Towards Auto Contract Generation and Ensemble-based Smart Contract Vulnerability Detection

Original Scientific Paper

K. Lakshminarayana
Puducherry Technological University,
Ph.D Scholor, Department of Computer Science and Engineering,
Puducherry, India
kodavali.lakshmi@pec.edu

K. Sathiyamurthy
Puducherry Technological University,
Faculty of Computer Science and Engineering,
Puducherry, India
sathiyamurthyk@ptuniv.edu.in

Abstract – *Smart contracts (SC) are computer programs that are major components of Blockchain. The "intelligent contract" is made up of the rules accepted by the parties concerned. When the transactions started by the parties obey these established rules, then only their transactions will be completed without the involvement of a third party. Because of the simplicity and succinct nature of the solidity language, most smart contracts are written in this language. Smart contracts have two limitations, which are vulnerabilities in SC and that smart contracts can't be understood by all stakeholders, especially non-technical people who are involved in the business, since they are written in a programming language. Hence, the proposed paper used the XGBoost model and BPMN (Business Process Modeling Notation) tool to solve the first and second limitations of the SC respectively. Attackers are drawn to attention because of the popularity and fragility of the Solidity language. Once smart contracts have been launched, they can't be changed. If that smart contract is vulnerable, attackers may then cash it. BPMN is used to represent business rules or contracts in graphical notation, so everyone involved in the business can understand the business rules. This BPMN diagram can be converted into a smart contract template through the BPMN-SOL tool. A few publications and existing tools exist on smart contract vulnerability detection, but they require more time to forecast and interpretation of vulnerability causes is also difficult. Thus, the proposed model experimented with several deep learning approaches and improved F1 score results by an average of 2% using the XGBoost model based on the ensemble technique to detect vulnerabilities of SCs, which are: Denial of Service (DOS), Unchecked external call, Re-entrancy, and Origin of Transaction. This paper also combined two important features to construct a data set, which are code snippets and n-grams.*

Keywords*: Blockchain, Smart Contract Vulnerabilities, Ethereum, Machine Learning, Ensemble Model, BPMN*

1. INTRODUCTION

A Blockchain works in a decentralized environment and it has a sequence of blocks that are connected using cryptographic techniques [1]. As shown in Figure 1, each block consists of transaction data, a hash of the preceding block, and a timestamp. By its design, blockchain is unsusceptible to changing data by its design. In a Blockchain, transactions between two parties are recorded in an efficient, verifiable, and permanent way [2]. Such a Blockchain can present an innovative solution to long-standing problems of security related to data storage in centralized systems. Blockchain can be considered the new face of cloud computing and is expected to reshape organizational and individual behavioral models.

An important feature of a Blockchain is, that it is a distributed database. It means no centralized database or server exists Instead, the same Blockchain is duplicated on every node of the network. Each node in the system receives a duplicate of Blockchain where all chunks have a grade of dealings in an encrypted format using asymmetric keys. Due to the complexity of mathematical formulas used in cryptography techniques, it is practically impossible to guess the keys and crack the transactions. The sender can use his private key to encode a message to be sent, and the recipient can use his public key to decrypt the message. Every new transaction is broadcast and updated to all the network nodes to maintain a consistent database across the whole Blockchain network [3]. Bitcoins are the major general Blockchain stand in the world. Ethereum is another popular Blockchain that introduces smart contracts.

Smart Contracts (SC) are the programs for predefined rules which are deployed into the Blockchain and these programs execute automatically to make sure that every transaction has to satisfy the predefined conditions to complete the transaction. Smart Contracts work based on simple conditional statements. Smart Contracts are playing a more vital role in business among a group of untrusted people, where every transaction can be completed according to rules agreed by all business stakeholders without the involvement of third-party verification [4]. Initially, SC basis codes are written in a high-level language, for example, Solidity by designers. The source code is compiled into byte codes (EVM code) by a compiler, it is in a hexadecimal arrangement. These byte codes can be converted into EVM instructions and are called opcodes [5].

Broadly three reasons attackers are focusing on smart contracts: first, the smart contracts of Ethereum are mainly money oriented transactions; secondly, after being deployed into the Blockchain, it is not possible to alter vulnerable SC; and finally, smart contracts have no defined measures to determine the quality of smart contracts [6]. Many smart contract assaults in 2016 led to large money losses (multi-million dollar losses) as a result of vulnerabilities in SCs. In smart contracts on Ethereum, it's currently worth focusing on automated deep study models to effectively detect SC vulnerabilities [7], especially in financial matters such as money transfers and more complicated code.

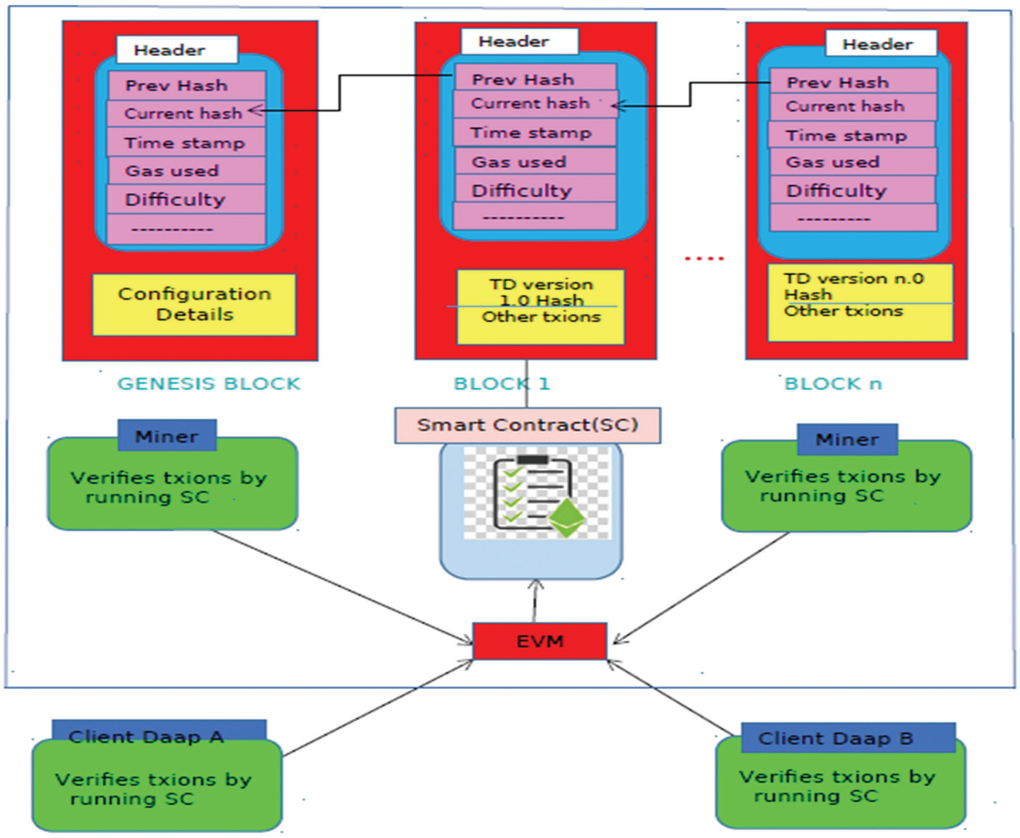


Fig 1. Ethereum Blockchain Network

Smart contract vulnerabilities are divided into four categories, 1. security, 2. functional, 3. developmental, and 4. operational [11]. Security concerns include reentrancy, external contract, DOS - denial of service, the use of tx.origin, unchecked external call, and usage of send() in place of transfer(). Functional vulnerabilities are locked money, integer division, integer underflow, integer overflow, unsafe interface type, and reliance on time stamps. Development concerns involve infringement of token API, private modifier, non-compiler version fixation, violation of the style guide, duplicated back function, and degree of implied visibility. Finally, operational problems include byte array and expensive loop vulnerabilities. The major focus of this article is on security concerns that consist of unchecked external calls, re-entrancy, DOS, and the origin of thetransaction.

Reentrancy vulnerability[10,15]: In which they can take on the control flow and modify your data that is not expected by the call function. There are numerous shapes this bug class might take. The initial release of this problem is single-function reentrancy in which functions can be called repeatedly before the original call is made. This might lead to harmful interactions between the multiple calls of the function. The other release of this problem is the re-entrancy of a cross-function, in which an attacker can also attack the same state with two functions [8]. As re-entrancy can occur through several functions and even between different contracts, a single function will not be enough to prevent re-entrancy. Instead, all internal work (i.e. state modifications) has been recommended first, and then the external function has to be called to prevent re-entrancy vulnerability.

DOS vulnerability: A specific quantity of gas (transaction fee) is required for the execution of smart contract functions. The Ethereum network establishes a gas limit for every block and should not exceed that amount of all transactions in a block. Programmable statements in smart contracts that cause DOS (Denial of Service), if the gas limit is exceeded when these statements are executed. DOS vulnerability may be caused in scenarios such as 1) A loop variable with a value higher than or less than 382 depends more or less on the network gas limit. 2) Work with unfamiliar array sizes.

Vulnerability of transaction origin: The keyword called "tx.origin" in solidity language indicates the address of the account that began a transaction. For example, consider the sequence of call series X--> Y and Y--> Z. From the Z viewpoint, msg.sender is Y, and tx.origin is X. The "tx.origin" keyword can sometimes lead to dubiety, therefore try to avoid the use of "tx.origin" for authorization. Instead, it can be handled with msg.sender.

Machine Learning (ML) algorithms are classified by learning style, consisting of supervised, semi-supervised, and unsupervised. In supervised learning, input or training data has a predefined label. Initially, a classifier has to be designed with appropriate layers to train on training data and to predict the label of test data. The classifier has to be tuned well to get a good level of prediction accuracy. In unsupervised learning, training data does not have a label, hence the classifier is designed to cluster unsorted data based on similarities and variance.

Wang Wei et al[5] presented an automated vulnerability detection model for smart contracts using the XGBoost machine learning model by extracting bigram features from SC opcodes. The limitation of this paper is that bigrams (2-gram) may not always be suitable to detect all types of vulnerabilities, because some vulnerabilities may require more than bigram features. Interpretation of opcodes is more difficult compared with high-level source code. The proposed framework uses an ensemble-based XGBoost supervised model [5] to perform multi-label classification to detect SC vulnerabilities with the help of a data set created by n-gram features which are extracted from high-level SC source code. Ensemble algorithms join the outcomes of a set of simple and feeble models to get better predictions than those that are obtained using a single learning algorithm. The association of this remaining paper is organized as follows: Part 2 gives literature work on smart contract vulnerability detection systems using machine learning; part 3 explains the proposed work for vulnerability detection using the XGBoost learning model; part 4 demonstrates experiment details and comparison outcomes; and finally, conclusion and future scope will be in part 5.

2. LITERATURE WORK

Recently, a few papers [5,6,9,10,11] were published on behalf of smart contract susceptibility detection using different ML techniques. Jian-Wei Liao et al. [6] present smart contract susceptibility detection using machine learning and fuzz testing techniques. The Authors used existing SC vulnerability detection tools, which are Oyente and Remix, to label training data sets. These static detection tools are more time-consuming [12]. The Authors also stated that to detect vulnerabilities of SC, it requires SC skilled people or predefined patterns of vulnerabilities. They extracted features from SC opcodes to prepare the data set.

Pouyan Momeni et al [9] presented smart contract security analysis with machine learning techniques, and the authors used existing traditional tools, which are Mythril [13] and Slither to label the SCs that are present in the dataset. These SC are given to a solidity parser as an input, and it generates an AST (Abstract Syntax Tree). Processing of AST is straightforward and easy to interpret. Features can be extracted comfortably from this AST as its output. The traditional tools used in the paper have been taking more time to predict vulnerabilities[12]. Feng Mi et al. [14] presented a paper on the automatic detection of SC vulnerabilities using deep learning. The authors prepared a data set with extracted features from the SC byte code. Moreover, the interpretation of byte code makes it difficult to analyze results. Peng Qian et al. [7] proposed graph neural networks for smart contract vulnerability detection with the help of expert knowledge. The authors constructed a graph for the extracted patterns from a given SC. Authors developed an open-source tool to extract patterns or features. Hence, in the proposed work, the open source feature extraction tool has been used and tailored as per the proposed work requirements.

Lakshminarayana. K et al [15] experimented with basic classification methods, which are binary classification, multiclass classification, multi-label classification, and auto encoding techniques to detect smart contract vulnerabilities, which are reentrancy, DOS, and Tx.origin. The proposed paper also tries to improve detection results of the same vulnerabilities using a combination of two techniques, which are the contract snippets, n-gram features, and the XGBoost classification technique. Yiping Liu et al. [19] presented an SC vulnerability detection model based on symbolic execution by taking SC assembly code (opcode) as input to generate a control flow graph. Zhang L et al [20] presented an SC vulnerability detection model based on an information graph and ensemble technique. Input for this model is considering SC opcodes to find the critical opcode sequence for each vulnerability. But the proposed paper has been using high-level source code directly as input since SC source code is easier to trace the source of vulnerability than SC byte code or SC opcodes. Nowadays, deep learning techniques have been extensively used in many areas in real life, for example, to prevent COVID by tracing social distances [16], network behavior monitoring [17], to help programmers who feel it is difficult to learn by identifying their mistakes and suggesting corrections [18], and also for the detection of unusual activities in the health care sector, etc.

Zhenguang Liu et al. [10] present automated reentrancy detection for SC. This paper used BLSTM for the classification task. The authors propose contract snippets (keywords) to capture semantic information from a given SC. The limitation of this paper is that it considers every word of each line in an SC to prepare a feature set. This may increase the number of features, but it may lead to a reduction in classification accuracy. Wesley Joon-Wie Tann et al [11] presented the LSTM machine learning model for safer smart contracts. LSTM will consider a sequence of opcode features to detect SC vulnerabilities. But LSTM doesn't care about the inspection of data and control-flow possessions (ex: loops and function calls). K. Frantzet al[21] and Luciono B et al [22] proposed methods to convert BPMN diagrams to solidity smart contract templates. Authors in [21] developed a domain-specific language (DSL) called ADICO-Solidity DSL for conversion from BPMN to solidity code. Authors in [22] prepare a Petri net graph as an intermediate step from BNMN elements and then convert them to solidity code from Petri nets. Each of the above papers has its pros and cons. Hence, the proposed paper combines the pros of the above papers, which are the usage of a parser, contract snippets, and usage of n-grams to prepare the data set in the proposed SC vulnerability system, and also does work towards auto SC generation using the BPMN-SOL compiler [23,24].

3. PROPOSED SYSTEM

The general building of the proposed model is shown in Figs. 2(a) and 2(b). In the proposed system, a smart contract is supplied as input to the solidity parser. It performs preprocessing steps like removing comments and identifying functions and loops. From the parser output, code snippets (important keywords) and n-grams can be extracted as shown in Table1.

These n-grams play a major role in identifying vulnerabilities. Now it is possible to prepare a data set by considering n-grams as features, and they could be labeled according to the n-grams found in the SC. As shown in figure 3, different classes of vulnerability (C1 to C4) are correlated with different disjoint sets of n-gram features (N1 to N7). It applies to all solidity smart contracts if we provide high- level solidity source code as an input instead of byte code or opcode of smart contracts.

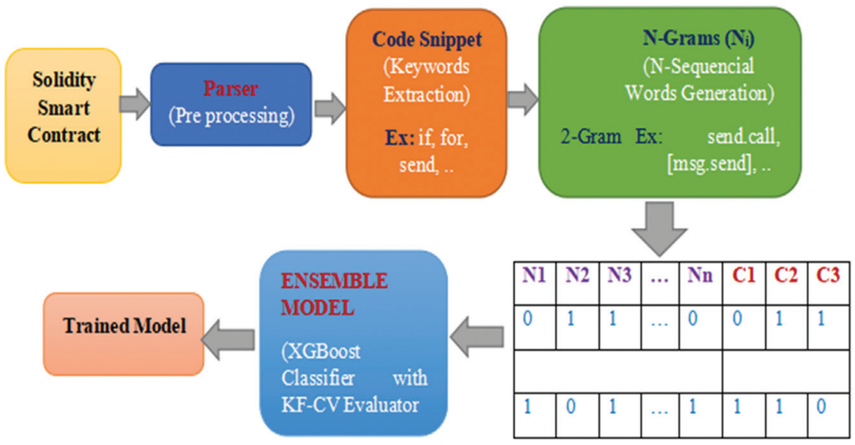


Fig 2(a). Training a Machine Learning Model

Table 1. Examples for Code Snippets and n-grams from smart contracts

Code Snippets (Key Words)	n-grains
if, ( ), msg, sender,	msg. sen der.ca 11.va lue, a ddr.transfer(*)
call, value, . , [,],	msg.sender, tx.origin( )9 solidity ^9
While, transfer, function,	funtion( ), [msg.sender]=0
return. require. revert.	msg.sender.transfer, addr.send("),
addr,	if(*addr.send(*)) revert, if(*( )),

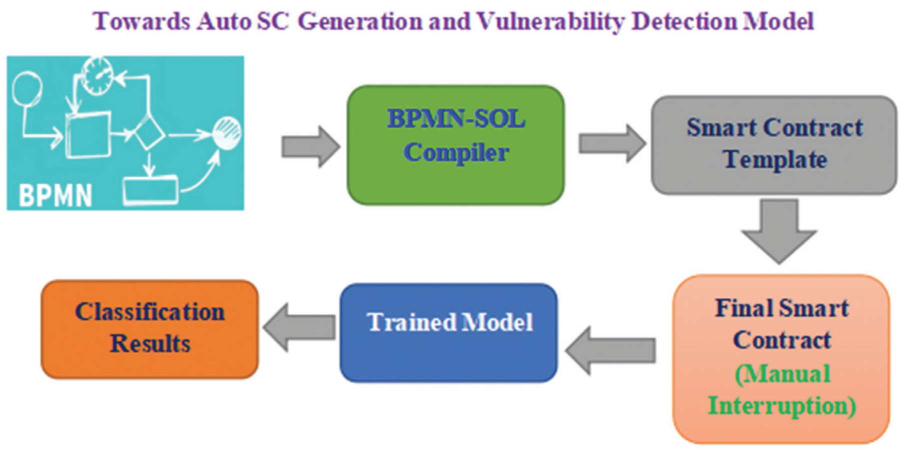
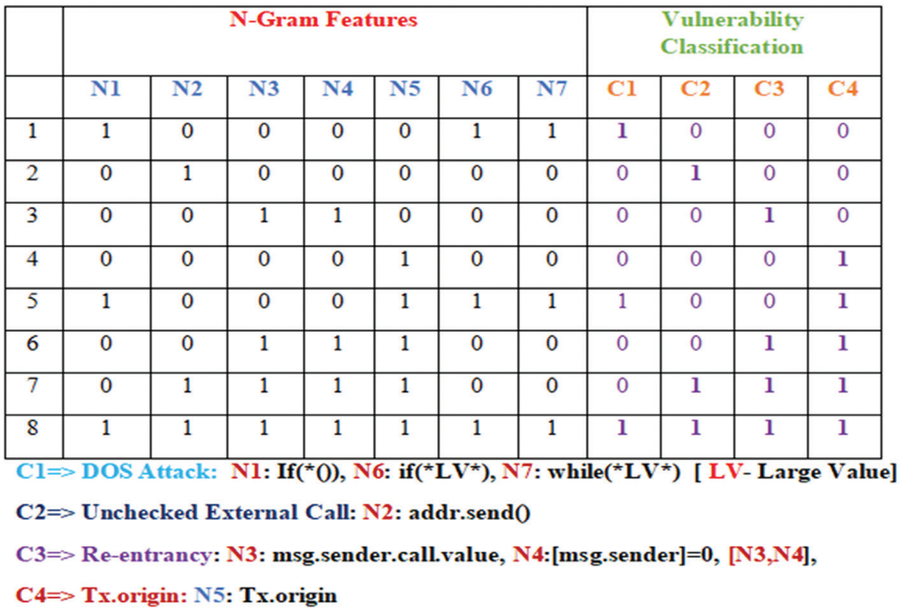

Towards Auto SC Generation and Vulnerability Detection Model
BPMN
BPMN-SOL Compiler
Smart Contract Template
Final Smart Contract (Manual Interruption)
Trained Model
Classification Results

Fig 2(b). Smart Contract Template generation and testing a Model



	N-Gram Features							Vulnerability Classification			
	N1	N2	N3	N4	N5	N6	N7	C1	C2	C3	C4
1	1	0	0	0	0	1	1	1	0	0	0
2	0	1	0	0	0	0	0	0	1	0	0
3	0	0	1	1	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0	0	0	0	1
5	1	0	0	0	1	1	1	1	0	0	1
6	0	0	1	1	1	0	0	0	0	1	1
7	0	1	1	1	1	0	0	0	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1

C1=> DOS Attack: N1: If(*()), N6: if(*LV*), N7: while(*LV*) [LV- Large Value]

C2=> Unchecked External Call: N2: addr.send()

C3=> Re-entrancy: N3: msg.sender.call.value, N4:[msg.sender]=0, [N3,N4],

C4=> Tx.origin: N5: Tx.origin

Fig 3. Data Set for multi-label classification to detect vulnerabilities

The data set is created after analyzing the smart contract source codes, which are collected from [25, 27] as per the algorithm shown in Figure4. In the literature on SC, many papers prepared data sets from the byte code of SC, which are difficult to analyze and difficult to confirm whether SC is vulnerable or not. Hence, this paper prepared datasets from high-level SC source code instead of from byte code or opcodes of SC.

Multi-label classification task: this requires one or more labels for each input sample as an output, and the outputs are required at the same time. The hypothesis is that the output labels depend on the inputs. In the proposed system, four classes of vulnerabilities, which are DOS (C1), unconditional external call (C2), reentrancy (C3), and transaction origin (C4), can be predicted.

This prepared data set is given to ensemble-based XGBoost classifiers to train the network, where it uses internal K-Fold cross-validation (KF-CV) to test and improve its training performance. Cross-validation means an evaluation of machine learning models on a small sample of data. Cross-validation is mostly utilized to evaluate the skills of a machine learning model on invisible data in applied machine learning. A K-Fold CV is a collection of K sections/folds where each fold is utilized as a test set at a certain moment. Consider the case of a 5-fold CV, where K=5. In this case, the total data set is divided into five equal partitions. First, the first partition acts as a test set and the remaining partitions act as training sets. In the next iteration, the 2nd partition acted as the testing set and the remaining partitions acted as training sets. This process will continue for all five partitions, and finally, the results of all folds are averaged to predict the model's final performance.

XGBoost stands for Extreme Gradient Boosting. It is a scalable and tree-based boosting machine learning model. It is a popular and efficient open-source implementation. Gradient boosting is a learning method that tries to forecast a target variable by integrating estimates for several weaker and simpler models. In

Ensemble Learning, XGBoost is included in the boosting methods group. Ensemble learning consists of a group of predictors that offer higher prediction accuracy through several models. In gradient-boosted algorithms, the loss function is optimized, unlike in other booster techniques where incorrectly categorized branch weights are raised. XGBoost is a sophisticated gradient booster implementation with certain regulatory features. XGBoost features include, that it can be executed both on single and distributed systems (Hadoop, Spark) (regression and classification problems), parallel processing support, optimization of cache, and effective memory management for big data sets over RAM. It contains many regularizations that help to reduce overfitting problems. Auto tree pruning means the decision tree does not increase further internally beyond specified limitations, can manage any missing information, has cross-validation integrated, and takes care of some outliers.

As illustrated in Figure 5 in the XGboosting, mistakes caused by earlier models may be rectified by successive models. The trees will be created in sequence to minimize the mistakes of the previous tree in every successive tree. The previous tree lists each tree and updates the remaining bugs. The successive trees in the series will thus find information from an updated residual version. Gradient improvement is only a framework into which any model may be plugged, but tree-based models offer superior results.

```
----------------------------------------------------------------
Algorithm: Dataset_Preparation(contract.sol)
----------------------------------------------------------------
Input      : Smart_Contract_Program (contract.sol)
Output     : Generate ngram_Vector[N1, N2, N3, N4, N5, N6, N7]

//ngrams_initialization
n1=if(*()) // here * indicates any character
n2=addr.send(), n3=msg.sender.call.value, n4=[msg.sender]=0
n5=tx.origin, n6=if(*LV*) //if condition with large value(>382)
n7=while(*LV*) //while condition with large value(>382)

Function Feature_Extraction( Path/contract.sol )
     Remove comments in contract.sol file using solidity parser
     //split contract program into list of words using space delimiter
     cwords=contract.split()
     //Initialize ngram_vector with all zeros
     [N1, N2, N3, N4, N5, N6, N7, C1, C2, C3, C4]=0
     if n1,n6,n7 present in cwords
          set N1, N6, N7, C1 with 1 //DOS_Vulnerability
     if n2 present in cwords
          set N2, C2 with 1 //Unchecked_External_Call Vulnerability
     if n3, n4 present in cwords
          set N3,N4,C3 with 1 //Re-entrancy Vulnerability
     if n5 present in cwords
          set N5,C4 with 1 //Tx.origin Vulnerability
     add ngram_vector[N1,N2,N3,N4,N5,N6,N7,C1,C2,C3,C4] into
                                            dataset as one record
end function
//Repeat above function for every smart contract, to prepare final data set.
----------------------------------------------------------------
```

Fig. 4. Algorithm to generate data set from given smart contracts

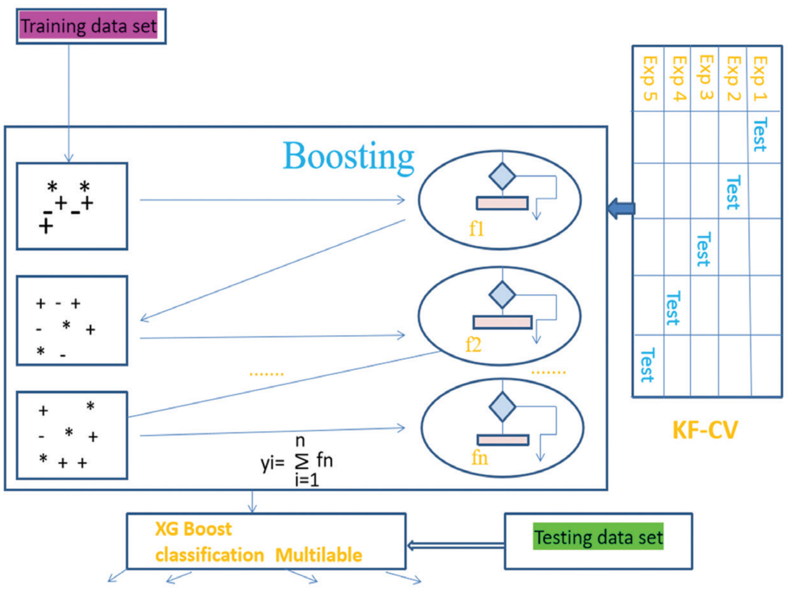


Fig. 5. XGBoost Classification Model

Once a machine learning model gets trained well, then it will classify or predict the smart contract vulnerabilities if we supply any SC as input for it. As smart contracts are written in programming languages, all stakeholders of a business can't interpret them to make them conform to whether they meet all business requirements or not. Hence, this paper used the Business Process Modeling Notation (BPMN) tools [25,26] to construct a graphical representation of a smart contract, as these BPMN diagrams are easy to understand by everyone, even if they do not have any technical knowledge, as shown in Figure 6. It includes functions for withdrawal, deposit, and balance inquiry. This BPMN diagram can be converted into a smart contract template by the BPMN-SOL tool. It is a compiler to convert a BPMN file to a solidity file [23,24]. The BPMN-SOL tool internally follows the caterpillar engine to convert a given smart contract into a solidity code template [18]. The output of the BPMN-SOL tool is the SC template as shown in Figure7 (sample code, it is not generated, we are still working on it). The SC template can't be processed by the solidity compiler. Hence, developer involvement is required to make slight changes to convert it into a final smart contract as shown in Figure 8.

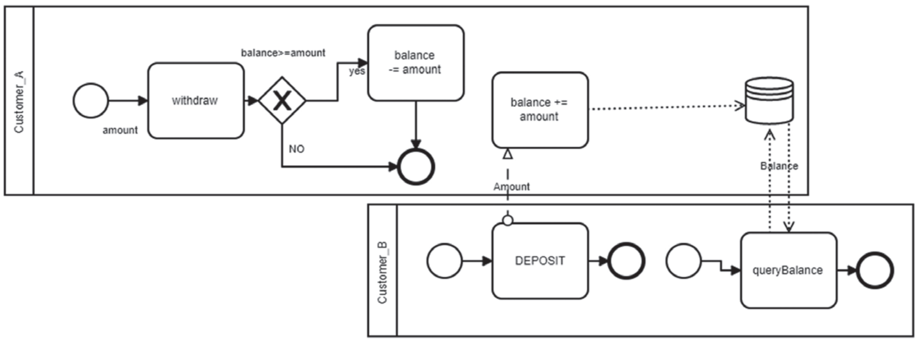
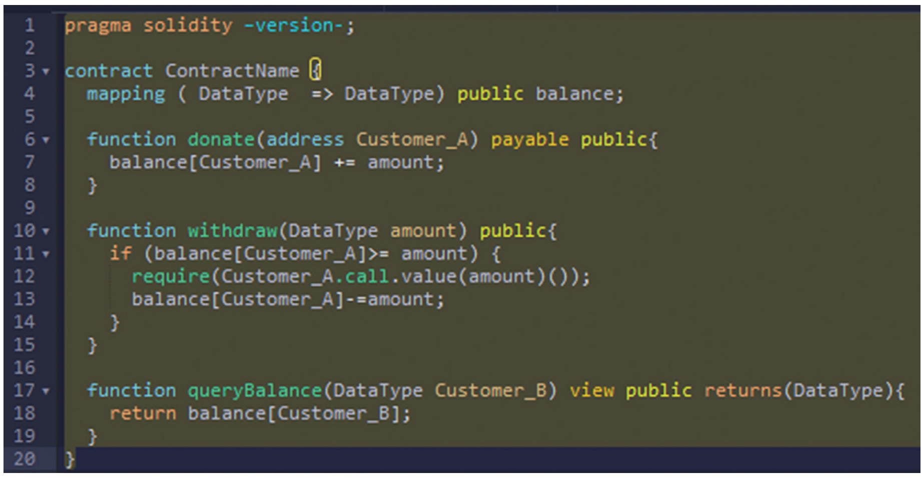
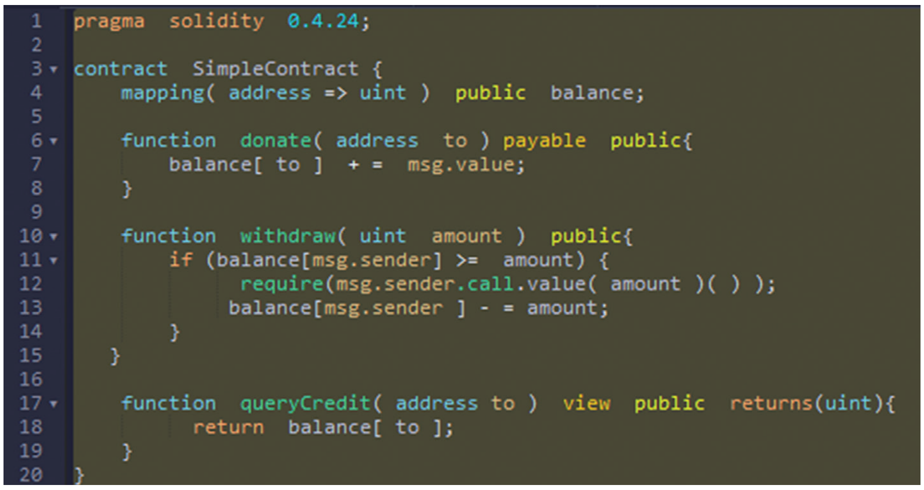


Fig. 6. BPMN diagram for Withdraw, Deposit, and Balance Enquiry functions



```
pragma solidity -version-;

contract ContractName {
  mapping ( DataType  => DataType) public balance;

  function donate(address Customer_A) payable public{
    balance[Customer_A] += amount;
  }

  function withdraw(DataType amount) public{
    if (balance[Customer_A]>= amount) {
      require(Customer_A.call.value(amount)());
      balance[Customer_A]-=amount;
    }
  }

  function queryBalance(DataType Customer_B) view public returns(DataType){
    return balance[Customer_B];
  }
}
```

Fig. 7. Sample Smart Contract Template for the given BPMN diagram (Fig. 6)



```
pragma  solidity  0.4.24;

contract  SimpleContract {
    mapping( address => uint )  public  balance;

    function  donate( address  to ) payable  public{
        balance[ to ]  + =  msg.value;
    }

    function  withdraw( uint  amount )  public{
        if (balance[msg.sender] >=  amount) {
              require(msg.sender.call.value( amount )( ) );
             balance[msg.sender ] - = amount;
        }
    }

    function  queryCredit( address to )  view  public  returns(uint){
          return  balance[ to ];
    }
}
```

Fig. 8. Final Smart Contract after modifications to SC Template

This paper concentrated on eXtreme Gradient Boosting (XGBoost) to build a suggested system to discover vulnerabilities in smart contracts based on code snippets and n-grams. For comparison purposes, this paper also used Random Forest (RF), K-Nearest Neighbor (KNN), and Support Vector Machine (SVM) machine learning models.

Random Forest is a flexible and easy-to-use algorithm for classification tasks. It is a kind of supervised learning algorithm. Random forests may construct many trees from randomly selected data samples. Its final classification prediction result is based on the majority voting results of trees constructed by it. The robustness of this model depends upon the number of trees constructed by it. Even though the random forest is a good model, compared with gradient-boosted trees, it has lower accuracy. SVM is the extensively utilized classification process, and it aims to identify a hyperplane in positive or negative samples in each of the segments such that there is the highest margin for the two segments, where the classification system is very reliable and generalizes new samples. The KNN classification algorithm is also used frequently. It is efficient and straightforward. For the given test sample, k-samples nearest to the test sample are determined based on a certain distance measure, and then information from k-neighbors is predicted. In k-samples, the most common category marks are chosen as the prediction outcomes, depending on the majority voting.

4. EXPERIMENT DETAILS AND RESULTS

The Keras Python package is being used to experiment with deep-learning models. Keras is a user-friendly, free, and useful framework to create and evaluate deep learning models with a few code lines. The COLAB resource is helpful for executing all the Python programs needed for this task online without cost. First of all, we must upload the data sets to the Google Cloud (colab) before the applications are run.

Metrics[29]: The matrices used in the proposed work are confusion matrix, recall, precision, F1 score, Micro-F1, and Macro-F1 to measure the deep learning models (XGBoost, SVM, RF, and KNN) performance for multi-label classifications. The metric types used will direct us to choose better machine learning models. To assess the performance of multi-label classification, the useful measures are Micro-F1 and Macro-F1. Micro-F1 and Macro-F1 are called Global-F1 and Average-F1, respectively. These measures can be calculated from the confusion matrix. It involves variables TN, FN, FP, and TP, which stand for True Negatives, False Negatives, False Positives, and True Positives, respectively.

***Precision** = TP / (FP + TP);*

***Recall** = TP / (FN + TP);*

***F1-SCORE** = (2 ∗ Precision ∗ Recall)/(Precision + Recall)*

Both the Micro F1-score and the Macro F1-score are used to evaluate the performance of multi-label binary problems. For both, the best value is 1 and the worst value is 0. The Micro-F1_score is defined as the harmonic mean of precision and recall. The Micro-F1_score measures the aggregated F1_score of all classes. Note that precision and recall have the same relative contribution to the F1_score. It can give high values even if the model is performing poorly on the rare labels since it gives more importance to the frequent labels. For the calculation of the micro-averaging F1-score, initially compute the sum of all false positives, true positives, and false negatives over all the labels. Then calculate the global precision and global recall from these sums. Finally, compute the harmonic mean to obtain the micro F1-score.

***Micro-F1_Score** = (2∗ Global Precision ∗ Global Recall)/ (Global Precision + Global Recall)*

Macro-F1_score will treat all classes equally. It will give a low value for the models that only do well in the frequent classes while showing unsatisfactory results in the rare classes. Macro F1-averaging is calculated by computing the F1-score for each class and then averaging them.

***Macro-F1_Score** = (2* Average Precision * Average Recall)/ (Average Precision + Average Recall)*

For the XGBoost classifier, all metrics are calculated as shown in Table 2. The dataset consists of a total of 1380 records, with 320, 330, 350, and 380 records for C1, C2, C3, and C4 respectively. The detailed calculation of Micro and Macro F1-scores of XGBoost classifier is shown below.

STP (Sum of all TP) = (110+180+215+175) = 680;
SFP (Sum of FP) = (5+5+7+9) = 26
SFN (Sum of FN) = (0+8+8+0) = 16;
Global Precision = STP / (STP+SFP) = 0.9631
Global Recall = STP / (STP + SFN) = 0.977;
Micro-F1 = 0.97;

Average Precision=(sum of all precisions)/4 = 0.9622; Average Recall=(sum of all Precisions) /4 = 0.9803; Macro-F1 = 0.971.

For comparison purposes, the above metrics are also calculated for other deep learning models (RF, SVM, and KNN) similarly to those calculated for XGBoost as shown in Table3. XGBoost classifier demonstrates good performance to detect smart contract vulnerabilities compared with RF, SVM, and KNN as shown in Figure 9.

The novelty of the proposed paper can be observed in Table 4, as it combines the two important features, which are contract snippets from [7,10] and the n-gram feature from [5], then an XGBoost ensemble model inspired by [5,20] to improve detection accuracy results and also work done towards auto SC generation inspired by [21,22].

Table 2. F1 Score calculation for XGBoost from confusion matrix elements.

XG Boost	TP	FP	FN	TN	Precision	Recall	F1-Score
C1	110	5	0	205	0.9565	1	0.9777
C2	180	5	8	137	0.9729	0.9574	0.9651
C3	215	7	8	120	0.9684	0.9641	0.9662
C4	175	9	0	196	0.9510	1	0.9749

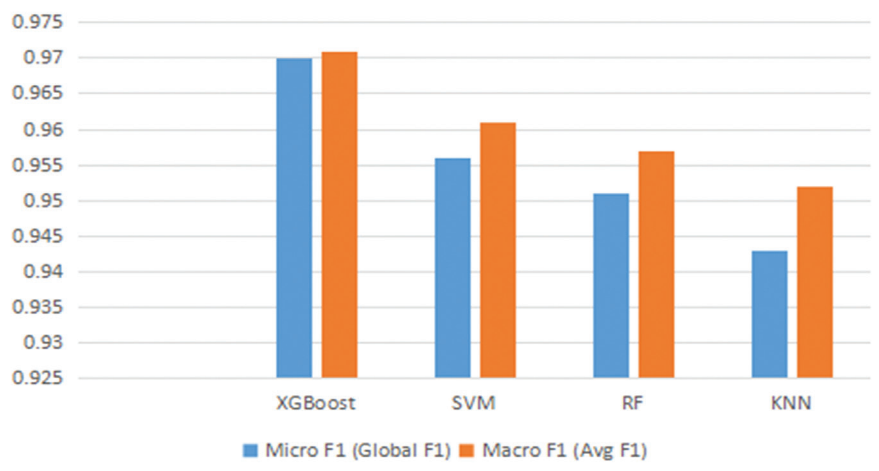


Fig. 9. Performance Comparison graph

Table. 3. Micro-F1 and Macro-F1 calculation for Machine Learning Models.

ML MODEL	FI-SCORE					
	DOS (C1)	Unchecked Exception (C2)	Reentrancy (C3)	Tx_Origin (C4)	Micro_F1 (Global F1)	Macro_F1 (Avg F1)
XGBoost	**0.977**	**0.965**	**0.966**	**0.974**	**0.97**	**0.971**
RF	0.962	0.965	0.944	0.921	0.951	0.957
SVM	0.951	0.947	0.962	0.954	0.956	0.961
KNN	0.942	0.956	0.937	0.968	0.943	0.952

Table. 4. Comparing proposed model with Existing Papers

	Vulnerabilities Detection	Features Used	ML Model Used	Work Done Towards Auto SC Generation
Liu Zhenguang et al [10]	Reentrancy	Contact Snippets	Bi-LSTM	No
WeiWang et al [5]	Reentrancy, Timestamp, Overflow, Underflow, Callstack, TOD	Bi-gram	XGBoost	No
Peng Qian et al [7]	Reentrancy, Timestamp, Infinite Loop	Pattern (Snippets) Extraction, Graph construction	CNN	No
Zhang L [20]	Reentrancy, Timestamp, Overflow, Underflow, Callstack, TOD	Information Graph	Ensemble Learning	No
C. K. Frantz et al[21]	--NA--	ADICO-Solidity DSL	--NA--	YES
Luciano B et al [22]	--NA--	Petri net graphs	--NA--	YES
Proposed Model	Re-entrancy, DOS, Unchecked external call, Origin of Transaction	Contact Snippets, N-gram, BPMN-SOL Compiler	XGBoost	YES

5. CONCLUSION & FUTURE WORK

This paper proposed work towards auto-smart contract generation and smart contract vulnerability detection models to identify specific security-related vulnerabilities using the XGBoost multi-label classification model. As shown in Figure 9, the proposed XGBoost model produced a 2% better average F1 score (2% better results than RF and KNN, and 1% better results than SVM), Compared with RF, SVM, and KNN deep learning models, by combining the best two features (called contract snippets and n-grams) from the literature to prepare a data set for the XGBoost model to detect SC vulnerabilities, which are Denial of Service (DOS), Unchecked external call, Re-entrancy, and Origin of Transaction. This paper also makes use of BPMN and BPMN-SOL tools to initiate the work towards auto-smart contract generation. The limitation of this work is that the BPMN-SOL tool produces the smart contract template, where developers have to put effort into preparing the final smart contract. In our future work, we will concentrate on improving SC template quality, again trying to make it a fully automated conversion, and also try to improve SC vulnerability detection accuracy results using advanced deep learning concepts.

6. REFERENCES

[1] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, "A survey on consensus mechanisms and mining strategy management in Blockchain networks", IEEE Access, Vol. 7, 2019, pp. 22328-22370.

[2] A. Bruyn, Shanti, "Blockchain an introduction", University Amsterdam, 2017, pp. 1-43.

[3] A. P. Joshi, M. Han, Y. Wang, "A survey on security and privacy issues of Blockchain technology", Mathematical Foundations of Computing, Vol. 1, No. 2, 2018, pp. 121-147.

[4] Cointelegraph, https://cointelegraph.com/bitcoin-for-beginners/how-Blockchain-technology-works-guide-for-beginners#where-can-Blockchain-be-used (accessed: 2022)

[5] Wei. Wang, Song. Jingjing , Xu. Guangquan, Li. Yidong, Hao. Wan, Su. Chunhua, "ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts", IEEE Transactions on Network Science and Engineering, Vol. 8, No. 2, 2021, pp. 1133-1144.

[6] Liao. Jian-Wei, Tsai. Tsung-Ta, "SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testin", Proceedings of the Sixth International Conference on Internet of Things: Systems, Management and Security, 2019, pp. 458-465.

[7] P. Qian, W. Xun, "Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection", IEEE Transactions on Knowledge and Data Engineering, 2021. (in press)

[8] S. S. Gupta, O. Yew-Soon, "Learning Approach to Detecting Security Threats", Proceedings of ACM, New York, NY, USA, 2019.

[9] P. Momeni, Y. Wang, R. Samavi, "Machine Learning Model for Smart Contracts Security Analysis", Proceedings of the 17th International Conference on Privacy, Security and Trust, Fredericton, NB, Canada, 26-28 August 2019.

[10] L. Zhenguang, H. Qingming, "Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models." IEEE Access, Vol. 8, 2020, pp. 19685-19695.

[11] W. J.-W. Tann, X. J. Han, S. S. Gupta, O. Yew-Soon, "Towards Safer Smart Contracts: A Sequence Learning Approach to Detecting Security Threats", Proceedings of ACM, New York, NY, USA, 2019.

[12] B. Mueller, "ConsenSys/Mythril", http:// github.com/ ConsenSys/mythril (accessed: 2020)

[13] O. López-Pintado, B. García-Bañuelos, M. Dumas, I. Weber, A. Ponomarev, "Caterpillar: A Business Process Execution Engine on the Ethereum Blockchain", Software: Practice and Experience, 2018, pp:1–45.

[14] M. Feng, Wang. Zhuoyi, "VSCL: Automating Vulnerability Detection in Smart Contracts with Deep Learning", Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency, Sydney, Australia, 3-6 May 2021.

[15] K. L. Narayana, K. Sathiyamurthy, "Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning", Materials Today: Proceedings, 2021. (in press)

[16] G. Chhaya, N. S. Gill, P. Gulia, "SSDT: Distance Tracking Model Based on Deep Learning", International Journal of Electrical and Computer Engineering Systems, Vol. 13, No. 5, 2022.

[17] M. R. Isa, M. A. Khairuddin, "SIEM Network Behaviour Monitoring Framework using Deep Learning Approach for Campus Network Infrastructur", International Journal of Electrical and Computer Engineering Systems, 2021.

[18] S. A. Baharudin, A. Lajis, "Deep Learning Approach for cognitive competency assessment in Computer Programming subject", International Journal of Electrical and Computer Engineering Systems, 2021.

[19] L. Yiping, X. Jie, C. Baojiang, "Smart Contract Vulnerability Detection Based on Symbolic Execution Technology", Proceedings of the China Cyber Security Annual Conference, 2021, pp. 193-207.

[20] L. Zhang, J. Wang, W. Wang, "A Novel Smart Contract Vulnerability Detection Method Based on Information Graph and Ensemble Learning", Sensors, Vol. 22, No. 9, 2022.

[21] C. K. Frantz, M. Nowostawski, "From Institutions to Code: Towards Automated Generation of Smart

Contracts", Proceedings of the Workshop on Engineering Collective Adaptive Systems, co-located with SASO, Augsburg, 2016.

[22] B. Luciano, A. Ponomarev, "Optimized Execution of Business Processes on Blockchain", arXiv:1612.03152v1, 2016.

[23] BPMN-SOL Compiler, https://github.com/signavio/ BPMN-Sol (accessed: 2022)

[24] BPMN-SOL Compiler, https://github.com/shaunazzopardi/bpmn-to-solidity (accessed: 2022)

[25] BPMN, https://www.visual-paradigm.com/guide/bpmn/what-is-bpmn/ (accessed: 2022)

[26] BPMN, https://www.lucidchart.com/pages/bpmn. (accessed: 2022)

[27] Smart Contract Dataset, https://swcregistry.io/docs/ SWC-107#modifier-reentrancy-fixedsol (accessed: 2022)

[28] Smart Contract Dataset, https://github.com/smartbugs/ smartbugs (accessed: 2022)

[29] Classification-loss-metrics, https://peltarion.com/knowledge-center/documentation/evaluation-view/ classification-loss-metrics/f1-score (accessed: 2022)

[30] Ethereum Smart Contract Best Practices, https://consensys.github.io/smart-contract-best-practices/ known_attacks/ (accessed: 2022)