Materials Today: Proceedings xxx (xxxx) xxx

Contents lists available at ScienceDirect

Materials Today: Proceedings

journal homepage: www.elsevier.com/locate/matpr

# Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning

K. Lakshmi Narayana [*], K. Sathiyamurthy

*Department of CSE, Pondicherry Engineering College, Puducherry, India*

ARTICLE INFO

*Article history:*
Available online xxxx

*Keywords:*
Blockchain
Ethereum
Smart contracts
Solidity
Vulnerabilities
Deep learning models

ABSTRACT

Smart contracts are a vital component of applications being built by blockchain or distributed ledger technology. Smart contracts consists of computer code that composed set of rules agreed upon by the involved parties. When these predefined conditions or rules are satisfied by the transactions initiated by parties, the smart contract executes itself to complete the transaction. Normally, while performing transactions with agreements involving multiple parties, a third party have to be involve to verify all the information, which makes it complex and time consuming process. Smart contracts simplify this process by eliminating the third party and automating the process, enabling the stakeholders to perform transactions directly with each other. Smart contract itself is replicated among multiple nodes of a blockchain there by giving benefits of immutability, security and permanence. Most smart contracts are written in Solidity programming language due to its simplicity and conciseness. Attackers are attracted by the popularity of Solidity language and its vulnerability possibilities. For example, in the 2016 year, 60 million US dollars was theft by Decentralized Autonomous Organization (DAO) attack due to vulnerabilities present in smart contracts. There are few existing tools and papers on this area to find vulnerabilities on smart contracts but taking more time to predict. Thus, this paper presents different Deep learning techniques with satisfactory results to identify smart contract vulnerabilities which are Re-entrancy, Denial of Service (DOS) and Transaction Origin using binary, multi class and multi label classification techniques.

## 1. Introduction

Blockchain is the key technology for crypto currencies and it consists of sequence of blocks called public ledger. Each block in a Blockchain connected back with previous one by storing hash value of the previous block. If anybody do any small change in any block, then hash value of that block will change, ultimately link will be disconnected with next block, in such a way that Blockchain satisfying immutability property. Each transaction which is taking place in blockchain will be verified by majority of participants in the corresponding blockchain network. Blockchain technology is initially proposed by a unknown person or group of individuals named 'Satoshi Nakamoto' in the year 2008. Blockchain records all conformed transactions in the database named ledger which is distributed to all participants over the blockchain network, making it satisfying immutable property of blockchain technology. Smart contracts (SC) consists of computer code that constitutes a set of rules agreed upon by the concerned parties. When these pre defined rules or conditions are fulfilled, the smart contract executes itself and provides the output. Normally, while executing agreements involving various stakeholders, a third party audits all the information which makes it a complex and time-consuming process. Smart contracts simplify it by eliminating the third parties like brokers, lawyers, bankers and automating the process, enabling the stakeholders to directly transact with each other. Smart contracts give the freedom and convenience to perform transactions with interested parties from all over the world without worrying about trustworthy versifiers or middlemen. The in built encryption mechanisms present in Blockchain made smart contracts secure, trust and tamper proof process.

Initially Blockchain was developed for Bitcoin crypto currencies, later people used to apply this technology for different Peer to peer applications. Another popular blockchain platform is Ethereum, it uses Ether crypto currency. The main idea of Ethereum is separation of contract layer and blockchain layer, so that it provides more

[*] Corresponding author.
*E-mail address:* kodavali.lakshmi@pec.edu (K.L. Narayana).

https://doi.org/10.1016/j.matpr.2021.04.125
2214-7853/© 2021 Elsevier Ltd. All rights reserved.
Selection and peer-review under responsibility of the scientific committee of the International Virtual Conference on Sustainable Materials (IVCSM-2k20).

flexible development environment than Bitcoins. All initiated transactions by parties must be satisfying the specified constraints mentioned in smart contract to complete transaction, otherwise transaction will be rejected. The main aim of smart contracts are to provide better secure transactions than traditional contract laws with low transaction fee. Smart contracts can be generally used for simple money related transactions like sending crypto currency from X to Y. Later they can also be used for registering any kind of ownership, access control for shared documents and property rights. Combination of Blockchain and Smart contracts enables people to work with other parties or organizations where no trust among them. As smart contracts are written by people, there are chances of vulnerabilities in the code. Attackers are get benefit of crypto currency transfer from other accounts to their accounts by making use of these code vulnerabilities. Hence it is worth to concentrate on efficient detection of smart contract code vulnerabilities before embedding into Blockchain. Now a days Blockchain technology attracts many other domains such a land registration, courier services, education, IOT, Assets management, Car and shoe companies etc. Organization of this remaining paper is, Section 2 give smart contract vulnerabilities, Section 3 explains related work on smart contract vulnerabilities in literature, Section 4 demonstrates proposed architecture and its explanation, Section 5 illustrates experimental results, finally conclusion & future work will be in Section 6.

## 2. Smart contract vulnerabilities

Attackers are concentrating on smart contracts because of broadly three reasons: first smart contracts in Ethereum mostly deal with money related transactions, second, it is impossible to modify vulnerable SC once it is deployed into blockchain, finally smart contracts are not having any evaluation metrics for determining quality of smart contract [1]. Many smart contract attacks were identified from 2016 which leads to huge loss of money (multi million dollar losses) due to vulnerabilities present in SC's. As smart contracts in Ethereum deals mostly with financial issues such as money transfers and more complexity of code, it is worth full now to concentrate on automated deep learning models for efficient detection of SC vulnerabilities [2].

Smart contract vulnerabilities are classified into 4 groups which are 1. Security issues, 2. Functional issues, 3. Operational issues, 4. Developmental issues [3]. Security issues consists of Re-entrancy, Denial of Service (DOS) by external contract, using tx.origin, unchecked external call and send() instead of transfer() vulnerabilities. Functional issues consist of integer division, locked money, integer overflow and underflow, time stamp dependency and unsafe type interface. Developmental issues consists of violation of token API, private modifier, not fixing compiler version, style guide violation, redundant fall back function and implicit visibility level. Finally operational issues consists of byte array and costly loop vulnerabilities. This paper mainly focuses on security issues which are reentrancy, DOS and transaction origin. Detailed explanation about these security issues are described in the Section 4.

## 3. Related work

In the literature few SC Analysis Techniques were proposed which are formal methods, static code analysis and software testing to detect security issues of smart contracts. Software testing allows us to verify the behavior of a program for given inputs. Each execution path of a program can be verified with single test, however verifying multiple paths in a program with single test is usually difficult task.

The static analysis can able to find the vulnerabilities in program without executing it, however, they has been taking more time. There are few existing tools of static code analyzers to detect SC vulnerabilities [4–8]. To detect SC vulnerabilities, time taken by the existing tools Mythril, Oyente and Securify are around 60, 28, and 18 s respectively. Formal methods are very complex to apply because they are depend on the mathematical properties of corresponding functions to detect vulnerabilities [9]. In addition these methods can only be used after the development of a SC is complete. Formal verification approaches are applicable to the small scale smart contracts with complex functional design [4]. There are few challenges to detect SC vulnerabilities which are 1) No static code analyzer or tool that can able to detect all vulnerabilities because each analyzer is concentrating on a few vulnerabilities only and they adopt symbolic analysis to walk through all executable paths in a contract. 2) Solution to this first challenge is to use multiple analzers to detect all vulnerabilities, however it requires more processecing time and computation power. 3) Deciding to use which analyzers is also challenging work. 4) Developers feel difficult to detecting vulnerabilities from huge number of SC using different analyzers.

Deep Learning models are updated models to Artificial Neural Networks (ANN) and they are dealing with developing much larger and more complex neural networks with very large data sets of labelled analog data, such as text, image, video and audio. Deep learning models are classified into supervised and unsupervised models. Supervised models include Classic / ANN, Recurrent NN (RNNs), Convolution NN (CNNs). Unsupervised models include Auto encoders, Boltzmann Machines and Self organizing maps (SOM).

As an overview in the literature few papers [4–8,10–13] have been proposed and they have been using either formal verification methods or statistical methods or dynamic methods and they require predefined patterns of vulnerabilities which are identified by domine experts. Other papers [1,2,14–18] were published to detect SC vulnerabilities using deep learning techniques, but they have been taking existing tools support to prepare datasets, hence taking more time to predict vulnerabilities. All existing deep learning papers are trying to extract features from opcode or assembly code of SC to prepare data sets but proposed framework extracting features from Abstract Syntax Tree (AST) itself, it is in the form of high level language and easy to process AST. By considering above issues in literature, proposed paper experimented with different deep learning techniques to predict SC vulnerabilities with less time and which does not dependent on neither any existing analyzers nor domain experts.

## 4. Proposed approach

Fig. 1 shows proposed model for vulnerability detection, it takes advantage of extracting features directly from SC source code to prepare training data set from AST generated by solidity parser. Solidity parser [19] is a open source tool, easy to install, run and it allows us to walk over AST through built in functions and dictionary methods. Once training data set is ready, then deep learning models could train on it and can able to classify new data to detect vulnerabilities within the less time. Proposed Architecture consists of mainly 6 steps which are 1. Data collection, 2. AST Generation, 3. Token Sequences, 4. Dataset Generation, 5. Applying deep learning models and 6. Experimental Results.

Smart contract source code is available from different sources of online [3,20,21]. One such source is Etherscan which allows us to explore and search for smart contracts, transactions, addresses, tokens, prices and other activities taking place on Ethereum blockchain. Generating Abstract Syntax tree (AST) can be done for col-

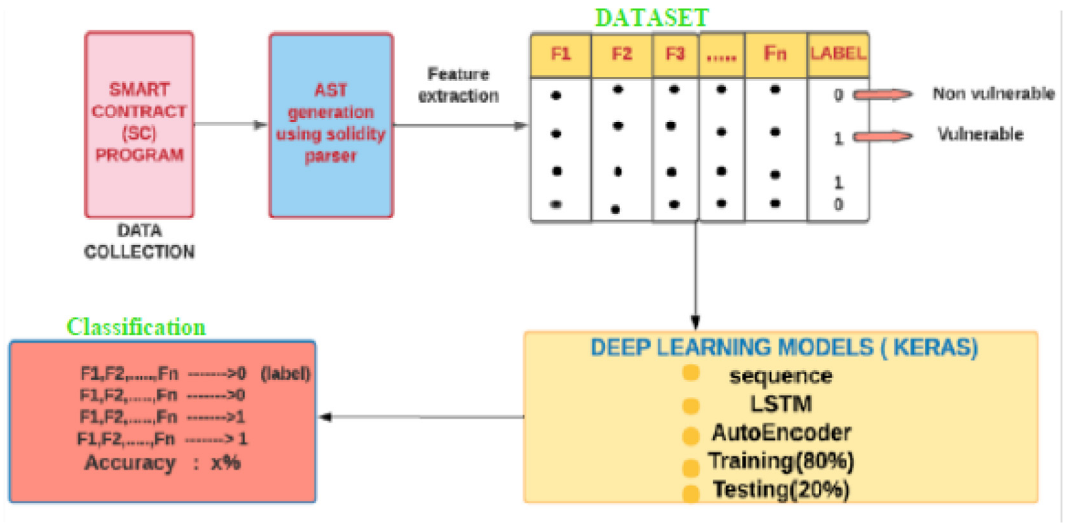

DATASET
SMART CONTRACT (SC) PROGRAM
DATA COLLECTION
AST generation using solidity parser
Feature extraction
F1
F2
F3
.....
Fn
LABEL
0
1
1
0
Non vulnerable
Vulnerable
Classification
F1,F2,......,Fn -------->0 (label)
F1,F2,......,Fn -------->0
F1,F2,......,Fn -------->1
F1,F2,......,Fn --------> 1
Accuracy : x%
DEEP LEARNING MODELS ( KERAS)
sequence
LSTM
AutoEncoder
Training(80%)
Testing(20%)

**Fig. 1.** Proposed Architecture.

lected smart contracts using solidity parser [19]. Solidity parser allows us to walk through AST to verify required variable values, keywords (if, while, for, ..) and function names easily through dictionary methods or predefined object methods.

Generation of dataset and vulnerability detection logic for SC vulnerabilities (Re-entrancy, DOS and Tx.origin) are discussed in detail as follows. All datasets and programs which have been used in this paper are available in github [22].

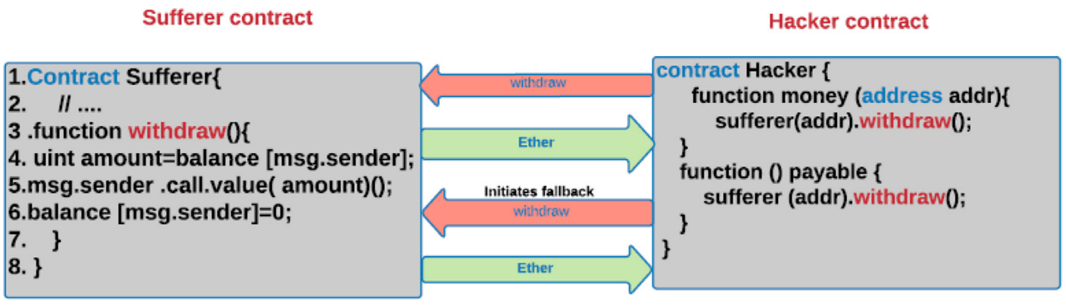
### 4.1. Reentrancy vulnerability

In a solidity smart contracts, there is a function called fallback which does not have name, does not take any arguments and does not return anything. This fallback function could forced to execute automatically from a SC in two cases 1) when transfer of Ether received by a SC, 2) when a contract called with no function matched. Fig. 2. having a real time example of two smart contracts (Sufferer and Hacker) with reentrancy attack possibility. The function money() from the Hacker contract, attempts to execute a withdraw operation by calling the withdraw() from contract Sufferer, which transfers the Ether to Hacker using the method (msg.sender.call.value()). Hacker contract as it received money, it forced the fallback() to execute. Fallback() is having withdraw() which inturn money will withdraw again from suffer contract, which initiates the fallback() second time from Hacker contract. This process continue until sufferer account balance is zero Ethers. We could not stop this flow of Ether transfer to Hacker account instead of simply watching. This is called reentrancy attack. We can avoid this attack by interchanging 5th and 6th lines in Sufferer Contract that is initializing balance variable with zero before sending amount to Hacker Contract. Now even fallback() from Hacker contract executes withdraw() present in Sufferer Contract, transaction will be fails since balance is zero.

The function call. value() sends amount to another smart contract. The function msg.sender.call.value () will calls the anonymous fallback function on caller (msg.sender) contract. Delegate call() also used to call external smart contracts. Now consider proposed dataset for reentrancy vulnerability detection as shown in Fig. 3. It consists of 7 attributes or feature, if we found these attributes in the smart contract then we mark these attribute values in dataset as either 1 or 2. One (1) means less impact, two (2) means more impact to consider as vulnerability. Detailed algorithm to prepare dataset for identifying reentrancy vulnerability as shown in Fig. 4.

### 4.2. Denial of Service (DOS) volunerability

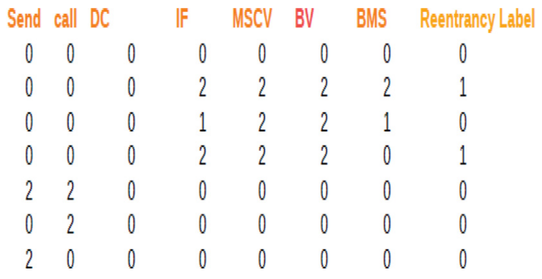
To execute smart contract functions, they requires a certain amount of gas (transaction fee), based on amount of computation involved in the function. The Ethereum network fixes a gas limit for each block and the sum of all transactions present in a block should not exceed that limit. Programming statements in a smart contracts which causes a Denial of Service (DOS) vulnerability, if it exceeds the gas limit while executing those statements. DOS vulnerability is possible to arise in four situations. 1) while or for condition involves a variable with value greater than 382 more or less is depending on gas limit fixed by a network. 2) when working with an array of unknown size. 3) when trying to empty an array



```
1.Contract Sufferer{
2.    // ....
3 .function withdraw(){
4. uint amount=balance [msg.sender];
5.msg.sender .call.value( amount)();
6.balance [msg.sender]=0;
7.    }
8. }
```

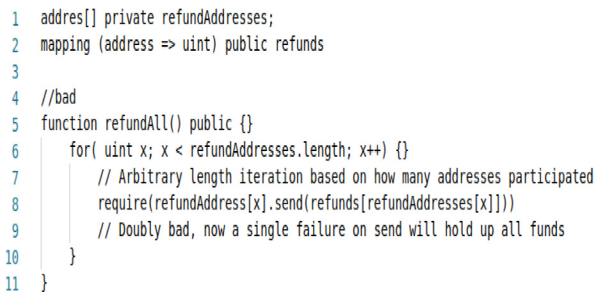
```
contract Hacker {
   function money (address addr){
      sufferer(addr).withdraw();
   }
   function () payable {
      sufferer (addr).withdraw();
   }
}
```

**Fig. 2.** Re-entrancy Vulnerable Smart Contract.



| Send | call | DC | IF | MSCV | BV | BMS | Reentrancy Label |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 2 | 2 | 2 | 1 |
| 0 | 0 | 0 | 1 | 2 | 2 | 1 | 0 |
| 0 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Fig. 3.** Dataset to Detect Re-entrancy Vulnerability.



```
addres[] private refundAddresses;
mapping (address => uint) public refunds

//bad
function refundAll() public {}
    for( uint x; x < refundAddresses.length; x++) {}
        // Arbitrary length iteration based on how many addresses participated
        require(refundAddress[x].send(refunds[refundAddresses[x]]))
        // Doubly bad, now a single failure on send will hold up all funds
    }
}
```

**Fig. 4.** Algorithm for Feature Extraction from Solidity Code.

of addresses. 4) Another example is as shown in Fig. 5, for loop is iterate to pay amount to all users. Fig. 5 shows that inside the for loop, it has used function require() to send money. The purpose of "require()" function to verify the condition and throw an exception if the condition is not satisfied and perform revert operation to reach to previous state. The problem here is that if payment to one user fails, then it will revert payments to all users which leads to for loop will never complete. Hence no user get paid due to transaction problem with one address. Based on above four conditions, DOS dataset can be prepared by considering the same procedure as done for Reentrancy Dataset.

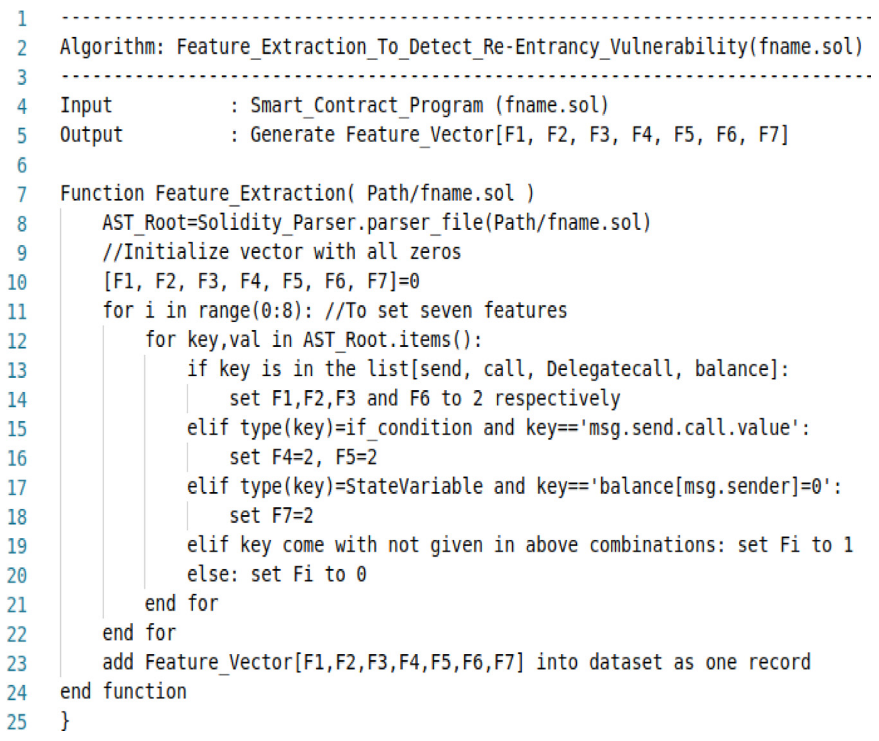
### 4.3. Transaction origin vulnerability

Solidity language having one global variable named "tx.origin", it gives the address of the account who initiated the transaction. For example consider in a call sequence X → Y → Z, from the Z point of view, tx.origin is X, and msg.sender is Y. The variable "tx.origin" which may leads to ambiguity some times, so better to avoid usage of "tx.origin" for authorization, instead preferable to use msg.sender. Hence in the SC code, if the variable tx.origin is found, then mark that attribute with 2, otherwise 0 in a dataset.

## 5. Experimental results

The proposed work uses Keras python library to experiment deep learning models. Keras is a open source, free, easy to use and powerful library for deep learning models development and evaluating with few lines of code [23]. Keras models are defined as sequence of layers. All deep learning programs used in this work are executed online using COLAB web resource which is used to execute all machine learning programs with free of cost. First we have to mount with google drive to access datasets which already have to be upload in our drive before executing the programs. To create a deep learning models, sequence of steps have to follow which are 1. Data loading, 2. Defining NN model, 3. Model compilation, 4. Model Fitting, 5. Model Evaluation, 6. Predicting the model.

Metrics: The proposed work used metrics which are accuracy, precision, recall, F1 score and confusion matrix to evaluate the performance of deep learning algorithms for classification. The type of the metrics used will influence more to decide or choose better deep learning models. The proposed work experimented with three classification techniques which are binary, multi class and multi label classifications. Binary classification is performed with



```
---------------------------------------------------------------------------
Algorithm: Feature_Extraction_To_Detect_Re-Entrancy_Vulnerability(fname.sol)
---------------------------------------------------------------------------
Input           : Smart_Contract_Program (fname.sol)
Output          : Generate Feature_Vector[F1, F2, F3, F4, F5, F6, F7]

Function Feature_Extraction( Path/fname.sol )
    AST_Root=Solidity_Parser.parser_file(Path/fname.sol)
    //Initialize vector with all zeros
    [F1, F2, F3, F4, F5, F6, F7]=0
    for i in range(0:8): //To set seven features
        for key,val in AST_Root.items():
            if key is in the list[send, call, Delegatecall, balance]:
                set F1,F2,F3 and F6 to 2 respectively
            elif type(key)=if_condition and key=='msg.send.call.value':
                set F4=2, F5=2
            elif type(key)=StateVariable and key=='balance[msg.sender]=0':
                set F7=2
            elif key come with not given in above combinations: set Fi to 1
            else: set Fi to 0
        end for
    end for
    add Feature_Vector[F1,F2,F3,F4,F5,F6,F7] into dataset as one record
end function
}
```

**Fig. 5.** DOS Vulnerability Smart Contract.

two deep learning models which are ANN and Auto Encoder. Details about these classification models which are number of layers, number of nodes in each layers, activation functions and metrics used are given in Table 1.

Proposed work tested binary classification models using accuracy which is the most popular metric to evaluate performance for classification models. Accuracy can be defined as the number of predictions correctly made out of all predictions. It can be calculate using confusion matrix with the help of variables FN, TN, TP, FP which stands for False Negatives, True Negatives, True Positives and False Positives respectively Formula: [Accuracy= (TP + TN) / (TP + FP + FN + TN)].

## 5.1. ANN model

The proposed work experiment with ANN and Auto Encoder deep learning models for binary classification to detect Reentrancy vulnerability. ANN model is supervised model but Auto Encoder is a unsupervised model. Learning a model involves mapping of input samples (Y) to an output label (Z), as Z = f(Y). Class variable is either 0 or 1. ANN model allows us to add layers one at a time until it able to train well. The number of input attributes (7 in the proposed work) determines the number of nodes in the inputs layer. The proposed work considered a fully connected neural network using the Dense class in Keras with 4 layers including input layer.

To train a network, best set of weights are mapped to inputs and outputs. After this we must specify loss function to evaluate weights. In the proposed work 'binary_crossentropy' is considered as loss algorithm and 'adam' is used as optimizer. Training will be performed over epochs and each epoch is divided into batches. Epoch means passing through once all of the records in the training dataset. Batch means one or more records will be considered in each epoch by the model before updating the weights. One or more batches are combined into an each epoch, depending on decided batch size. The proposed work considered 150 epochs and batch size with 10 and trained ANN model with entire dataset and considered same dataset for evaluating the performance of the model and got 100% accuracy.

## 5.2. Auto Encoder model

An Autoencoder is a neural network that encodes input into hidden representation called Encoding and tries to reconstruct its original input from hidden representation called Decoding. It's main objective is to minimize the reconstruction error between the input and the output. Initial step to detect anomalies using Autoencoder is dividing the dataset into training, testing and validation sets. Then remove the anomaly instances from training and validation set. Next perform encoding on training data (without anomalies) and decoding on test data. If the reconstruction error for the test data is above some threshold (Ex threshold_fixed = 0.006), then label it as an anomaly, otherwise normal. The proposed work uses binary classification to detect reentrancy vulnerability with help of Autoencoding deep learning model and got 100% accuracy. The classification results for both ANN and AutoEncoder programs are shown in Figs. 6 and 7 respectively.

## 5.3. Multiclass classification

Multiclass classification can also be referred as single-label classification or multinomial or polytomous or one-of classification, in which the class labels are mutually exclusive. Each input record must be in just one of the classes. In the proposed work, we classified as either Good(no vulnerability) or Reentrancy or Transaction Origin with class labels 0, 1 and 2 respectively. Initially we performed split on attributes as input variables (I) and output variables (O). Next step is to encode the output variable means reshape the output variable (O) from a vector to a matrix with a binary values 0 or 1 for each class to specify each record belongs to that class or not as shown in Table 2.

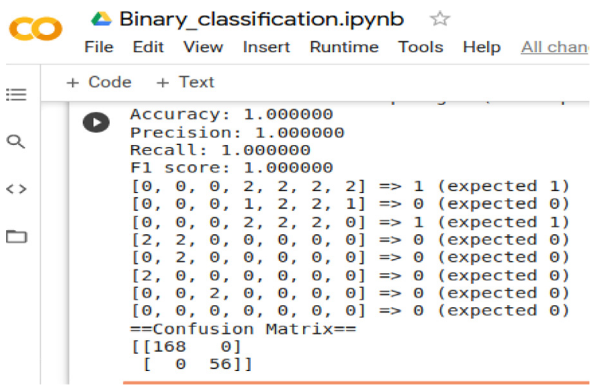
Then we considered a fully connected neural network model with one hidden layer that consists of 8 neurons and input layer with 8 inputs for 8 features. The hidden layer uses a "rectifier" activation function, the output layer have to include 3 nodes, one for each class. The output value with the largest will be considered as the class predicted by the model. The proposed model experiment with "softmax" activation function in the output layer since it gives output in the range 0 to 1 as we expects. Testing of this model is performed using K-Fold Cross Validation in which given data set is divided into a K-number of parts or sections, where each part is used as a testing set at some point. This process is continue till each fold/part have been used in the testing set and finally it makes average of all testing results using mean and standard deviation metrics and our experimental results shows that all records are classified correctly as shown in Fig. 8.

## 5.4. Multi-label classification

In multi-label classification, it could predict more than one class label for each record in the dataset i.e class labels are not mutually exclusive. It can also be referred as multiple label classification for short. The proposed work can predict three classes of vulnerabilities such as Reentrancy(C1), Transaction Origin(C2) and DOS(C3), hence three nodes are required for output layer. Each node in the output layer have to use the sigmoid activation function since it has to predict a probability of class membership between 0 and 1 for each class. The dataset considered for this classification consists of14 inputs or features and 3 outputs or classes, hence the neural network model constructed with 14 input nodes in the input layer and 3 nodes in the output layer as shown in Fig. 9. The hidden layer consists of 20 nodes that were chosen after some trial and error. This model is evaluated using repeated K-Fold CV

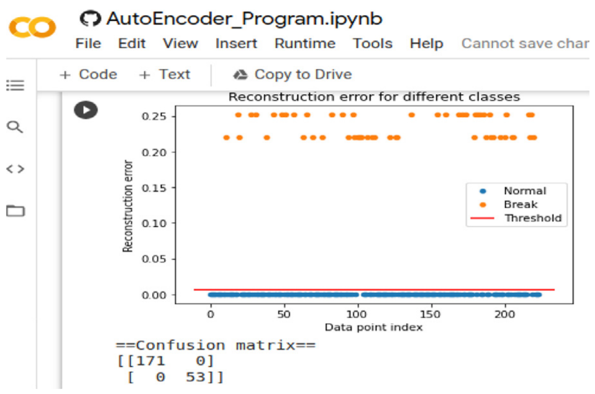
**Table 1**
Deep Learning Network Attributes Vs DL Models & Classifiers.

| DL Network Attributes Vs Models/ Classificat | #Nodes in the Input Layer | #Nodes in the Hidden Layer | #Nodes in the Output Layer | Activation Function Used | Loss Function | Optimizer Algorithm Used | Metrics Used | Accuracy Results |
|---|---|---|---|---|---|---|---|---|
| ANN | 7 | Two Hidden layers with **12, 8 Nodes** respectively | 1 | ReLU, sigmoid | binary_crossentropy | adam | Accuracy, F1_Score, Confusion Matrix | 100% |
| AutoEncoder | 7 | 32 | 1 | ReLU, linear | mean_squared_error | adam | Accuracy, Confusion Matrix | 100% |
| Multi Class Classification | 8 | 8 | 3 | Rectifier, softmax | categorical_crossentropy | adam | K-Fold CV with M & SD | 100% |
| Multi Label Classification | 14 | 20 | 3 | sigmoid | binary_crossentropy | adam | K-Fold CV with M & SD | 97.40% |


Binary_classification.ipynb
File Edit View Insert Runtime Tools Help All chan
+ Code + Text

```
Accuracy: 1.000000
Precision: 1.000000
Recall: 1.000000
F1 score: 1.000000
[0, 0, 0, 2, 2, 2, 2] => 1 (expected 1)
[0, 0, 0, 1, 2, 2, 1] => 0 (expected 0)
[0, 0, 0, 2, 2, 2, 0] => 1 (expected 1)
[2, 2, 0, 0, 0, 0, 0] => 0 (expected 0)
[0, 2, 0, 0, 0, 0, 0] => 0 (expected 0)
[2, 0, 0, 0, 0, 0, 0] => 0 (expected 0)
[0, 0, 2, 0, 0, 0, 0] => 0 (expected 0)
[0, 0, 0, 0, 0, 0, 0] => 0 (expected 0)
==Confusion Matrix==
[[168   0]
 [  0  56]]
```

**Fig. 6.** NN Binary Classification Results.

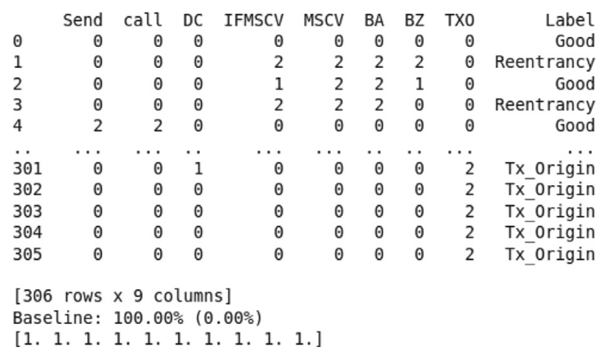

AutoEncoder_Program.ipynb
File Edit View Insert Runtime Tools Help Cannot save char
+ Code + Text Copy to Drive
Reconstruction error for different classes
Reconstruction error
Data point index
Normal
Break
Threshold
0.25
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200

```
==Confusion matrix==
[[171   0]
 [  0  53]]
```

**Fig. 7.** AutoEncoder Classification Results.

**Table 2**
Label Encoding.

| Encoding Output Variable | | |
|---|---|---|
| Good | Reentrancy | Tx.Origin |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |



```
     Send  call  DC  IFMSCV  MSCV  BA  BZ  TXO       Label
0       0     0   0       0     0   0   0    0        Good
1       0     0   0       2     2   2   2    0  Reentrancy
2       0     0   0       1     2   2   1    0        Good
3       0     0   0       2     2   2   0    0  Reentrancy
4       2     2   0       0     0   0   0    0        Good
..    ...   ...  ..     ...   ...  ..  ..  ...         ...
301     0     0   1       0     0   0   0    2   Tx_Origin
302     0     0   0       0     0   0   0    2   Tx_Origin
303     0     0   0       0     0   0   0    2   Tx_Origin
304     0     0   0       0     0   0   0    2   Tx_Origin
305     0     0   0       0     0   0   0    2   Tx_Origin

[306 rows x 9 columns]
Baseline: 100.00% (0.00%)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

**Fig. 8.** Multi Class Classification Results.

with fold = 10 and repeat = 3 values. Then the model will predict the three probabilities for each record and then rounding the values to either 0 or 1 for each class. Classification accuracy can be calculated by collecting scores across all repeats and folds and can be sum up by calculating the mean and standard deviation. Our experimental results show that 97.4% of records are classified correctly. The proposed work experimented with 3 types of classifications which are binary, multi class and multi label, and accuracy of these classifier results are shown in Fig. 10.

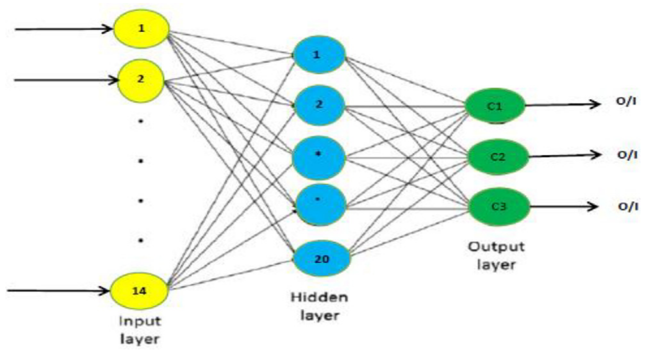


**Fig. 9.** Multi Label Classification Network Model.

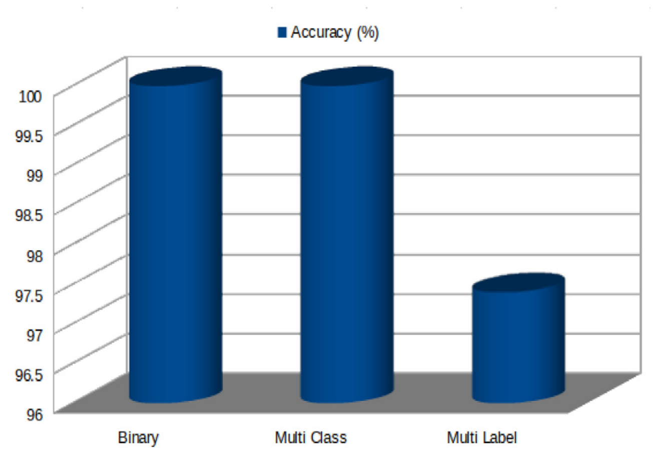


**Fig. 10.** DL Models Classification Results.

## 6. Conclusion & future work

This paper proposed different deep learning models to identify mainly security related vulnerabilities which are reentrancy, transaction origin(Tx.origin) and Denial of Service (DOS) using three classification techniques which are binary, multi class and multi label classifications. Proposed DL techniques are practically implemented and tested on datasets and giving satisfactory results i.e. more than 97% with less time. Right now in the proposed work, we have prepared quality datasets with thousand rows approximately and same dataset is considered for both training and testing purpose, so we have been getting better results. In future work we will increase dataset size and consider separate datasets for training and testing datasets with other deep learning models also which have not covered in this paper.

## CRediT authorship contribution statement

**K. Lakshmi Narayana:** Investgation, Methodology, writing-original draft. **K. Sathiyamurthy:** Investgation, Methodology, writing-original draft.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

[1] L. Jian-Wei, T. Tsung-Ta, SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testing, 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), IEEE, 2019, pp. 458–465.

[2] Wesley Joon-Wie Tann, Xing Jie Han, Sourav Sen Gupta, and Yew-Soon Ong. Towards Safer Smart Contracts: A Sequence Learning Approach to Detecting Security Threats. In Proceedings of ACM (Conference '19), 2019; ACM, New York, NY, USA, 12 pages.

[3] Veloso. (Aug. 08, 2018). smartbugs/dataset/. Accessed on: Nov 2, 2020. [Online]. Available: https:// github.com/smartbugs/smartbugs/ tree/master/dataset.

[4] B. Mueller (Sep. 17, 2017). ConsenSys/Mythril. Accessed on: Nov 2, 2020; [Online] Available: http://github.com/ConsenSys/ mythril.

[5] J Krupp and C.Rossow. teEther: Gnawing at Ethereum to automatically exploit smart contracts. In Proc. 27th USENIX Secur. Symp., Baltimore, MD, USA, 2018; pp. 1317–1333.

[6] Tsankov Petar, Andrei Dan, Dana Drachsler-Cohen, Securify: Practical Security Analysis of Smart Contracts, in: CCS '18: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 67–82.

[7] Feist Josselin, Gustavo Greico and Alex Groce. Slither: A Static Analysis Framework For Smart Contracts. IEEE 2nd International Workshop on Emerging Trends in Software Engineeringfor Blockchain (WETSEB), September 2019.

[8] Mark Mossberg, Manzano Felipe, Hennenfent Eric and Groce Alex. Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), Jan 2020.

[9] L Luu, DH Chu, H Olickel, P Saxena, and A Hobor. Making smart contracts smarter, in Proc. ACM CCS, Vienna, Austria, 2016; pp. 254–269.

[10] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, Y. Alexandrov. Smartcheck: Static analysis of Ethereum smart contracts,. In Proc. IEEE/ACM 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB), May/Jun 2018; pp. 916.

[11] Fu. Menglin, Wu. Lifa, A critical-path-coverage-based vulnerability detection method for smart contracts, IEEE Access 7 (October 2019) 147327–147344.

[12] Sarwar Sayeed, Hector Marco-gisbert. Smart Contract: Attacks and Protections. IEEE Access, January 2020.

[13] Yuan Zhuang, Liu Zhenguang. Smart Contract Vulnerability Detection Using Graph Neural Networks. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20), pp.3283-3290.

[14] Mimeni Pouyan, Yu Wang and Reza Samavi. Machine Learning Model for Smart Contracts Security Analysis. IEEE conference, 2019.

[15] Liu Zhenguang, He Qinming, Towards automated reentrancy detection for smart contracts based on sequential models, IEEE Access 8 (2020) 19685–19695.

[16] Shi Meilong, He Peng, Xiao Haitao, Li Huixin, and Zeng Cheng. An Approach to Semantic and Structural Features Learning for Software Defect Prediction. Hindawi, Mathematical Problems in Engineering, Volume 2020; Article ID 6038619, 13 pages.

[17] M S.Kumar, P. Neelima,"An efficient Blockchain-based IIoT System for Transparency and Concurency Mechanism", Journal of Critical Reviews, Vol 7 Issue 13, 2020.

[18] Anh Phan Viet, Nguyen Minh Le and Lam Thu Bui. Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction. IEEE International Conference on Tools with Artificial Intelligence (ICTAI), June 2018.

[19] Wei Wang, Song Jingjing, Xu Guangquan, Li Yidong Hao Wang and Su Chunhua. ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts. IEEE Transactions on Network Science and Engineering, 2019.

[20] Bond Federico. (Jun 16, 2017). solidity-parser-antlr. Accessed on: Nov 2, 2020; [Online]. Available: https://github.com/federicobond/solidity-parser-antlr/ blob/ master/README.md.

[21] 2020 SmartContractSecurity/SWC-registry, Accessed on: Nov 2, 2020; [Online]. Available: https://swcregistry.io/.

[22] Ethereum Smart Contract Security Best Practices. Accessed on: Nov 2, 2020; [Online]. Available: https://consensys.github.io/smart-contract-best-practices/known_attacks/

[23] LakshmiNarayana K. (Nov 02, 2020). klngithubsairam/Research. Accessed on: Nov 2, 2020; [Online]. Available: https://github.com /klngithubsairam/ Research.