect of the model, at the cost of not too

TABLE IV
THE PERFORMANCE OF FEATURE COMBINATION. (UNBALANCE=10)

Unbalance	Unigram	Unigram+Bigram	Unigram+Bigram+Trigram
Precision	0.9641	0.9709	0.9929
Recall	0.8702	0.8962	0.8897
F1	0.9147	0.9320	0.9385
AUC	0.9914	0.9928	0.9926

many features. In the unigram+bigram model, all the metrics are further improved, and the model itself is not very large. However, adding trigram features do not significantly improve the model, since the results have been excellent, there is not much room for improvement. On the other hand, a large number of features are added to the model, which reduces the generalization ability of the model to some extent. Therefore, we believe that unigram+bigram is a better choice.

4) Feature importance in Binary Classification: Figure 3 shows the importance of the features in the binary classification of honeypot. The suffix *_ratio* of the feature represents the ratio of the corresponding opcode (or the combination) in the opcode file. It is hard to explain why some combinations of opcode are more useful in discriminations, but there are two results worth noting. On the one hand, it can be seen from the figure that the bi-gram features are better than uni-gram features in the classification. This result indicates that it makes sense for us to consider the code contextual relevance. On the other hand, the opcode ratios, not the absolute quantities, are more useful in the model.

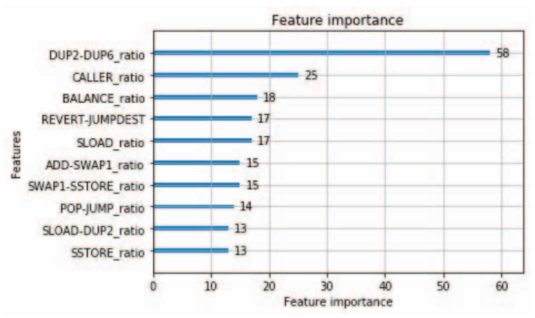


Fig. 3. Opcode file fragment of a smart contract.

D. Multiple Classification of Honeypot

In order to better understand the performance of the model, we compared the performance of the model under different categories and different undersampling ratios. Table V shows the results.

From Table V, we can obtain the following observations.

- For honeypot contract classification, it is not that the more balanced the data set is, the better the performance of the model will be. It can be seen that the performance of the model is poor when unbalance=1. In particular, we can see that for the four types of honeypots with relatively few category samples, the model completely lost the ability to identify them. One possible reason for this result may be that when the numbers of positive and negative samples are equal, the negative samples are oversampled and too much information about their description is lost. Considering that when unbalance=1, i.e., sample negative samples to the same size as the positive, the total data set size of the model built for those with a small sample size is no more than 60 records of data, it is hard to expect the model to learn anything useful from such a small data set.
- When the imbalance ratio becomes too high, the performance of the model also begins to decline. By comparing the results of unbalance=10 and unbalance=100, it can be found that the overall performance of unbalance=10 is significantly better than that of the control group. The reason is that when the imbalance ratio increases by 10 times, the effect of the imbalance in the data set begins to emerge gradually. Therefore, combining with the above observation, we can determine from the experimental results that the unbalance=10 group is the best.
- The model is invalid for honeypot categories with too few samples, i.e., SESL and TRO. From the results of the three control groups, we can see that although repeated experiments were conducted in each control group, the model was still very unstable in the two honeypot categories, and this randomness made us feel that the results were not very reliable. We need to see that the number of positive samples in the two categories is not more than 10, and with such a small number of positive samples, we think it is not a good choice to use the model to identify them.
- It can be seen that in the groups with unbalance=10 or unbalance=100, the classification effect of the model for honeypot categories with a large number of samples is very good. All the AUC value is above 0.95, and in the case of unbalance = 10, the recall value for the six categories with relatively more samples are still in a high level, this result indicates that our model can identify most of these honeypot contracts in cross-validation. Furthermore, the precision is also very high, which means that the misjudgment rate of our model for negative samples is also very low. It is also noted that the model still performs well even on the data set of unbalance=100, which is already very imbalanced, indicating that the model performs well even on imbalance data set.

V. CONCLUSION AND FUTURE WORK

In this paper, we mainly analyzed honeypot contracts on Ethereum. Our goal was to identify honeypot contracts with high accuracy at the time of its creation. By downloading 16, 609, 273 smart contracts' bytecode on Ethereum, we decompiled it and obtained all the operation code (opcode). In the process of analyzing the operation code and build the model, we construct a series of N-Gram based features and using a feature selection method to dropout those useless features. After that, we build a LightGBM model to detect

TABLE V
THE PERFORMANCE OF MULTIPLE CLASSIFICATION.

Metrics	HSU	ID	US	SMC	BD	HT	SESL	TDO
Unbalance=1								
Number of samples	350.0000	74.0000	64.0000	64.0000	27.0000	22.0000	10.0000	5.0
Precision	0.9627	0.8649	0.9298	0.8340	0.5000	0.5000	0.5000	0.5
Recall	0.9638	0.8925	0.9394	0.7903	0.5000	0.5000	0.5000	0.5
F1	0.9632	0.8785	0.9346	0.8116	0.5000	0.5000	0.5000	0.5
AUC	0.9922	0.9222	0.9907	0.9287	0.5000	0.5000	0.5000	0.5
Unbalance=10								
Number of samples	350.0000	74.0000	64.0000	64.0000	27.0000	22.0000	10.0000	5.0
Precision	1.0000	1.0000	1.0000	0.9773	1.0000	1.0000	0.0000	0.0
Recall	0.9257	0.8239	0.9055	0.9221	1.0000	0.9107	0.8889	0.0
F1	0.9614	0.9034	0.9504	0.9489	1.0000	0.9533	0.0000	0.0
AUC	0.9952	0.9766	0.9872	0.9791	0.9994	0.9777	1.0000	0.5
Unbalance=100								
Number of samples	350.0000	74.0000	64.0000	64.0000	27.0000	22.0000	10.0000	5.0
Precision	1.0000	1.0000	0.9375	1.0000	0.9375	1.0000	0.5000	1.0
Recall	0.8770	0.6756	0.6883	0.8579	0.9630	0.6131	0.6944	1.0
F1	0.9345	0.8064	0.7938	0.9235	0.9501	0.7601	0.5814	1.0
AUC	0.9854	0.9698	0.9838	0.9660	0.9987	0.9919	0.9884	1.0

the honeypot contract. The experimental results showed that the model with unigram+bigram features has the F1 value of 0.93 and the AUC value of 0.99 for the binary classification of honeypot contract recognition. Furthermore, the honeypot contract is further classified using the model, we make some appropriate exploration of the imbalance of the data set. When an appropriate unbalance rate is used, the model can achieve an excellent effect that the AUC exceeds 0.95 in most honeypot contract types with the precision and recall are also good. This further proves the validity of our model. In the future, we plan to further improve our work. by further exploring the construction method of n-gram features and the feature filtering methods. Besides, we plan to analyze the historical information of the creator (including their trading behavior, contract creation behavior, etc.) and combine the two parts to get a more accurate classification model for detecting honeypot contracts.

ACKNOWLEDGMENT

The work described in this paper was supported by the National Key R&D Program of China (2018YFB0204303), the National Natural Science Foundation of China (61722214, U1611261, 61872392), the Natural Science Foundation of Guangdong (2018B030312002), project funded by China Postdoctoral Science Foundation (Grant no. 2019TQ0372, 2019M660223), the Natural Science Foundation of Guangdong (2019B020214006), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant NO. 2016ZT06D211 and Pearl River S&T Nova Program of Guangzhou under Grant NO. 201906010008. Zhiguang Chen and Zibin Zheng are corresponding authors.

REFERENCES

[1] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *Ethereum white paper*, 2014.

[2] N. Szabo, "Smart contracts: Building blocks for digital markets," Sep. 1996. [Online]. Available: http://www.fon.hum.uva.nl/

[3] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[4] F. Xiang, W. Huaimin, S. Peichang, F. Yingwei, and W. Yijie, "Jcledger: A blockchain based distributed ledger for jointcloud computing," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 289–293.

[5] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. IEEE, 2016, pp. 1–3.

[6] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaría, "Blockchain and smart contracts for insurance: Is the technology mature enough?" *Future Internet*, vol. 10, no. 2, p. 20, 2018.

[7] D. Siegel, "Understanding the dao attack," Jun. 2016. [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists

[8] S. Petrov., "Another parity wallet hack explained," Nov. 2017. [Online]. Available: https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c

[9] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 254–269.

[10] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: finding reentrancy bugs in smart contracts," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM,

2018, pp. 65–68.

[11] L. Breindenbach, P. Daian, F. Tramèr, and A. Juels, "Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1335–1352.

[12] C. Ferreira Torres, M. Steichen *et al.*, "The art of the scam: Demystifying honeypots in ethereum smart contracts," in *USENIX Security Symposium, Santa Clara, 14-16 August 2019*, 2019.

[13] R. Camino, C. F. Torres, and R. State, "A data science approach for honeypot detection in ethereum," *arXiv preprint arXiv:1910.01449*, 2019.

[14] A. Sherbachev, "Hacking the hackers: Honeypots on ethereum network," Aug 2018. [Online]. Available: https://hackernoon.com/hacking-the-hackers-honeypots-on-ethereum-network-5baa35a13577

[15] A. Sherbuck, "Dissecting an ethereum honey pot," Feb. 2018. [Online]. Available: https://medium.com/coinmonks/dissecting-an-ethereum-honey-pot-7102d7def5e0

[16] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *International Conference on Principles of Security and Trust*. Springer, 2017, pp. 164–186.

[17] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2018, pp. 9–16.

[18] B. Jiang, Y. Liu, and W. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 259–269.

[19] I. Grishchenko, M. Maffei, and C. Schneidewind, "A semantic framework for the security analysis of ethereum smart contracts," in *International Conference on Principles of Security and Trust*. Springer, 2018, pp. 243–269.

[20] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 2018, pp. 1409–1418.

[21] W. Chen, Z. Zheng, E. C.-H. Ngai, P. Zheng, and Y. Zhou, "Exploiting blockchain data to detect smart ponzi schemes on ethereum," *IEEE Access*, vol. 7, pp. 37 575–37 586, 2019.

[22] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, "Dissecting ponzi schemes on ethereum: identification, analysis, and impact," *Future Generation Computer Systems*, vol. 102, pp. 259–277, 2020.

[23] M. Bartoletti, B. Pes, and S. Serusi, "Data mining for detecting bitcoin ponzi schemes," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 75–84.

[24] M. Vasek and T. Moore, "Theres no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams," in *International conference on financial cryptography and data security*. Springer, 2015, pp. 44–61.

[25] M. A. Harlev, H. Sun Yin, K. C. Langenheldt, R. Mukkamala, and R. Vatrapu, "Breaking bad: De-anonymising entity types on the bitcoin blockchain using supervised machine learning," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[26] H. Liu, Z. Yang, C. Liu, Y. Jiang, W. Zhao, and J. Sun, "Eclone: Detect semantic clones in ethereum via symbolic transaction sketch," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 900–903.

[27] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

[28] E.-A. Minastireanu and G. Mesnita, "Light gbm machine learning algorithm to online click fraud detection," *J. Inform. Assur. Cybersecur*, vol. 2019, 2019.

[29] X. Sun, M. Liu, and Z. Sima, "A novel cryptocurrency price trend forecasting model based on lightgbm," *Finance Research Letters*, 2018.

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[31] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.